IFC-Bank of Italy Workshop on "Machine learning in central banking"

19-22 October 2021, Rome / virtual event

Unsupervised outlier detection in official statistics[1]

Nhan-Tam Nguyen, Deutsche Bundesbank,
and co-authors from the Deutsche Bundesbank
and the German Research Center for Artificial Intelligence

[1] This presentation was prepared for the Workshop. The views expressed are those of the authors and do not necessarily reflect the views of the Bank of Italy, the BIS, the IFC or the central banks and other institutions represented at the event.

Unsupervised Outlier Detection in Official Statistics

Tobias Cagala Jörn Hees Dayananda Herurkar Mario Meier
Nhan-Tam Nguyen Timur Sattarov Kent Troutman Patrick Weber[*]

November 1, 2021

Abstract

This paper presents a summary of a joint project conducted by the Deutsche Bundesbank and the German Research Center for Artificial Intelligence (DFKI). As a joint use case in the area of financial micro data, we evaluate the performance of all major classes of unsupervised learning algorithms for outlier detection and implement a complete machine learning workflow. Our workflow extends beyond pre-processing the data and flagging outliers by incorporating explainable AI methods and a possibility for the algorithm to exploit feedback by domain experts. We apply our approach to micro data sets that are typically collected by a central bank in the Euro area and that cover the structure and format of a wide range of financial data, namely the interest rates statistics (MIR), the money market statistics (MMSR), the (sectoral) securities holdings statistics (SHS-S), and the investment funds holdings statistics (IFS). With our work, we contribute to both the improvement of the data quality management work done by official statistical departments as well as to the literature on applied machine learning.

[*]Cagala: tobias.cagala@bundesbank.de; Hees: joern.hees@dfki.de; Herurkar: Dayananda.Herurkar@dfki.de; Meier: mario.meier@bundesbank.de; Nguyen: nhan-tam.nguyen@bundesbank.de; Sattarov: timur.sattarov@bundesbank.de; Troutman: kent.troutman@bundesbank.de; Weber: patrick.weber@bundesbank.de. The paper represents the authors' personal opinions and does not necessarily reflect the views of the Deutsche Bundesbank or its staff.

Contents

1 Introduction

To meet the demand for timely provision of high-quality micro data in an environment of steadily rising data volumes, statistics departments of governmental organizations are increasingly turning to statistical learning methods from the fields of data science and machine learning (see for example Tissot et al. (2018)). The motivation is clear: these methods potentially promise higher process efficiency with the input of fewer (costly) human resources.

This paper presents a summary and lessons learned of a joint project conducted by the Deutsche Bundesbank and the German Research Center for Artificial Intelligence (DFKI). As a joint use case in the area of financial micro data, we evaluate the performance of all major classes of unsupervised learning algorithms for outlier detection and implement a complete machine learning workflow. Our workflow reflects the recursive nature of modern machine learning applications by extending beyond simple feature engineering and model estimation and into how to incorporate explainable AI methods and feedback by domain experts. We apply our approach to data sets that are collected by the Bundesbank and cover the structure and format of a wide range of financial data that include the interest rates statistics, the money market statistics, the sectoral securities holdings statistics, and the investment fund holdings statistics.

With our work, we contribute to the applied machine learning literature in two ways. First, we evaluate the performance of unsupervised learning algorithms in improving the data quality of micro data by flagging reporting errors that have characteristics of outliers. To this end, we collect reporting errors that domain experts (humans) detected in the past which gives us a unique labeled data set of errors and/or outliers that we can use to benchmark how well unsupervised methods recognize these errors. Second, we provide guidance on the implementation of all steps of an automated, unsupervised machine-learning pipeline that ranges from the pre-processing and selection of algorithms, to the application of explainable artificial intelligence (Explainable-AI) and active learning to enable and incorporate human feedback for official statistical data.

Our key findings are as follows: First, related to the performance of unsupervised algorithms, we show that most of the algorithms successfully isolate anomalous data points in micro data that were previously flagged in the data quality management (DQM) process by humans.[1] They achieve this without information on the labels by separating data points that deviate from the underlying structure of the data. In fact, we find that unsupervised algorithms can not only detect erroneous data points, but also hint at unusual data points and patterns that can further be analysed by data users. Second, we show that methods from the field of Explainable-AI provide domain experts with hints on how the models distinguish between anomalous and regular data points and can thereby inform business intelligence and allow statistics departments to issue more targeted DQM-related requests to reporting agents. Third, we address the challenge of incorporating the expertise of domain experts and reporting agents back into the production pipeline by implementing an active learning loop.

We conclude that unsupervised learning algorithms, applied to granular, financial data of the sort collected by a central bank, are not only suitable to detect incorrect reporting and thereby improve data quality, but that these methods can also detect unusual patterns in very heterogeneously structured data sets. However, our work also stresses that a production pipeline that is largely automated and that provides the possibility to actively incorporate (human) feedback is at least as important as a proper selection of algorithms.

The remainder of this paper is structured as follows: In Section 2, we summarize the different kinds of outlier detection methods, introduce measures for evaluating the performance of unsupervised outlier detection algorithms, describe our approach to counteract over-fitting, and describe the ecosystem in which we implemented the algorithms. In Section 3, we provide an overview of the data sets that we used for our study and descriptive statistics on outliers as well as the dimensions of each data set. Section 4 discusses pre-processing before running unsupervised algorithms, including how to deal with null values, how to handle categorical data, and how to scale data in the presence of dependencies. We further discuss the impact that feature

[1] Following Aggarwal (2015) an outlier or anomaly is a data point that is significantly different from the remaining data. In this paper, we use the terms outlier and anomaly interchangeably.

engineering can have on the performance of algorithms in this section. Next, we provide a broad conceptual overview of algorithms for the unsupervised detection of outliers in Section 5, including an outline of the strengths and weaknesses of each algorithm and their usefulness for detecting local versus global outliers. Section 6 moves away from single outlier detection methods to approaches that combine several machine learning algorithms into one single model, aiming at improving the performance of the final model. In Section 7 we provide an introduction to active learning. In Section 9, we open up the black box of unsupervised learning algorithms and introduce methods from the toolbox of Explainable AI. We close the section with an example on how explanations for outliers that were detected with autoencoders can provide novel insights to data producers and users alike. Section 8 shows cross-validated performance metrics for the detection of outliers in granular financial data sets with the approaches that we discussed in sections 5 and 6. Finally, Section 10 concludes this paper.

2 Background

2.1 What constitutes an outlier?

Detecting outliers or anomalies is a common data analysis task across various domains (e.g. health-care, quality assurance, financial data) with a variety of application scenarios (e.g. the identification of diseases, intrusions, mistakes, fraud, see e.g. Hodge and Austin (2004); Ahmed et al. (2016); Bhuyan et al. (2014)).

In general, outlier detection (OD) tries to solve a heavily imbalanced binary classification problem between a few points of interest (the true outliers) and the majority of other "normal" points (the true inliers). To solve the problem, outlier detection in general relies on the assumption that the true outliers can be distinguished from true inliers in the vector-/feature-space, e.g., by having a larger distance to their neighbors. Points showing such irregularities in feature-space are often called predicted outliers or simply outliers, while those similar to the majority are often called predicted inliers or simply inliers (Aggarwal, 2015; Aggarwal and Sathe, 2015; Chandola et al., 2009). It is worthwhile emphasizing the difference between "true outliers" and "predicted outliers": While the former are defined by experts, the latter are defined by distributions in feature-space. Whenever the distinction between the two is important, we will use the longer, more explicit names. In real world scenarios, the assumption that true outliers can be distinguished (easily or at all) from true inliers in feature-space is sometimes violated, leading to cases where true outliers can be predicted inliers and true inliers can be predicted outliers. What makes the detection of outliers an interesting use case is that an evaluation of these data points (the predicted outliers) might reveal certain patterns or problems (the true outliers) with a higher likelihood than when simply investigating a random sample of data points. This is particularly pertinent when the number of data points is prohibitively high and the fraction of true outliers is very low.

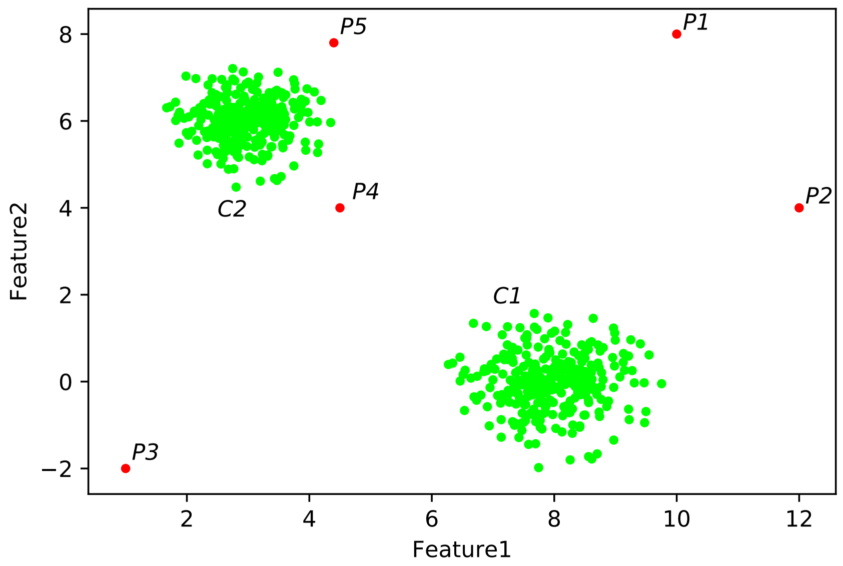
In the following, we will briefly describe the common sub-classes of outlier detection (also see Goldstein and Uchida (2016); Zhang et al. (2010)), based on the type of outliers of interest and the knowledge (if any) that is available about the true outliers and true inliers.

2.1.1 Global versus local outliers

Global outliers are data points which are classifed as anomalous to due to being (far) outside the overall distribution of the data set (Khoa and Chawla, 2010; Ernst and Haesbroeck, 2017; Dang et al., 2013; Goldstein and Uchida, 2016). An easy example for this class of outliers are points that are at least three standard deviations outside of an interval of an n-dimensional Gaussian that has been fitted to the whole data set. Outliers of this class are often data points that are orders of magnitude away from the others points and often caused by data entry mistakes.

In contrast, local outliers are points which are not anomalous on a global, but on a local scale (Khoa and Chawla, 2010; Ernst and Haesbroeck, 2017; Dang et al., 2013; Goldstein and Uchida, 2016). Such points deviate from the distribution/regularities of their local neighborhood. By this, local outliers are directly related to cluster analysis and account for the fact that many real world data sets can be better modelled as a composition of multiple distributions. Local outliers are then those data points which are close to such clusters, but which still behave different with respect to the local distribution of clustered points.

Figure 1: Example of local and global outliers in a 2D space



Notes: The figure shows an example for local and global outliers in a 2D space. C1 and C2 are two clusters of points. P1, P2, P3 represents global outliers, and P4, P5 represents local outliers with reference to cluster C2.

Local and global outliers are illustrated in Figure 1. While the detection of global outliers is often relatively trivial, it is typically much more difficult to detect local outliers, as such data points can reside well within the normal distribution limit of the data set. Unlike our illustration, real-world data sets typically have hundreds if not thousand of dimensions, making it challenging to find meaningful clusters and boundaries (Prieditis and Russell, 1995; Kriegel et al., 2005). Also, the algorithms that are optimized to find local outliers, often rely on a large variety of parameters to determine what is a “neighborhood” and what is “different”. Still, the detection of not only global, but also local outliers is desired in many application areas. An overview of a variety of outlier detection algorithms can be found in Section 5.

2.1.2 Available labels

Depending on the availability of knowledge (also called ground truth labels or simply labels) about data points, outlier detection use cases can be divided into 3 main groups: unsupervised, supervised, and semi-supervised (Chandola et al., 2009; Chalapathy and Chawla, 2019; Goldstein and Uchida, 2016). In order to evaluate the outcomes of any algorithm and to compute the evaluation measures presented in Section 2.3, one needs at least some labelled data independent of the following groups.

Unsupervised outlier detection Unsupervised outlier detection is the process of detecting outliers without data labels, but solely by using density or distance measures of the data samples (Chalapathy and Chawla, 2019; Aytekin et al., 2018; Goldstein and Uchida, 2016). In this case the detection algorithm can only rely on the intrinsic properties of the data in feature-space to distinguish the abnormal samples (outliers) from the ordinary data (inliers). However, due to the advantage of not relying on often difficult or costly to acquire labels of the true outliers or true inliers, unsupervised OD is often the first choice in any OD application.

Supervised outlier detection For supervised outlier detection data labels are essential. The data points (or at least a subset of all points) have to be labelled as either true outlier or true inlier. A supervised outlier detection method is essentially a (strongly imbalanced) binary classifier with the task to classify a given data point into either an inlier or an outlier. The labelled data set is divided into at least training and test set so that the supervised OD model can be trained on the training set and evaluated on the test set, in order to evaluate its generalization to unseen data. In application oriented use-cases this approach is the least preferred one, because it is often difficult (or costly) to acquire a large enough amount of labelled data. The imbalanced nature of the data also complicates the acquisition of labels, as simple random sampling approaches often lead to situations in which the true outlier class suffers from too few samples to be well represented. In general it is debatable if the class of true outliers can be (or should be) well represented with examples, as focusing on such representations might hinder the detection of completely novel outliers in the future. Hence, while helpful for the detection of micro-clusters of outliers, it is advisable to combine fully supervised algorithms with those of the other classes.

Semi-supervised outlier detection Unlike the supervised context, semi-supervised algorithms are only trained on the true inlier labels. The underlying idea is that such an algorithm should model normality by learning the distribution of features from true inliers. Everything sufficiently deviating from this normality is then labelled as an outlier (Chalapathy and Chawla, 2019). In terms of classification, this is also called one-class classification. In practical use-cases, the acquisition of true inlier labels is often much simpler than that of true outlier labels. Especially based on previous unsupervised OD and a human review of the resulting predicted inliers, a large set of true inliers can often be generated with minimal human effort, making semi-supervised algorithms promising for practical applications.

2.2 On the importance of train-test splits

Splitting data between a train set for model estimation and a test set for validation is common practice in machine learning. In supervised learning, the goal is to avoid estimating a model that provides a tight fit to the relationship between features and predicted labels in the training data, but, due to modeling spurious relationships that do not generalize, does not provide accurate predictions of the labels in the test data. However, in unsupervised or semi-supervised learning applications, an algorithm cannot overfit on the prediction of a specific label, therefore the question whether to split the data into a train and test sample is more subtle.

The argument for a train-test-split in an unsupervised context is that it can prevent the estimation of overly-complex separation frontiers. If, for example, an unsupervised algorithm learns to distinguish clusters of observations, an over-complex separation frontier would be unstable.[2] Because overly-complex separation frontiers are partly driven by random, rather than structural relationships in the data, a model that returns different separation frontiers depending on random draws from the input data most likely suffers from over-fitting. A resulting measure of over-fitting in unsupervised settings is the cluster stability of a model.

There are different ways to split data into a train and test sample and in many settings, a random split of the data is sufficient to create proper train and test data sets. Many, if not most, data sets collected by central banks have a panel structure. In the context of financial (panel) data, three aspects should be considered: (a) the time-dimension of the data (b) the group structure of the data (i.e. holdings of bank A and holdings of bank B) and (c) the rarity of the outlier label. The time dimension is most subtle because financial data often contains features that reflect values from previous periods or changes across periods. If we split the data into a train and test set along the time dimension, using the later periods for testing, there could be data leakage from the train into the test set if features are serially correlated. Another source of data leakage are features that allow the algorithm to model serial correlation by including data from different time periods. In time series or panel data, we might include first differences as features. In this case, if we split the data randomly into train

[2]In our example, unstable separation frontiers result in different clusters if we train the unsupervised algorithm on different random samples from the same population. Stability is a desirable feature of a separation frontier because it implies that the algorithm learned structural and not spurious relationships in the data.

and test sets, information that we use in the train set can appear (e.g. in its lagged realization) in the test set. This can result in data leakage between the train- and the test set.

In our data sets, empirically either using a random split or attributing all months below a threshold date t to the train set and all months above the threshold to the test set does not have an effect on our findings because the algorithms cannot exploit this link in the data. The fact that there might be time persistence of outliers across time does not affect this logic. However, if there is a structural break in the data at some point in time, the random split might be more stable and better to extrapolate. If one splits the data according to some time threshold, it might happen that data before the structural break are the train set and the rest in the test set. This could considerably affect the performance in the test data. In contrast, time persistence should help to detect outliers more easily and should lead to a more stable algorithm if errors re-appear in *new data* that are fed into the algorithm.

Besides the aforementioned aspects, one needs to take into account that financial data often has a panel structure, with relationships within groups of time series. In our application, this group structure is relevant because reporting errors could be highly correlated within a certain (reporting or economic) group. In general, there are two ways to deal with the panel structure: First, it is possible to do a stratified split according to these groups so that the distributions across the train and test data sets are the same. The disadvantage, however, is that this could result in data leakage from the train to the test set. Second, it is possible to split the data set according to the groups themselves so that one group with all its observations is either always in the train or test data set. This, however, is only advisable if there are many small groups. However, since the data sets collected by central banks are relatively large, random splitting is usually sufficient to ensure that the distributions in the train and test data sets in terms of group belongings closely align. Still, it is advisable to do stratified sampling across relative membership categories such as banks, funds or sectors to ensure proper sample distribution by construction.

Another aspect that needs to be considered in the train-test split is that – by definition – the outlier label is heavily imbalanced. To ensure that the train and test data set contain the same fraction of outliers, we stratify the train-test-split according to the outlier label.[3]

Finally, the choice on the size of the test set should depend on the size of the data set and the fraction of outliers in the data set. For evaluation purposes it is necessary that an appropriate number of outliers is available in the test data set to avoid noise in the evaluation metrics due to the scarceness of the outlier label. However, this is sometimes not easy to achieve if the data set is too small. For our data sets, we have picked different test sizes.

To summarize, splitting and stratifying the data is important to properly evaluate the success of an outlier detection model and to avoid over-fitting and noisy model selection. Therefore, we always split our data, estimating the outlier detection model using train data and evaluating the model using test data.

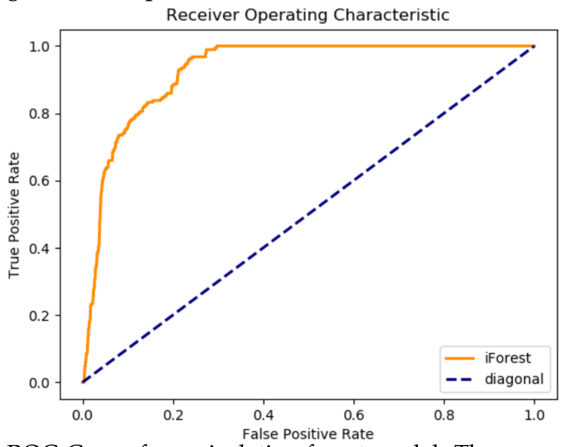
2.3 How to evaluate the success of unsupervised models?

For the evaluation of our models, we largely use two measures: the receiver operating characteristic (ROC) curve and the precision-recall (PR) curve. These two measures principally measure the trade-off between different competing ideas of model performance. Each model we estimate produces a score, whether an instance is an outlier or not. An exception is One Class SVM that produces binary labels. However, here we can derive a score from the distance of the observation of the hyperplane that separates the classes. The threshold at which an instance is considered anomalous is often driven by the nature of the data and the problem to be solved. In our case, we are concerned with measuring a model's ability to detect rare and infrequent anomalies in our data and thus, we are faced in most cases with a severe imbalance between labelled outliers and inliers. For imbalanced data, the PR curve is often a fitting performance metric because it focuses more strongly on the minority class (Davis and Goadrich, 2006) . Thus, we mainly rely on the PR-curve to evaluate our models. Below is a brief description of the PR- and ROC curve.

[3]If no label is available in the data set, stratifying according to the label is, of course, not possible.

ROC curve The ROC curve plots the true positive rate (i.e. the recall) against the false positive rate. This curve shows for different threshold values of what constitutes an outlier and the number of true positives change versus the number of false positives. In Figure 2, we show a sample ROC-curve for an Isolation Forest model. For a model that perfectly separates outliers from inliers, the orange line would make a right angle to the upper left-hand corner, indicating that there is no trade-off because there are no false positives and all true positives have been isolated. The blue line indicates the curve a random model should produce. The area between the blue and orange lines (the area under the curve or AUC) measures the degree of the trade-off between true positives and false positives. For this model, the ROC-AUC score is 0.93, which is relatively high.

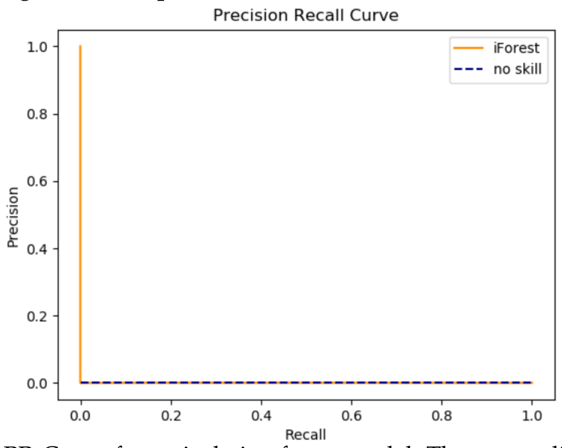
Figure 2: Sample ROC-curve for an isolation forest



Notes: The figure shows a sample ROC-Curve for an isolation forest model. The orange line is the ROC-curve for an isolation forest model, which shows the trade-off between the true positive and false positive rates for different thresholds of outlierness. A perfect model would form a right angle at the top left of this figure, whereas a skill-less model is represented by the diagonal blue line. Data source: WpInvest Aug-2017

PR curve However, we may not be interested in a model's ability to correctly predict inliers, but rather how well it predicts the much smaller outlier class. In this case, the PR-curve is much more useful, as it reflects the fraction of true positives among all positive predictions. Figure 3, which shows the PR-curve for the same model as above, makes clear that in the face of a class imbalance, PR-curves are a more appropriate measurement because potentially we may overestimate the ability of our model to predict the minority class on the basis of ROC AUC.

Figure 3: Sample PR-curve for an isolation forest



Notes: The figure shows a sample PR-Curve for an isolation forest model. The orange line is the PR-curve for an isolation forest model, which shows the trade-off between the precision and recall for different thresholds of outlierness. A perfect model would form a right angle at the top right of this figure, whereas a skill-less model is represented by the horizontal blue line. Data source: WpInvest Aug-2017

2.4 Computing environment

All experiments were conducted in *Python 3.6*. We used *Jupyter notebooks* during the evaluation phase as well as for the data exploration. The *scikit-learn* (Pedregosa et al., 2011) and *pyod* (Zhao et al., 2019) machine learning libraries were selected for training and evaluation of the models. The applicability of latter one was indeed good as it was specifically designed for anomaly detection tasks. The efficient training of the Autoencoder Neural Network was achieved by shifting expensive computations to GPUs (Nvidia V100). For such computations we used the *pytorch* (Paszke et al., 2019) deep learning library. In addition, the *statsmodels* (Seabold and Perktold, 2010) package was used for the estimation of the statistical models as well as for conducting statistical tests.

3 Data sets

The goal of our investigation is to apply the major classes of unsupervised algorithms to micro data sets which differ significantly in their dimensionality, frequency of collection, and their inherent properties. We will benchmark the performance of these unsupervised algorithm classes against information on all previously detected errors and outliers and evaluate their potential usefulness in a central bank's data quality management process. For our study, we use four different micro data sets that are typically collected by a central bank and for which we have the initially reported data set with all errors, next to a respective final data set where the errors were corrected and the outliers were flagged. A short summary of the data sets can be found in Table 1 and the following paragraphs.

The Investment Funds Statistics (IFS) collects all information about the individual holdings on a security-by-security basis for all investment funds issued by investment companies and public limited investment companies residing in Germany and subjected to the German Capital Investment Code. In addition to the granular holdings of each fund, a wide range of general information on the fund level is collected as well as the fund's key assets and liabilities[4]. Each line in the data set for the purpose of this paper corresponds to an asset or liability value submitted by the reporting entity at the end of each month.

The Securities Holdings Statistics (WpInvest) contains security-by-security information on all holdings of financial institutions registered in Germany for their domestic and foreign customers as well as the institution's own holdings. For each security – identified by the International Securities Identification Number (ISIN) – the nominal amount (or in some cases the number of units held) as well as the market value of the holding, the currency of the holding and the country of the holder are reported. In addition, flags are reported for securities repurchase and securities lending transactions. For the purpose of this paper, each line corresponds to a report by a single financial intuition for all of its customers located in Germany, broken down by the customers's sectoral classification (e.g. household, government, non-financial corporation etc.) and the customer's country of origin.

The MFI interest rate statistics covers all interest rates and the corresponding outstanding amounts (volumes) of existing and new business euro-denominated deposits and loans, broken down into the sectors households and non-financial corporations from the Euro area. The reporting is submitted by roughly 240 German banks on a monthly basis with month-end reporting values. Each line in the data set corresponds to a reported value by a bank for the interest rate or the corresponding outstanding amount, broken down by the aforementioned economic sectors for different (original) maturity buckets and loans and deposits respectively.[5]

Finally, the German part of the Money Market Statistical Reporting (MMSR) lists all transactions conducted in the money market of around 115 reporting agents from Germany. For the purpose of this paper, we focus only on the unsecured part of the MMSR which includes all unsecured transactions on a daily basis, covering

[4]Amongst others, this covers balance-sheet-like information on the total assets, the amount borrowed and loaned by the fund, the use of derivatives, and cash holdings in bank accounts. General information on the fund level cover information such as the number of fund shares outstanding, type of replication, assets under management and distributions.

[5]There a further breakdowns that are reported in the data set, for example the breakdown by the purpose of loans to households. For the purpose of this paper, however, we do not discuss those details of the data set.

borrowing and lending transactions for various instruments and for both fixed and variable rate contracts. Each row in our data set represents a single transaction of a bank with an eligible counterparty, including information on the counterparty itself, the agreed interest rate, the amount borrowed or lent as well as the maturity of the transaction and so forth.

Table 1: Overview of data sets

Name	Description
Investment Funds Statistics (IFS) 10k rows, 150 features, 5% outliers, Blaschke and Haupenthal (2020)	Monthly micro data on assets under management by German investment management and externally managed investment companies. Among other things, data consist of every security held by the respective investment fund on a security-by-security basis.
Security Holdings Statistics (WpInvest) 5M rows, 110 features, 0.001% outliers, Blaschke et al. (2020)	Securities reported by financial institutions domiciled in Germany which they hold for domestic or foreign customers. Furthermore, domestic banks provide information about their own holdings, irrespective of where the securities are held.
MFI Interest Rate Statistics (ZISTA) 40K rows, 12 features, 0.7% outliers, Bade and Krueger (2019)	The MFI interest rate statistics is composed as a representative sample of around 240 institutions. The MFI interest rate statistics measure the interest rates applied by domestic banks (MFIs) and the corresponding volumes for euro-denominated lending and deposit business with households and non-financial corporations domiciled in the euro area.
Money Market Statistics (MMSR) 25K rows, 33 features, 0.04% outliers, Bade et al. (2019)	The MMSR statistics provides the information on transactions carried out by monetary financial institutions on the euro money market. MMSR covers transactions in the secured, unsecured, foreign exchange swap and EONIA swap (euro overnight index swaps or OIS) market segments.

4 Data pre-processing and recommendations

4.1 General considerations

4.1.1 Encoded categorical values

Mixed data consists of numerical and categorical attributes. In order to work with categorical data, usually the non-ordinal categorical data is encoded with methods such as one-hot encoding or hashing (see Section 4.2 for a discussion of these approaches).[6] This change in representation, which in the case of hashing is not reversible, can lead to the generation of additional columns and different scaling. Because one-hot-encoding of categorical variables creates additional features, it can inflate the number of (encoded) categorical features relative to the number of (not encoded) numerical features. In financial data this can, for example, be the case for a feature that holds a large number of different currency codes. Creating a large number of features from categorical variables and keeping the number of numerical attributes unchanged can artificially increase the influence of categorical variables. If we use feature bagging in an isolation forest, for example, inflating the number of one-hot encoded categorical features increases the probability of drawing the categorical feature relative to drawing a numerical feature.[7] This problem is exacerbated by the fact that for each data point, there is at most one column in the expanded feature space for the categorical attribute that is set to one. Hence, the new feature space is quite sparse and data points are more likely to be equally far apart from one another (curse of dimensionality).

How to cope with this issue depends on the learning method. Trees in isolation forests need to grow deeper in order to capture the expansion of the feature space by a one-hot encoded column, more trees have to be generated, or more weight can be assigned to numerical columns by repeating them in the data set.

[6]For ordinal categorical data, order-preserving numeric encodings can be used.

[7]We use feature bagging in ensembles to reduce the correlation between estimators by training them on random samples of features instead of the entire feature set.

Alternatively, only a subset of the categories are encoded to keep the feature space small. For distance-based methods, measures such as Gower distance that take the mixed data structure into account or a reweighting of the individual columns are suitable.

4.1.2 Skewed and sparse distributions

This section will discuss how to deal with skewed and sparse data to detect outliers in the Bundesbank data. In particular, this section only covers numerical data, for a discussion of how to handle categorical data we refer the reader to Section 4.2.

Skewed data refer to data with long tails on either side of the distribution, which holds also true for multivariate distributions. One naïve approach to handle long tails of distributions is to truncate/clip them appropriately at empirically specified thresholds. In light of the fact that we are aiming to detect outliers, this needs to be handled with much caution in order to avoid truncating actual outliers. For the IFS data, we did some experiments truncating large values to a pre-specified threshold. This had two effects: (a) the focus slightly moved away from over-weighting large investment entities; however, (b) at the same time actual outliers were "truncated away" and could not be detected anymore. In the end, to avoid truncating away actual outliers and because most columns had no long tails, we did use truncation to deal with skewed data. The same applies to the WpInvest and ZISTA data sets.

Sparse data refer to the fact that in finite samples, high dimensional feature spaces are sparsely populated with data points. Because our data only provide us with a relatively small sample size, sparsity creates some difficulties in detecting outliers when a large number of features are included in the estimation. In section 4.1.4 we will discuss concrete approaches to reduce the dimensionality of the feature space to avoid estimating models in too sparsely populated feature spaces.

4.1.3 Missing values

The data sets used in this exercise are, relatively speaking, quite complete, as many of the values are required fields and do not pass basic validation checks if not filled. In cases where information in the training data is indeed missing, be it due to missing reference data or because the key is appearing for the first time (thus leading to missing values in the lagged values columns), the missing data is filled with zero or, in the case of categorical variables, assigned to a placeholder string. Additionally, for the ZISTA data, where continuous data is indeed missing, a separate categorical variable is created, which indicates the amount type, either positive, negative, or missing. The additional missing category had a negligible impact on the model's performance.

4.1.4 High dimensional data

The following table provides an overview of different approaches to reduce the dimensionality of the data. Our data sets have many categorical variables, and as such, are high dimensional following the aforementioned preprocessing steps. Aside from the bucketing/binning approach mentioned below, we explored using PCA and Autoencoders ways to reduce the dimensions prior to estimation, as well as bagging to reduce variance of the base estimators. Table 2 describes these methods and our evaluation thereof.

Table 2: High dimensional data reduction methods

Approach	Description	Evaluation
PCA	Linearly maps features into lower dimensional space.	PCA is efficient and easy to understand. However, it does not handle categorical variables or nonlinear relationships between features well.
Autoencoder (AE)	Uses a neural network architecture to compress or encode features into a lower dimensional space.	Is able to flexibly model nonlinearities between feature categories (unlike PCA). In particular, using it as a means of vectorising a large number of categorical variables is promising. AE suffers from a high degree of tuning parameters and long training times.
Feature Bagging	A meta estimator that combines (via mean or max) a number of base detectors on various sub-samples of the data set to improve the predictive accuracy and to control over-fitting. Features are randomly sampled from a subset of the features.	This is a useful technique we used during the intermediate and model validation stages of model development.
Feature Selection	Choosing or omitting features based on domain knowledge or empirical developments.	We did very little manual feature selection, except to omit non-reported features merged from reference data sets.

4.1.5 Large data sets

Large data sets may pose a problem for learning methods with high time and space complexity. Instance-based methods such as k-nearest neighbour (Ramaswamy et al. (2000); Angiulli and Pizzuti (2002)) and local outlier factor (Breunig et al. (2000)) have a time complexity of $O(nd)$ (or $O(log(n)d)$ when using an indexing structure) in the testing phase, where n is the training set size and d is the number of features. Thus, large training sets incur additional indexing structure construction cost in the training phase and query cost that depend on the training set size in the testing phase. In addition, instance-based methods need to keep the whole training set in memory during testing. Instead of working with the complete training data, a stratified subsample can used. However, the subsample may have an insufficient number of outliers. Therefore, for the WpInvest data, for example, we train on a subsample with a fixed normal-to-outlier class ratio of 10:1. In order to avoid inflated performance metrics, we have to ensure that we perform the evaluation on the complete (testing) set and not on the training data with over-sampled outliers.

4.1.6 Excluding extreme outliers

Exploratory analysis of the data has revealed a number of extreme outliers in the data. In particular, by simply plotting the development of a particular position over time in the ZISTA data set, one may notice the spike(s) in such series. It should be noted that such spikes are rare fluctuations of the business that do not represent errors but correctly reported data. As a result, training the model on such data may negatively influence the detection rate of the outliers on unseen data.

Considering the above-mentioned findings, we have added a pre-processing step where a subset of extreme outliers was removed from the data before the execution of the training cycle. The set of candidate samples for removal was selected according to the following criteria: the data point has to reside outside of three standard deviations from the mean of the distribution.[8]

Such pre-processing steps resulted in a higher detection rate and an overall positive outcome in terms of the defined performance metrics. We believe this step mainly affected the decision boundary of the model and

[8]There is a number of other techniques (like an Inter Quantile Range or Mean Absolute Deviation instead of the Standard Deviation) for removing the extreme outliers in the pre-processing step.

subsequently increased the generalization capabilities of the model. We found that this approach was a useful, computationally cheap method to improve the quality of the trained model.

4.2 Categorical variables

The variables in many financial data sets are of mixed types. Besides continuous variables, the data often include categorical variables. If the data set has a panel structure, for example, the cross-sectional and time dimension trivially correspond to categorical variables. We discuss four approaches to deal with categorical variables: One-hot Encoding, one-hot encoding a subset of categories, hierarchies, and hashing.

One-hot encoding One common approach to dealing with categorical variables is One-Hot Encoding. For each category, we create a binary variable that takes the value one if the categorical variable is equal to the category and zero otherwise. One advantage of this approach is its simplicity. A downside is that for categorical variables with many categories, the resulting number of binary variables is large. This can lead to a very sparsely populated feature space, especially if only small numbers of observations are part of each category. We can furthermore run into performance issues for algorithms whose computational cost increases with the number of features in the model. Another downside of one-hot encoding is that we lose the information that for a given categorical variable, the realizations of the binary (one-hot encoded) variables are not independent from each other. Because each observation belongs to one category, only one binary variable out of the group of one-hot encoded categories can take the value one. Although a model can learn this type of structure, we make the task of the model harder by not encoding information on the dependence between binary variables that stem from the same categorical variable.

One-hot encoding a subset of categories One way to counteract the large number of sparse features resulting from one-hot encoding is to one-hot encode only a subset of categories. Encoding only a subset can improve the performance metrics by counteracting overfitting to categories with a small number of observations. By resulting in a smaller number of encoded variables, we can also reduce the training time by selecting categories. A disadvantage is that the approaches require the selection of additional hyperparameters (e.g., a variance threshold of number of variables k). The approaches furthermore remove the distinction between non-encoded categories, which may pose problems for explainability. For the main part of our project, we used two ways to select a subset for encoding. The first approach is the *selection of top-k groups*. Instead of introducing a binary variable for each value of a categorical feature, we only create a binary variable for the top-k values, where the top-k values refer to the categories with the largest number of observations. Residual categories that are not in the top-k values are mapped to a separate binary variable. A special case of this approach is encoding the mode of the categorical variable. The second approach is using a *variance threshold*. Here, we calculate the variance of each one-hot vector and only include those with a variance above a certain threshold. Because the ranking of categories is identical if we use the variance or count the number of observations in a category, the variance threshold and the selection of top-k groups yield the same results as if we select an adequate threshold (value of k).

Hierarchies For many of the categorical variables in financial data, a grouping is possible. Because the number of groups is smaller than the number of categories, one-hot encoding groups results in a smaller number of binary variables than simple one-hot encoding. The approach preserves the information on the group level. For country codes, for example, we can group categories by larger geographic regions (Europe, Asia, ...) and one-hot encode these groups. This corresponds to mapping the categories to higher levels in hierarchical categorizations.

Hashing Another approach to encode categorical variables that results in a lower number of features than one-hot encoding is hashing. A hash function maps a categorical attribute with domain size k to a domain with size $k' \ll k$, thereby keeping the feature space small. However, relationships between categories are not preserved. Because the size of the target space of the categorical attributes can be set via the hyperparameter k', the feature space does not become uncontrollably large. Since hashing maps categorical values uniformly to the target space, collisions prohibit a one-to-one mapping from the feature space back to the original space.

Findings We find that one-hot encoding a subset of variables and using hierarchies improved the performance in terms of the success of the models in isolating outliers and running times of the training as compared to simple one-hot encoding. Out of the outlined approaches, hashing had the worst performance regarding the models' ability to isolate outliers.[9]

4.3 Numeric variables

4.3.1 Scaling

Independent scaling Here we scale features independently, i.e. without taking information from other features into account. For continuous variables, we apply min-max scaling to rescale variables between zero and one (one-hot-encoded categorical-type variables are naturally already scaled between zero and one). We also applied standard scaling, which has the effect of centring all inputs with a mean of zero and a variance of one. Scaling is essential for distance-based methods (such as LOF) to ensure equal weighting of features, and generally for efficient optimization of the cost function. For the particular case of the ZISTA data, we log-scaled features to normalized skewed distributions. In addition to global scaling, we also explored several stratified scaling approaches, described in the following section.

Scaling that captures dependencies The observations in our data sets are not independent. For example, own securities holdings that banks report in WpInvest data belong to reporting the banks' portfolio. If we feed the raw data to an algorithm, we do not exploit the domain knowledge on potential interdependencies between observations, i.e. positions in the same portfolio. To incorporate this information, we can scale numerical variables with aggregates by groups. For example, we can divide all own holdings of a bank in WpInvest by the aggregate size of own holdings of the bank. An alternative to scaling is to incorporate additional features that capture interdependencies. In the WpInvest example, we can include indicator variables for banks that allow the algorithm to model relationships between all own holdings of a bank, such as a larger average size of the holdings of the bank compared to the other banks.

For the IFS data we can normalize with the own-fund volume of investment funds. This gives an indication of the relative importance of funds positions and avoids a too large weight on large positions in absolute value. When we re-scale the numerical features in the IFS data, we do not find improvements in the overall performance of the outlier detection algorithms. However, the flagged outliers focused less on large funds and more on funds with relatively large positions in specific asset classes.

4.3.2 Binning

Binning is a method to discretize or smooth numerical data. Usually the continuous data is discretized in a fixed number of bins of equal width or by using quantiles to generate bins with an approximately equal number of observations. Then, for discretization, the result can be encoded in one-hot or in an ordinal format. For smoothing, values can be replaced, e.g., by their bin means.

For the WpInvest data, we observed that the performance of the quantile strategy was superior to equal-width binning. However, omitting the binning step altogether led to the best performance. One-hot encoding

[9]For the IFS data, some categorical classifications can be directly inferred from some of the numerical attributes. Hence, the information gain associated with these features might be small. Indeed, the results without categorical features are almost as good as with categorical features. In addition, because most outlier detection models are only properly specified for numerical data only this might be the cleanest approach for IFS data without losing much in terms of detecting outliers.

was inferior to ordinal encoding because of the increased size of the feature space and the loss of ordinal information.

4.4 Feature engineering

As for other types of data, feature engineering can have a large impact on the performance of algorithms that learn the structure of the data. We discuss three types to features that we can engineer in many financial data sets.

Past realizations In time series analysis, we commonly include lags to account for the influence of past realizations of a variable on future realizations. In outlier detection with unsupervised machine learning methods, past realizations provide context to the algorithm that can help to distinguish common from unusual data points. Large values of numerical variables, for example, can seem anomalous if we do not account for past realizations of the same variable in the previous period. A common method to account for past realizations is to calculate first differences. For a data generating process with

$$y_t = \mu + y_{t-1} + \epsilon_t, \tag{1}$$

first differencing leaves us with

$$y_t - y_{t-1} = \epsilon_t - \epsilon_{t-1}. \tag{2}$$

We eliminate the fixed component μ from the data, which our model then does not have to explain to model the structure of the data. We also make the implicit assumption that the previous realization's marginal effect on future realizations is one. If we want to leave the choice of the marginal effect size to the model or allow for different marginal effects for different groups of observations, we can include the previous realization as a feature, instead of calculating the first difference.

Aggregates Including aggregates also allows for a contextual evaluation by the model. For example, we can include the overall issued nominal value of a security as a yardstick for the model to compare to the size of the holdings. Because most algorithms can flexibly learn interactions between variables but cannot learn to aggregate values across rows, aggregation should be part of the feature engineering if we believe that it adds useful information to the data.

Context from other statistics Another source of contextual information can be other statistics. If we model banks' interest rates in the ZISTA data, for example, we can include interest rates, set by the Governing Council of the ECB.

Findings For all data sets, we find that first differencing and adding lagged features only slightly improved the performance of the algorithms. Adding aggregated interest rates per maturity and reporting period, led to a slight improvement for ZISTA data. In the IFS data and for the WpInvest data, the inclusion of aggregates also resulted in small improvements of the performance. Adding reference interest rates to the ZISTA data did not improve the performance of the model. One reason for the absence of a gain in performance is that the inclusion of one-hot encoded periods (time fixed effects), already allows the model to take into account contextual changes at time t. Therefore, additional information on changes in the interest rates, set by the ECB, do not add explanatory value. However, because they can allow for better explainability of the results by having a clear interpretation, the interest rates are superior to one-hot encoded time periods.

5 Approaches to detect outliers and recommendations

In this section, we strive to provide a broad overview of algorithms that allow for the unsupervised detection of anomalies.[10] Because of the abundance of resources on the methodology of the algorithms, we do not provide a detailed description of their inner workings in this paper. Instead, Table 3 refers the reader to the original paper that introduced the algorithm and further resources.

Table 3: Resources on the methodology of the anomaly detection algorithms

Algorithm	Original Paper	Further Reading
Isolation Forest	Liu et al. (2008)	
kNN	Ramaswamy et al. (2000); Angiulli and Pizzuti (2002)	
DBSCAN	Ester et al. (1996)	
LOF	Breunig et al. (2000)	
FINCH	Sarfraz et al. (2019)	
One Class SVM	Schölkopf et al. (1999)	
Autoencoder	Rumelhart et al. (1986)	Schreyer et al. (2017)
PCA & rPCA		
HBOS	Goldstein and Dengel (2012)	
ARIMA		Junttila (2001)

Table 4 shows a short intuition behind the algorithm and conceptual differences between the anomaly detection algorithms that we evaluated. Here, we distinguish between five groups of algorithms on the basis of their methodological approach. The first algorithm uses decision trees to isolate outliers. Algorithms in the second group use notions of distance or estimates density functions. Cluster based approaches use clustering algorithms, whereas SVM based algorithms rely on Support Vector Machines for classifying observations as outliers. Reconstruction based methods map the data to a lower dimensional space and then reconstruct the higher dimensional representation of the data. They then flag observations as outliers that have a high reconstruction error. Finally, we also discuss more classical statistical approaches.

[10]For a taxonomy and discussion of different anomaly detection algorithms, see, e.g., Goldstein and Uchida (2016) and Zhang et al. (2007).

Table 4: Overview of anomaly detection algorithms

		Intuition	Strengths	Weaknesses	Global vs. Local	Assessment
Tree Based	Isolation Forest	Anomalous instances in a data set are easier to separate from the rest of the sample (isolate), compared to normal data points. In order to isolate a data point, the algorithm recursively generates partitions on the sample by randomly selecting an attribute and then randomly selecting a split value for the attribute. When the iTree is fully grown, each data point is isolated at one of the external nodes. Intuitively, the anomalous points are those (easier to isolate, hence) with the smaller path length in the tree, i.e. points that are earlier separated at nodes of the tree.	• Fast to estimate • Easy to implement • Intuition of approach is easy to understand and certain degree of explainability (allows us to look at individual trees) • Few hyperparameters • Results are relatively robust to hyperparameter tuning	• Not tuned towards detecting local anomalies • Standadrd implementations of isolation forest cannot handle categorical data. One-hot vectors are treated equally to numerical data. This is problematic in the same way as for decision tree classifiers with random splits between categoricals	Global	The algorithm is well suited for our (mixed) data and a very good baseline model for comparison with other algorithms. IForests are also a very good starting point when implementing alternative models, feature spaces etc, because they are easy to implement, have few hyperparameters, fast to estimate, and are relatively robust.
Distance and Density Based	kNN	To determine the outlyingness of a data point, determine the (average) distance to its k(th)-nearest neighbour(s). Outliers are far away from their nearest neighbours, whereas inliers are similar(=close) to their nearest neighbors.	• Conceptually simple • Explainability • Distance metric takes information of the complete row into account	• Hyperparameters (number of neighbors and distance metric) make tuning more difficult • Not tuned towards detecting local anomalies • Standard implementations do not scale well for large or high-dimensional data sets • Suffers from curse of dimensionality	Global	The method is well suited for small to medium-sized data sets of low/medium dimension. For high dimensions both outlier-detection and computational performance suffer. Due to its conceptual simplicity, this algorithm serves as a good baseline model for benchmarking

... continued on next page

Table 4: Overview of Anomaly Detection Algorithms (continued)

		Intuition	Strengths	Weaknesses	Global vs. Local	Assessment
	DBSCAN	Density-Based Spatial Clustering of Applications with Noise determines core samples that are in a neighbourhood with high density. A neighbourhood is dense for a sample if there are at least a certain number of samples within a given distance. Data points that are close to a core sample form a cluster. Data points that are neither core samples nor close to them are considered outliers.	Can handle clusters of arbitrary shapes	• Does not provide anomaly scores out of the box (but binary labels) • Possibly slow training with high memory usage	Tendency towards detecting global outliers, but depending on the choice of hyperparameters DBSCAN can detect local outliers as well	The algorithm performs for outlier detection relatively well. However, the delicate interplay of hyperparameters and the feature space complicates the usage of this method. Incorrectly setting the hyperparameters leads to large training times and high memory usage. In addition, the calculation of anomaly scores has to be implemented separately.
	LOF	Anomalies are not located in densely populated neighbourhoods. The algorithm calculates the LOF score of an instance as the ratio of the average distance of the instance to its k-nearest neighbours over the average distances of the k-nearest neighbours to their respective neighbors. Anomalies will obtain large scores as they have low local density compared to normal observations.	Easy to implement	• Not very robust estimates • Slow to estimate with large number of neighbours	Depending on the number of neighbours (hyperparameter), the LOF can detect local as well as global outliers in the data	Like kNN the method works well with small and low-dimensional data. Large data sets and high-dimensionality pose a challenge to the algorithm that then becomes intractably slow.

... continued on next page

Table 4: Overview of Anomaly Detection Algorithms (continued)

		Intuition	Strengths	Weaknesses	Global vs. Local	Assessment
Cluster Based	FINCH	Forms chains by linking data points to their nearest neighbour. If data points have the same first neighbour, it links them to each other. The connected components of this graph form a cluster. To generate additional clusters,the algorithm performs the previous steps recursively on computed average data points.	• Conceptually simple • Few hyperparameters • Fast training and estimation, so it can be used for large and high dimensional data set	• Not designed as an anomaly detection method • Multiple solutions • No singleton clusters	–	The method was not considered due to related scalability issues in the reference implementation of the authors of this method.
Kernel Based	One Class SVM	One-Class SVM is a special case of the traditional SVM algorithm that is used for unsupervised scenarios. The main property of the traditional SVM is the ability to build a non-linear decision boundary by projecting the data to a high-dimensional (feature) space. The "Kernel trick" is used to perform the projection. In the feature space a "straight" hyperplane is built to separate the data to classes (positive / negative). The goal is to find the function that is positive for regions with high density and negative for low density.	• Ability to learn complex decision boundary. • Provides an "anomaly score" per sample (distance to the hyperplane)	• Compute and storage requirements increases rapidly with the number of training samples, due to the expensive kernel computation. • Might become sensitive to hyperparameters. Selection of the kernel, rejection rate, soft margin etc. have to be adjusted according to the data set structure. • Difficult interpretability of the model for high-dimensional data sets. • Cannot handle categorical data.	Captures global outliers almost always. For detection of local outliers tuning of model hyperparameters might be required.	OCSVM does not scale well to larger data sets (although more computionally efficient implementations are being developed). It can require a certain degree of hyperparameter tuning to achieve an acceptable performance.

... continued on next page

Table 4: Overview of Anomaly Detection Algorithms (continued)

		Intuition	Strengths	Weaknesses	Global vs. Local	Assessment
Reconstruction Based	Autoencoder	Performs non-linear data transformations by reducing the dimensionality to a lower level and then transforming it back to the original data space. The transformation may consist of multiple steps (hidden layers). Anomalies are those samples that performed worst in the reconstruction phase.	• Ability to capture non-linear relations in complex data structure • Multiple assessment of errors: reconstruction error, latent representation	• Computationally expensive • Lots of hyperparameters for tuning • Interpretation of the results is difficult • Sensitive to the attributes selected	Captures global outliers better than local outliers	The algorithm performs well detecting the global outliers.
	PCA and rPCA	Performs linear data transformation by reducing the dimensionality to a lower level and then transforming it back to the original data space. Anomalies are those samples that performed worst at the reconstruction phase.	• Not many hyper-parameters • Relatively fast • Level of explainability is relatively high • Multiple assessment of errors: reconstruction error, latent representation	• Poorly performs capturing nonlinear relationships • Sensitive to the attributes selected	Captures global outliers better than local	The algorithm showed a relatively good performance and could be well suited as a baseline model for comparison.
Statistical	HBOS	Uses the histogram approach for calculating the outlier score. The frequency (relative amount) of samples in a bin is used as density estimation. In multivariate anomaly detection, the scores obtained from each histogram are computed individually and combined afterwards.	• High level of explainability • Extremely fast • Small number of hyper-parameters • Good interpretability	Does not capture (unusual) relationships between the features and is sensitive to the feature selection	Captures global outliers better than local outliers	The algorithm showed good performance only for a particular set of feature combinations. Therefore, for successful model selection, the set of features must be done carefully.

... continued on next page

Table 4: Overview of Anomaly Detection Algorithms (continued)

		Intuition	Strengths	Weaknesses	Global vs. Local	Assessment
	ARIMA	Forecasting the time series data using the historical observations of the series.	• Good explainability • Few hyperparameters	• Each model has to be built separately for individual time series • Scaling of the anomaly scores across multiple series has to be done carefully	Captures global outliers better than local outliers	The algorithm showed a big potential for time series data. However, it needs to be calibrated carefully. We can imagine that some models need to be retrained/-calibrated from time to time because trends for some of individual time series change over time.

6 Combination of detectors and recommendations

In this section, we discuss how to detect outliers by combining the output of multiple outlier detection algorithms. These so called ensemble methods are meta-algorithms that combine several machine learning algorithms into one predictive model and thereby aim to improve/boost the performance of the final model (Opitz and Maclin, 1999; Rokach, 2010; Polikar, 2006). The outlier detection algorithms that are used to construct the ensemble are known as components. Outlier detection ensembles have many advantages over individual outlier detection algorithms. Often there are cases where a model that was trained on a data set will work well for the particular subset of the data and will fail when applied to other parts of the data. Also, in some instances, a trained model can perform well solving a task in one data set and fail performing the same task in other data sets. An ensemble model helps to leverage the different strengths of algorithms by not relying on a single model that could work well on only a particular data set. If one component under-performs in detecting outliers in a specific scenario, it is likely that this doesn't strongly impact the overall ensemble model's performance, as other components can work well for the same data points and thereby compensate. Hence, overall it can be observed that ensemble models often provide more stable / robust results when compared to individual models (Aggarwal, 2012; Aggarwal and Sathe, 2017). Generally, the design of an outlier detection ensemble model follows three steps (Aggarwal and Sathe, 2017):

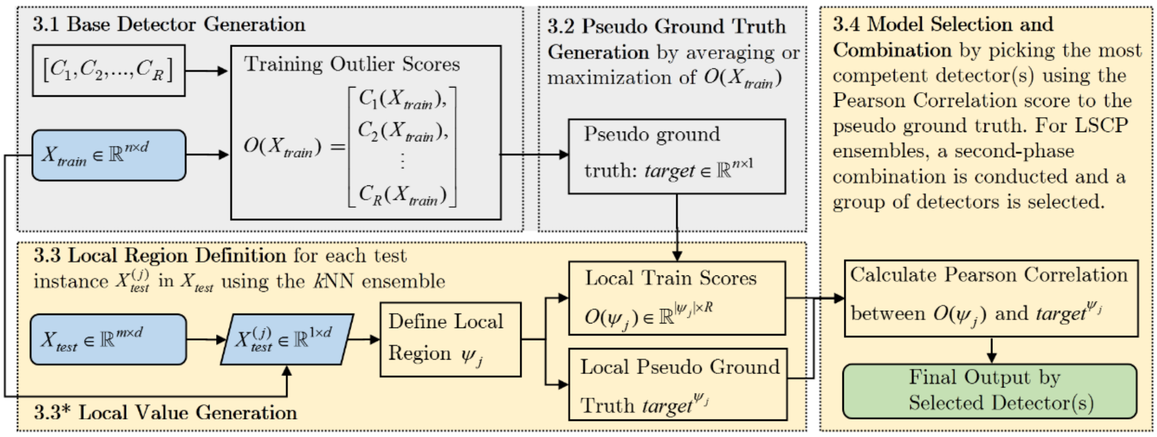
1. *Model creation*: This step includes methodology or algorithms used to create the components.
2. *Normalization*: Ensemble models may consist of multiple heterogeneous components and the output from each component can be in different ranges. Therefore it is important to normalize the different scales of outlier scores from different components.
3. *Model combination*: We refer to the algorithm that combines individual components' outputs as fusion method. We have utilized and implemented different fusion methods (see below).

Outlier detection ensembles can be categorized into multiple groups, depending on either the type/class of components used or based on dependency within the components in the ensemble model (Aggarwal, 2012). We designed an ensemble model that is a hybrid (independent and model-centered) of different outlier ensemble groups. Drawing on Aggarwal (2012); Zhao and Hryniewicki (2018); Pasillas-Díaz and Ratté (2016); Zimek et al. (2014), we implemented different model combination functions. In the following, we distinguish three approaches.

6.1 Simple fusion methods

As the name suggests, simple fusion methods combine different components' outputs by using simple mathematical operations as combination functions. Among others, these functions are: maximum, average, damped averaging, pruned averaging, majority voting, normalized to one per component max, normalized to one per component average. Apart from their simplicity, advantages of simple fusion methods are that they are easy to implement, allow for easy interpretability, and are less computationally intensive. On the other hand, they exhibit limitations. The functions are not capable to learn the patterns in the component output, and the performance improvement depends on the diversity of the components' output. Further limitations are that *max* has a tendency to overestimate the outlierness and *average* tends to dilute the outlierness due to irrelevant components (Zimek et al., 2014; Aggarwal and Sathe, 2015). We recommend starting with these methods in the ensemble due to their simplicity, even though in our tests, they were not able to outperform individual components.

Figure 4: Flowchart of DCSO algorithm



3.1 Base Detector Generation
$[C_1, C_2, ..., C_R]$
Training Outlier Scores
$O(X_{train}) = [C_1(X_{train}), C_2(X_{train}), \vdots, C_R(X_{train})]$
$X_{train} \in \mathbb{R}^{n \times d}$
3.2 Pseudo Ground Truth Generation by averaging or maximization of $O(X_{train})$
Pseudo ground truth: $target \in \mathbb{R}^{n \times 1}$
3.4 Model Selection and Combination by picking the most competent detector(s) using the Pearson Correlation score to the pseudo ground truth. For LSCP ensembles, a second-phase combination is conducted and a group of detectors is selected.
3.3 Local Region Definition for each test instance $X_{test}^{(j)}$ in X_{test} using the kNN ensemble
$X_{test} \in \mathbb{R}^{m \times d}$
$X_{test}^{(j)} \in \mathbb{R}^{1 \times d}$
Define Local Region ψ_j
Local Train Scores $O(\psi_j) \in \mathbb{R}^{|\psi_j| \times R}$
Local Pseudo Ground Truth $target^{\psi_j}$
Calculate Pearson Correlation between $O(\psi_j)$ and $target^{\psi_j}$
Final Output by Selected Detector(s)
3.3* Local Value Generation

Notes: The figure shows a flowchart of DCSO algorithm with an explanation for each step. The figure was adapted from Zhao et al. (2018).

6.2 Unsupervised fusion methods

In an outlier ensemble, multiple outlier detection algorithms are used as the components which are applied on the input data for the outlier prediction. Later, these components output are used as input to the ensemble model for fusion. If labels are available (supervised learning) the optimization in the fusion-step can be based on the predictive performance of the labels. If labels are not available, wich usually is the case in outlier detection, we need to rely on an unsupervised fusion method. We implemented two unsupervised combination methods: Dynamic Combination of Detector Scores (Zhao and Hryniewicki, 2018) and Ensemble of detectors with correlation votes / Ensemble of detectors with variability votes (Pasillas-Díaz and Ratté, 2016).

Dynamic Combination of Detector Scores (DCSO) consists of two main steps: generation and combination. In the generation step, different and diverse base detector algorithms are selected. These base detectors can contain any outlier detection algorithm. In the combination step, a local region is defined for each observation by selecting the top-n most similar neighbors. Then, the base detector which delivered the best performance in the defined local neighborhood is selected as the competent detector for the observation. This competent detector is used to predict the outlier score for the selected test instance. DCSO focuses on local regions in the data for the computation of outlier scores, hence it can detect local outliers. All the steps and complete flowchart of DCSO are shown in Figure 4.

In the case of EDCV (Ensemble of detectors with correlation votes) and EDVV (Ensemble of detectors with variability votes), the outlier scores of all the algorithms for input samples are stored in a matrix F of size $m \times T$ where m is the number of samples and T is the number of algorithms (components). In the first step, vote matrix V of size $m \times T$ is calculated which contains the number of votes assigned by each algorithm for each data sample. A modified boxplot technique is used for the calculation of votes where a sample gets a vote if its score is greater than 150% of the Inter Quartile Range. In the next step, a weight matrix W is computed based on EDCV and EDVV approaches. In the EDCV method, a correlation coefficient matrix C between the output score F is calculated, and then by using the matrix C the corresponding weights of each component are calculated using the equation

$$W_n = \frac{(\sum_{m=1}^{T} C_m n) - 1}{T - 1}. \tag{3}$$

Similarly in EDVV, a matrix D of mean absolute deviations (MAD) between output scores F is calculated, and later, weights of each component are calculated using

$$W_n = \frac{\sum_{m=1}^{T} D_m n}{T-1}. \tag{4}$$

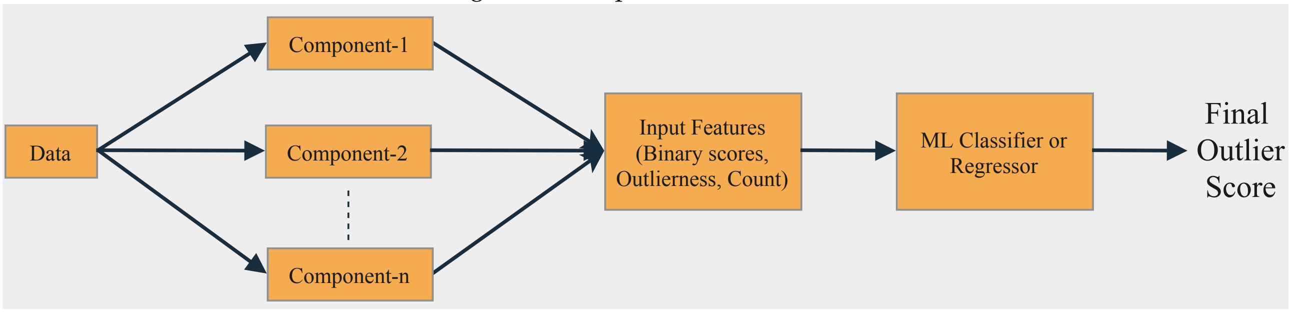
In the last step, the final score of each sample is calculated using the corresponding votes from V and weights from W. So ECVV and EDVV use the correlation and variability between individual components respectively to compute the final ensemble output.

Especially in the early stages of analyses, when labels are often missing, these methods can help to fuse the outputs of multiple components. However, due to large execution times, these methods sometimes have to be run on sub-sampled data sets. In our application, both methods were able to provide slight improvements in the results compared to the best output provided by any single outlier detection algorithm. However, because the performance improvements in our tests were not substantial, we opted to also investigate more complex fusion methods.

6.3 Complex fusion

In this approach, a supervised machine learning model is used as a fusion or combination function. In order to apply this approach, we need information on which observations are actual outliers as targets. We can then apply the fusion method to data, even if we have no information on the targets to isolate outliers. This method is similar to stacking or stacked generalization (Wolpert, 1992; Smyth and Wolpert, 1999; Breiman, 1996). The intuition behind this approach is that the outputs from several components (outlier detection algorithms) for an input sample are fed into another machine learning model to combine them into a single output. Here, the output of each component can be considered as a derived feature. So the derived feature can be a binary output (outlier/inlier), the outlierness score, or both. This fusion model can hence be seen as a meta-classifier/regressor that can use dependencies or identify patterns in prior components output. Figure 5 shows how this approach works. The figure illustrates that the input data is fed into different components of the ensemble, i.e. different outlier detection algorithms. The output prediction of each component takes the form of a binary output and an outlierness score. These outputs (features) are the input data for the ML classifier/regressor in the next stage. Here, the ML classifier/regressor is used for fusion by training the output from the previous step with given labels and predicts the final outlier score for the input sample. The advantage of this approach is that the ensemble methods are capable of learning from component outputs. Any supervised machine learning algorithm can be used as fusion method. Also, these algorithms do not rely on the diversity in previous components output but can identify patterns. Due to their learning capabilities, these methods were able to outperform all previous approaches substantially.

Figure 5: Complex fusion method

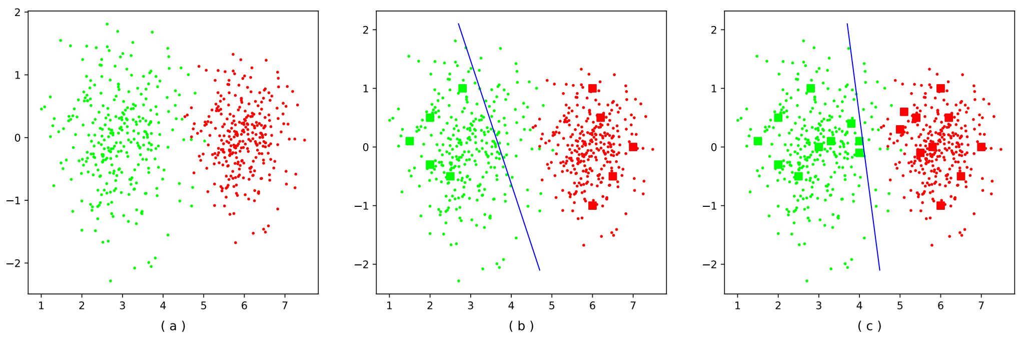

Data
Component-1
Component-2
Component-n
Input Features (Binary scores, Outlierness, Count)
ML Classifier or Regressor
Final Outlier Score

Notes: The figure illustrates the complex fusion method which takes prediction and outlier score output from each component as input. Then it fuses the input and learns the features to predict the final output from the ensemble model.

7 Active learning for outlier detection

Active learning is a unique type of machine learning where a learning model will frequently query the user/-expert for labels of selected samples for better performance (Settles, 2009; Rubens et al., 2011; Das et al., 2020). This method falls into the category of supervised learning in which only a small part of the data is labelled. In this method, human involvement in data labelling is treated as more valuable. This technique is used in cases where a large amount of unlabelled data is available and labelling is expensive (Settles, 2009). Figure 6 explains the advantages of this approach. Furthermore, with this approach, the outlier detection problem is started as unsupervised learning and then can gradually turn it into a supervised learning method.

Figure 6: Active learning



Notes: The figure shows an illustration of active learning. (a) Input unlabelled data consists of two clusters represented by colors green and red. (b) Classification result of active learning model on unlabelled data at the early stages. This approach is an iterative process where each iteration includes selecting few samples from unlabelled data based on the query strategies for the expert query, then labelling the selected samples by experts' feedback and later training the post-processing model (learner) on the labelled samples. These labelled samples are represented as squares. Here the decision boundary is represented as a blue line and is not optimal. (c) Result of active learning model on the unlabelled input data after few iterations. Here the decision boundary is more accurate in separating two clusters in the unlabelled data compared to the previous result due to the iterative learning process.

We designed active learning for outlier detection as an iterative process. In our case each iteration corresponds to a reporting period and is split into two steps (except in first iteration which includes only the first step). During the first step, an unsupervised outlier detection algorithm is applied on a new data set to detect potential outliers. We applied this approach on IFS data. Then, from the predicted output, the top 5% outliers are selected based on the outlierness score of each input sample. In the second step, a pre-trained supervised machine learning model is used which is also called as a post-processing model or a learner. The output from the first step is fed into the post-processing model that selects a number of (e.g.: top 30 or top 100 by output score of the post-processing model) samples to be reviewed by domain experts. The selection of samples for expert feedback depends on the scenario and on the implemented query strategies. We applied active learning in two scenarios and implemented two query strategies which we will discuss below.

- *Stream-Based Selective Sampling* (Lewis and Gale, 1994; Settles, 2009): Each unlabelled sample from a large corpus is drawn one at a time and fed to the learner. Then, the learner will decide whether to request the label of this sample from the expert or to discard it. We implemented this approach with a small modification that if the post-processing model can classify an unlabelled input sample with a score greater than some threshold then the learner itself can assign a label to such a sample, otherwise the sample will be dispatched for the expert query. We set the threshold to 90%. However, we couldn't find much progress with this approach due to the poor performance by the post-processing model in labelling the input samples.

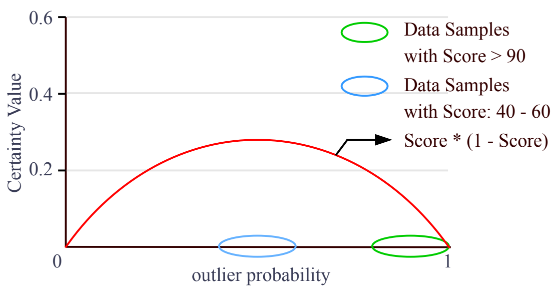
- *Pool-Based Sampling* (Settles, 2009): This is a scenario that is more commonly studied in active learning. In this type, a filter (in our case, an unsupervised model) is applied on a pool of unlabelled data and the samples are ranked based on the returned results. Then from the ranked list the top n samples are selected for the query, where n is a parameter. This parameter is application-specific and reflects the amount of resources that are available for labelling tasks. We tried n with values 30, 100, and 300. The filter is accompanied by a query strategy which we will discuss next.

The two query strategies used for this work are:

- *Certainty sampling* (Settles, 2009): In this query strategy, the samples for which the post-processing model is most certain in its outlier classification are selected for the expert query. Consequently, samples with the highest outlierness score are selected in each iteration (green zone in Figure 7).
- *Uncertainty Sampling* (Settles, 2009; Lewis and Gale, 1994; Pelleg and Moore, 2005): Here, the post-processing model selects such samples for the query for which it is least confident or least certain on how to label. The motivation behind this method is that having expert feedback / labels for the hardest samples will help the learner to improve its performance in the next iteration. Samples selected by this strategy are represented by the blue zone in Figure 7.

The selected samples by the post-processing model are queried for labels. Next, the post-processing model is trained using the new labelled samples and this two-step process is then repeated in each iteration.

Figure 7: Certainty sampling



Notes: The figure shows certainty sampling. Samples in the green zone are the ones for which the post-processing model scores above 90% and the blue zone represents the samples for which the model is least certain about its class.

8 Results and evaluation

The success of the algorithms and their relative performance is, of course, highly domain specific. Depending on the working definition of an outlier in financial data, the performance of the algorithms will differ. We still report detailed results for all algorithms. What is more, we show the performance with and without feature engineering and parameter tuning. The reason is that conditional on a data set and our definition of outliers, the results provide suggestive evidence on the necessity and benefits of feature engineering and parameter tuning as well as an indication of the heterogeneity in performance between different algorithms.

Tables 5 and 6 and Tables 7 and 8 present the evaluation results based on ROC-AUC and PR-AUC for the securities holdings statistics (WpInvest), the investment funds statistics (IFS), the interest rate statistics (ZISTA) and the money market statistics (MMSR) respectively. For each data set, we depcit the baseline results in the first column, the results with feature engineering in the second column, the results with parameter tuning in the third column and - where applicable – the results with feature bagging in the forth column. The table provides three main insights. First, we find that there is profound heterogeneity regarding the success of algorithms in isolating outliers. Second, an algorithm that, in our application, showed a good performance across different data sets is the Isolation Forest. Even without time consuming feature engineering and parameter tuning, the isolation forest provided a good performance relative to other approaches. Without a prior intuition which algorithm successfully isolates outliers in financial data, our findings suggest that Isolation Forests are a good starting point. Third, the importance of feature engineering, parameter tuning and (if applicable) feature bagging for the performance depends on the algorithm and data set. Even for Autoencoders that, due to their complex structure, can discover features, we find that feature engineering can provide sizeable improvements.

Table 5: Results (ROC-AUC) for unsupervised outlier detection algorithms and combination of detectors (WpInvest & IFS)

		Evaluation Results (ROC AUC)							
		WpInvest Total 3.9M samples and Outliers 0.001%				IFS Total 120000 samples and Outliers 4.6%			
		Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging	Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging
Ensemble (Homogeneous Learners)	Isolation Forest	0.965	0.963	0.982	–	0.597	0.641	0.652	0.638
Distance/Density Based	kNN	0.805	0.882	0.882	–	0.599	0.628	0.628	0.623
	DBSCAN	0.880	0.945	0.945	–	0.554	0.609	0.617	–
	LOF	0.626	0.740	0.740	–	0.570	0.569	0.570	0.652
	OCSVM	–	–	–	–	–	–	–	–
Cluster Based	Autoencoder	0.516	0.552	0.595	–	0.607	0.608	0.665	0.652
	PCA and rPCA	0.780	0.632	0.632	0.591	0.607	0.665	0.669	0.651
	HBOS	–	–	–	–	–	–	–	–
	ARIMA								
Combination of Detectors		(Simple Fusion) 0.9832	(Unsupervised Fusion) 0.954	**(Complex Fusion)** **0.9877**	–	(Simple Fusion) 0.6429	(Unsupervised Fusion) 0.6473	**(Complex Fusion)** **0.7648**	–

Notes: The table shows results (ROC-AUC) of all the unsupervised outlier detection algorithms and Combination of Detectors method for WpInvest and IFS data sets.

Table 6: Results (ROC-AUC) for unsupervised outlier detection algorithms and combination of detectors (ZISTA & MMSR)

		Evaluation Results (ROC AUC)							
		ZISTA Total 4.3M samples and Outliers 0.7%				MMSR Total 2.2M samples and Outliers 0.04%			
		Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging	Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging
Ensemble (Homogeneous Learners)	Isolation Forest	0.499	0.637	0.639	–	0.925	0.935	**0.935**	–
Distance/Density Based	kNN	0.702	0.733	0.738	–	0.894	0.905	0.905	–
	DBSCAN	–	–	–	–	–	–	–	–
	LOF	0.576	0.733	0.736	–	0.901	0.908	0.908	–
	OCSVM	0.506	0.625	0.626	–	0.921	0.928	0.928	–
Cluster Based	Autoencoder	0.701	0.704	0.708	–	–	–	–	–
	PCA and rPCA	0.714	0.744	**0.745**	–	0.844	0.865	0.865	–
	HBOS	0.695	0.735	0.735	–	0.915	0.928	0.929	–
	ARIMA	0.665	0.665	0.691					
Combination of Detectors		(Simple Fusion) –	(Unsupervised Fusion) –	(Complex Fusion) –	–	(Simple Fusion) –	(Unsupervised Fusion) –	(Complex Fusion) –	–

Notes: Result (ROC-AUC) of all the unsupervised outlier detection algorithms and Combination of Detectors method for ZISTA and MMSR data sets.

Table 7: Results (PR-AUC) for unsupervised outlier detection algorithms and combination of detectors (WpInvest & IFS)

		Evaluation Results (PR AUC)							
		WpInvest Total 3.9M samples and Outliers 0.001%				IFS Total 120000 samples and Outliers 4.6%			
		Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging	Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging
Ensemble (Homogeneous Learners)	Isolation Forest	0.0008	0.0009	0.0013	–	0.048	0.055	0.056	0.054
Distance/Density Based	kNN	0.0002	0.001	0.001	–	0.064	0.097	0.097	0.082
	DBSCAN	0.0001	0.0007	0.0007	–	0.052	0.092	0.092	–
	LOF	0.00004	0.00006	0.00006	–	0.055	0.057	0.057	0.073
	OCSVM	–	–	–	–	–	–	–	–
Cluster Based	Autoencoder	0.0001	0.0001	0.0002	–	0.063	0.064	0.085	0.082
	PCA and rPCA	0.023	0.0004	0.0004	0.0002	0.063	0.085	0.086	0.082
	HBOS	–	–	–	–	–	–	–	–
	ARIMA	–	–	–					
Combination of Detectors		(Simple Fusion) 0.001	(Unsupervised Fusion) 0.0009	**(Complex Fusion)** **0.003**	–	(Simple Fusion) 0.065	(Unsupervised Fusion) 0.056	**(Complex Fusion)** **0.3441**	

Notes: The table shows results (PR-AUC) of all the unsupervised outlier detection algorithms and Combination of Detectors method for WpInvest and IFS data sets.

Table 8: Results (PR-AUC) for unsupervised outlier detection algorithms and combination of detectors (ZISTA & MMSR)

		Evaluation Results (PR AUC)							
		ZISTA Total 4.3M samples and Outliers 0.7%				MMSR Total 2.2M samples and Outliers 0.04%			
		Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging	Baseline	+Feature Engineering	+Parameter Tuning	+Feature Bagging
Ensemble (Homogeneous Learners)	Isolation Forest	0.013	0.018	0.019	–	0.012	0.015	0.015	–
Distance/Density Based	kNN	0.012	0.023	0.024	–	0.012	0.015	0.015	–
	DBSCAN	–	–	–	–	–	–	–	–
	LOF	0.014	0.026	0.027	–	0.015	0.017	0.017	–
	OCSVM	–	–	–	–	–	–	–	–
Cluster Based	Autoencoder	0.071	0.078	**0.079**	–	–	–	–	–
	PCA and rPCA	0.013	0.070	0.072	–	0.013	0.016	0.016	–
	HBOS	0.013	0.022	0.028	–	0.012	0.018	**0.019**	–
	ARIMA	0.015	0.016	0.019	–	–	–	–	–
Combination of Detectors		(Simple Fusion) –	(Unsupervised Fusion) –	(Complex Fusion) –	–	(Simple Fusion) –	(Unsupervised Fusion) –	(Complex Fusion) –	

Notes: The table shows results (PR-AUC) of all the unsupervised outlier detection algorithms and Combination of Detectors method for ZISTA and MMSR data sets.

As mentioned in Section 6, we also applied different combination functions in outlier detection ensembles. We evaluated a total of 16 different combination functions, mainly categorized into three categories. The performance across the three categories and all the methods on the IFS data set are summarized in Table 9. There were some glimpses of performance improvement by simple and unsupervised methods but the difference was not large. Simple and unsupervised fusion methods couldn't outperform the best individual component result at a significant level which might be due to the lack of diversity in output between the individual components used as input to the combination functions. However, the performance of complex fusion methods were promising as they clearly outperformed all other methods by a large margin. The complex fusion methods were able to learn the patterns and relation between each component output for the prediction of output. So there were sizeable improvements in the ROC-AUC and PR-AUC values by complex fusion methods.

Table 9: Comparison of different fusion methods

	Combination functions	**F1-Score**	**PR-AUC**	**ROC-AUC**
	Best individual component	0.1021	0.0857	0.6643
Simple fusion method	Min	0.1195	0.056	0.5804
	Max	0.0834	0.0828	0.6539
	Avg	0.0834	0.0673	0.6601
	Normalize-one-per-Component-Max	0.0991	0.0696	0.6539
	Normalize-one-per-Component-Avg	0.0834	0.0655	0.6598
	Majority_Voting	0.0834	NA	NA
Unsupervised fusion method	DCSO-AVG	0.1092	0.063	0.588
	DCSO-MAX	0.115	0.0799	0.6273
	DCSO-AOM	0.0913	0.0629	0.6389
	DCSO-MOA	0.1109	0.0889	0.6336
	EDCV	0.0834	0.0650	0.6591
	EDVV	0.0834	0.0676	0.6595
Complex fusion method	SVM-Model	0.1123	0.074	0.6412
	LR-Model	0.1068	0.0923	0.6494
	KNN-Model	0.2912	0.3042	0.7045
	RFC-Model	**0.3515**	**0.3148**	**0.7734**

Notes: The table shows a comparison of results if we apply different fusion methods to a combination of detectors using the IFS data set. Complex fusion method outperformed the other fusion methods as well as best individual component used for outlier ensemble.

Regarding active learning, for the IFS data set, we observed slight improvements using either query strategies (see Table 10). Uncertainty sampling outperformed certainty sampling in all considered performance metrics.

Table 10: Comparison of query strategies

Query strategy	**Avg PR-AUC**	**AVG ROC-AUC**	**Avg Precision**	**Avg Recall**
Certainty sampling	0.3131	0.6077	0.1372	0.4772
Uncertainty sampling	**0.379**	**0.7221**	**0.3278**	**0.5925**

Notes: The table shows a comparison of query strategies using IFS data set.

Taken together, we find that there is no "one size fits all" solution to outlier detection in the financial data sets that we evaluated. Complex fusion can help to overcome the necessity of selecting a single approach by rendering the selection of an algorithm an empirical exercise. Therefore, although fusion did not dramatically improve the performance relative to the best single algorithm, combination of detectors with a fusion method can be helpful in some contexts. To go beyond a strictly data-driven isolation of outliers, active learning provides possibilities to take domain knowledge into account that goes beyond the detection capabilities of the algorithms.

9 Explainable AI

If our sole interest lies in successfully flagging outliers and we are confident that optimization with cross validation (see Section 2.2) leads to internally and externally valid results, we can treat the outlier detection algorithms as a black box.[11] However, in many business applications, we need a better explanation of our decision. If we apply outlier detection in DQM, we often need an explanation for data users why an observation was excluded that goes beyond referring to the decision of an algorithm. The same holds if we make inquiries at reporting agents regarding data points that were flagged as outliers by an algorithm. Likewise, to use outlier detection algorithms to uncover – economically meaningful – unusual structures in the data, we need better insight into what distinguishes an outlier from an inlier.

To provide such explanations, we can either use a transparent model that we can interpret directly to detect outliers, such as a histogram, or we can use a surrogate model that provides an explanation for a more complex algorithm's classification[12]. The idea of a surrogate model is to treat the prediction of a more complex algorithm as an outcome and use an algorithm to model the relationship between the outcome and the input features. We discuss methods that are applied globally in Section 9.1 and techniques that are applied locally in Section 9.2. Then. we further develop ways to explain the results of outlier detection with Autoencoders in Section 9.3.

To motivate the need for local explanations, the following briefly explains the difference between local and global explanations in the context of our use case.

9.1 Global methods

Global methods are applied to the full data set and include linear regressions and decision trees on which we elaborate below.

9.1.1 Linear regression

One method for approximating the decision function of a 'black box' model is to approximate it linearly; that is, to estimate the output of the non-linear model as a linear function of the inputs. To do this, we can estimate a linear regression, with the original model input as the independent variables and the model's output (either a probability score or class based on the probability score) as the dependent variable. We can then make inferences about the model's decision function based on the coefficients of the linear surrogate model. The r-squared, or explained variance, of the surrogate model tells us how well the surrogate model approximates the original model.

While this approach has the advantage that it is highly flexible and easily interpretable via the estimated coefficients, the disadvantages are manifold. First, if the surrogate model does a very good job of explaining the original model – for example with an R-squared of 98 percent – then it would behove us simply to use the linear model in the first place. If it does not explain the surrogate model well, then we cannot rely on the explanations it provides us with. Especially for our use case, where we want to explain rare and anomalous instances, they must necessarily not be able to be well-explained by a linear model.

It is important to reiterate that the estimates from a linear regression surrogate model, and any other surrogate model for that matter, are making inferences about the model, not about the data itself. Thus, unlike the original black box model itself which is an abstraction of the real data-generating process, the surrogate model is in turn an abstraction of the black box model.

[11]Following Patino and Ferreira (2018), internal validity is defined as the extent to which the observed results represent the truth in the population we are studying and, thus, are not due to methodological errors. Once the internal validity of the study is established, the researcher can proceed to make a judgment regarding its external validity by asking whether the study results apply to similar patients in a different setting or not.

[12]For a discussion of common approaches to explainable AI, see e.g., Molnar (2019)

9.1.2 Decision trees

One simple and intuitive method to understand the outlier scores of outlier detection algorithms is to estimate and visualize a decision tree.

An advantage of the decision tree over linear regression is that it is more flexible, i.e., it can capture non-linear relationships in the data and can model interactions between input variables that are not additive separable. To implement this method, the outlierness score and the data features are used to estimate a decision tree regressor where the outlierness score serves as the label.

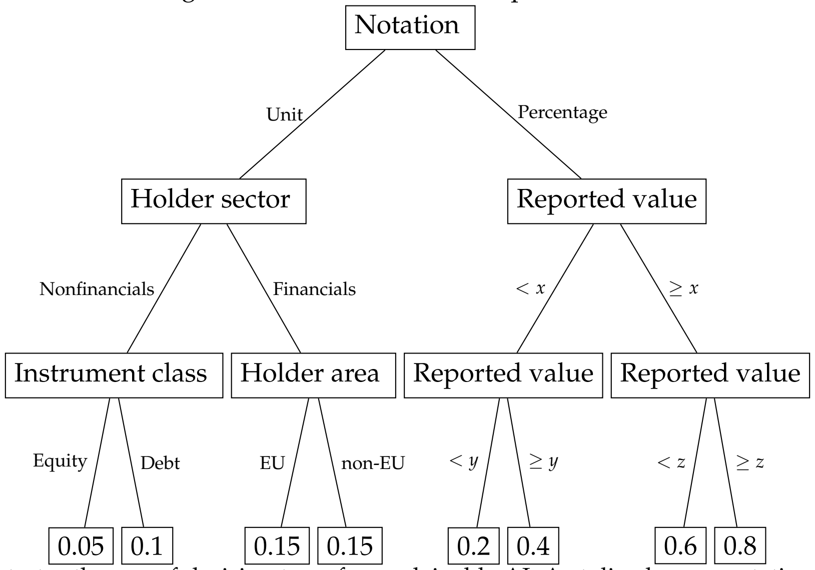
A decision tree highlights the most important features (in terms of entropy) for the outlier detection algorithm. In other words, a decision tree highlights those features that are most relevant for the attribution of a high or low outlierness score of a certain data point. It further delivers interpretable decision rules for these features that can be understood by applying some domain knowledge.

Alternatively, one can estimate a decision tree classifier using the predicted outliers as labels. The tree should be specified relatively simple, with only a small depth, so that it can be interpreted and visualized more easily.

To illustrate this method we apply it to the WpInvest data. Each observation in this data set consists of the aggregate amount that a bank holds for a client in custody for each security, holder sector, and holder area. Let A be a classifier that has been trained to detect reporting errors. Given a set of prediction scores for a reporting period by classifier A, we estimate a decision tree (see Figure 8 for a simplified representation). With the help of the decision tree, one could say that the trained classifier A assigns higher outlierness scores to observations where `Holder sector = Financials` and the reported values are large both in the previous and in the current period.

The illustration in Figure 8 can therefore be interpreted as follows: Since we normalized the outlierness scores to the interval $[0, 1]$ the numbers at the end of the tree can be interpreted as probabilities. Hence, if the notation is percentage notation and the reported value is larger than the respective cutoff values, then in 80% of cases the data point is classified as an outlier by the outlier detection algorithm (right most path in Figure 8). If the notation is unit notation, the holder sector is financials and the holder area is EU, then the data point is only in 15% of cases an outlier according to the outlier detection algorithm. This logic applies equally to all leafs of the tree.

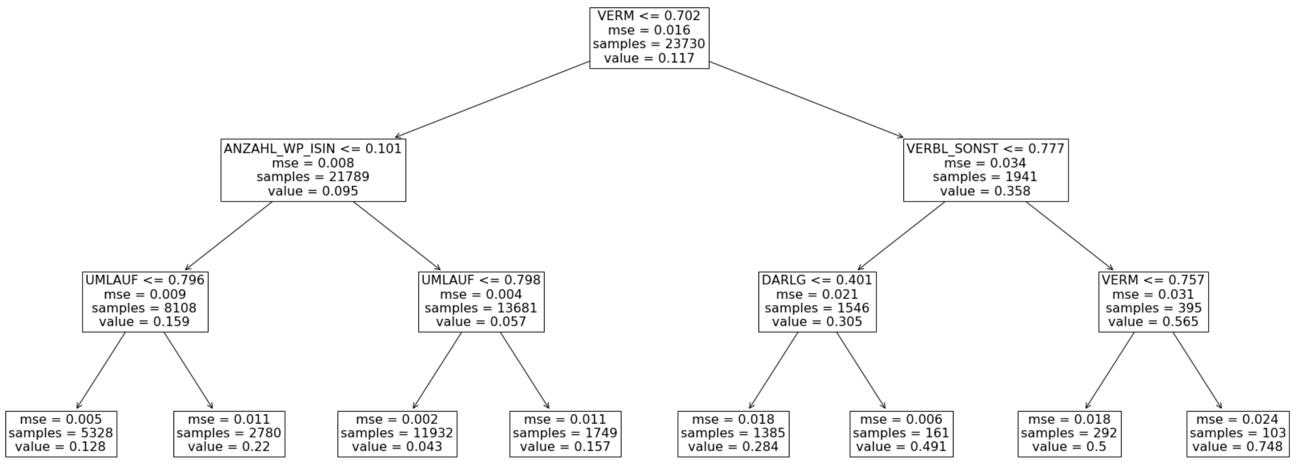
Figure 8: Decision trees for Explainable AI



Notes: The figure illustrates the use of decision trees for explainable AI. A stylized representation of a decision tree that was estimated using outlierness score and the data features. A leaf node gives the outlierness score that the classifier likely assigns to an instance. x, y and z represent cutoff values in the decision tree in case of continuous features.

Next, we apply decision trees to the IFS data to better understand the results of the outlier detection algo-

Figure 9: Example of a decision tree using IFS data



Notes: The figure shows an example of a decision tree using IFS data[13]. Underlying outlier detection model is baseline isolation forest.

rithm. In Figure 9, we plot the decision tree for our baseline Isolation Forest algorithm in the IFS data. The interpretation of the tree is the same as discussed earlier in this section. We find that in the IFS data funds with unusually large non-standard balance sheet positions like other equity and (other) liabilities (VERM and DARLG, VERBL_SONST) are more likely to be classified as outliers by the algorithm. This can be seen on the right side of the figure where funds with other equity positions greater than 0.757 and other liabilities greater than 0.777 get assigned a very high probability of being classified as outliers by the algorithm. This is an interesting economic relation that the algorithm detects because funds with large liabilities or non-standard equity positions are relatively uncommon. The result also suggests that outlier detection in the IFS is capable of detecting economically anomalous data points and that this particular model is less focused on data quality issues concerning funds. As one can see from this example, the decision tree is a very helpful tool to understand the estimated model and on which aspects of the data the algorithm is focused. With different input features or alternative models, the result could be very different leading to alternative insights about the data.

9.2 Local methods

If we have estimated a well-performing model which indicates that a particular instance is anomalous, the model has likely detected that this instance is meaningfully different from similar instances. For example, if a security has a market value in euros of €10 billion and is nominally denominated in Egyptian pounds, whilst all other securities of €10 billion are denominated in euros, the model might predict that this instance is an outlier; given the neighbourhood (market value in euros), the reported nominal currency is anomalous. This simplified example illustrates the need to estimate a surrogate model with input instances that are similar to the instance that is to be explained.

This need to have an explanatory method which is locally valid is one component of a more general framework for instance-specific explanatory methods. Although there is no single agreed-upon definition of what constitutes a good explanation, Table 11 summarises commonly used standards for valid local explanatory models. [14]

[13] VERM is other equity, ANZAHL_WP_ISIN denotes the number of ISIN-securities in the portfolio, VERBL_SONST is the amount of other liabilities, UMLAUF gives the units outstanding, and DARLG are loans to property companies.

[14] For a broader discussion, see Alvarez-Melis and Jaakkola (2018), Antwarg et al. (2019), and Lundberg and Lee (2017)

Table 11: Summary of common characteristics of a valid explanation

Local accuracy / faithfulness	An explanation should be accurate within the local proximity of the instance in question
Missingness	If a feature value is missing, it should receive a weight of zero in the explanation model
Explicitness / intelligibility	If a model changes, and the contribution of a feature in the new model increase relative to the old model, the surrogate model's attribution to that feature should not decrease. In other words, the explanation is consistent with human intuition.

There are only two (related) methods that satisfy at least two of these axioms. These methods two are described below.

9.2.1 Local interpretable model-agnostic explanations (LIME)

Following up on the idea that a linear regression provides a good linear approximation, even for nonlinear relationships in the data, LIME aims to estimate a regression in the local area around an instance to approximate the contribution of input features to the output value for that instance. The idea is very similar to that of a kernel regression.

For a given instance, LIME takes random samples of instances from the input space and perturbs them. These perturbed instances are plugged into the model and a linear regression is estimated on the resulting output. Importantly, the weight that each perturbed instance received in the regression is based on exponential smoothing kernel.

The choice of kernel is a major drawback for LIME, as with tabular data with possibly many binary variables, different distance kernels can lead to very different explanations (see Molnar (2019)). This drawback is addressed with the second method.

9.2.2 Shapley additive explanations (SHAP)

Directly picking up on drawbacks inherent in LIME's weighting function, SHAP uses a concept from game theory — Shapley values — to attribute to each feature their "fair" weight in local surrogate regression model.

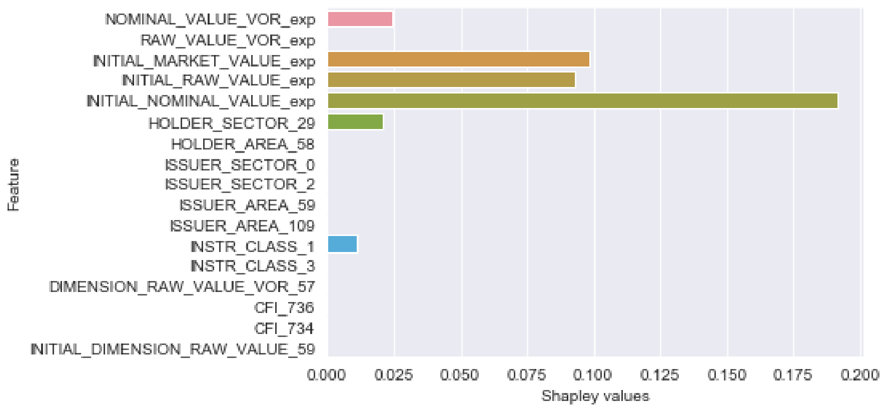
Originally used to fairly attribute payoffs to players in a multiplayer game, Shapley values are a permutation-based method repurposed to provide explanations that satisfy the conditions stated above. Briefly summarised, Shapley values are the average marginal contribution of each player to the output in a multiplayer game as they are present or absent in all possible coalitions with other players, therefore rendering its computation expensive. Yet a sufficient estimation of Shapley values can be obtained by focusing on small and large coalitions. SHAP uses this insight to create the SHAP Kernel, which uses the weights that each coalition would receive in the Shapley value calculation to weight the perturbed instances for the local surrogate regression model. Thus, SHAP provides us with an explanation method with strong theoretical grounding and all the desirable properties outlined in Table 11.

9.2.3 Applications of local explanations

Having local explanations of individual instances is useful in many parts of the machine learning pipeline. Below we identify two use cases in which we found local explanations to be helpful.

Use case 1: Individual explanations for business experts and end users This is one of the most common uses for individual explanations, and one we encountered often during the model validation phase. For example, after we had a working model that was showing good results on the partially labelled data we had available, we obtained predicted outliers on unlabelled validation data and wanted to have domain experts evaluate our predictions. Given the large number of features in the data set, there are many ways in which a particular data point may be anomalous. We wanted to be able to provide guidance on what – in particular – our model found anomalous with a particular instance. To this end, we applied SHAP to obtain instance-level feature attribution for a sample of highly anomalous instances. Figure 10 shows the Shapley values for one particularly anomalous instance.

Figure 10: SHAP explanations for a single instance



Notes: The figure shows SHAP explanations for a single instance. On the x-axis are the SHAP values, and on the y-axis are the input features into the original model. The figure shows how the features contribute (across all alternative "coalitions" of feature values) to the outlier score of the instance. Thereby, SHAP allows us to interpret a single feature's importance not only in the relation to the realization of other features' values that characterize the instance but also for alternative values of the remainder of features.

The Shapley values clearly point to three features as contributing most to the outlieredness of the instance: INITIAL_MARKET_VALUE, INITIAL_RAW_VALUE, and INITIAL_NOMINAL_VALUE. After consulting with business experts, this instance was indeed anomalous due to an incorrect reporting of the INITIAL_RAW_VALUE feature. This exercise also pointed out a limitation of the model: in so far as the input features are correlated[15], as is the case with all three of these features, the SHAP and LIME will not be able to distinguish between them.

Providing this additional information to domain experts is an improvement over a simple outlier-inlier indicator, and additionally help researchers understand the behaviour and edges of their model.

Use case 2: Feature influence During the development of the model, it is often illuminating to understand how the model's output is influenced by certain input features. This in turns can help developers with further tuning and feature engineering. Feature dependence plots on feature values and their corresponding Shapley values from the model help to do this.

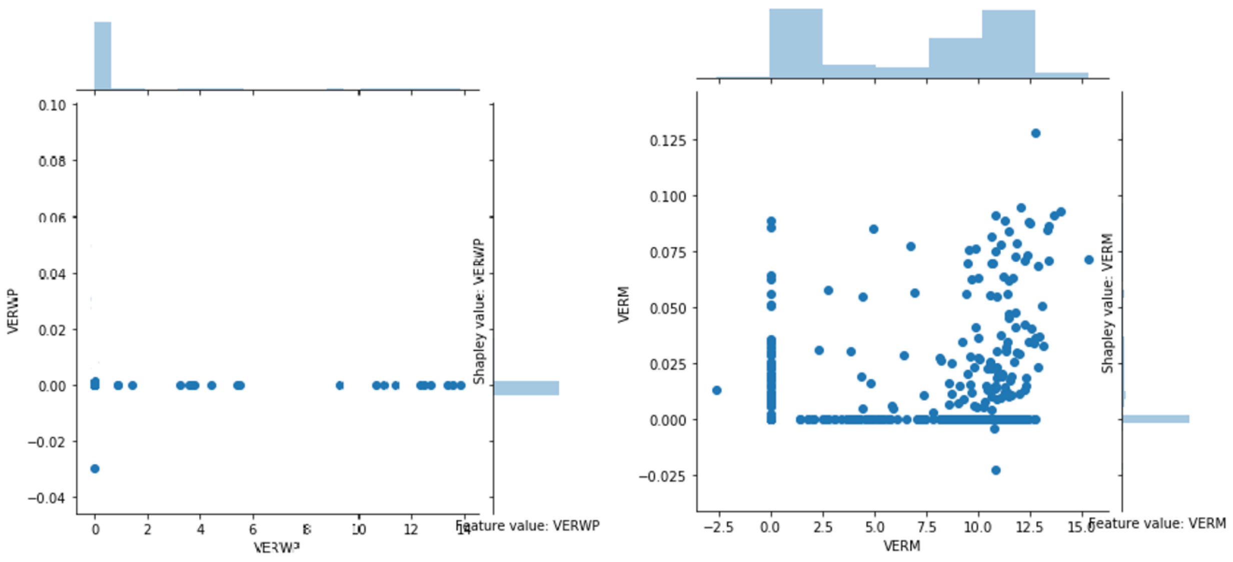
Figure 11 shows the SHAP explanations for two features in a model, and along with their respective feature values. The left panel shows that the feature, VERWP, is unimportant for this model's predictions. The right panel on the other hand indicates that this feature, VERM, is moderately important for the model, where values of zero are also meaningful for some instances. Without these explanation methods, researchers are often left to deduce which features are important by observing the models post-hoc performance with and with a particular set of features. In the unsupervised setting, this is not possible and thus such explanations are very useful for model comparison and diagnostics.

9.2.4 Evaluation

Although these methods are very useful additions to the unsupervised toolbox, they are both costly in terms of computation time, and in the case of LIME as discussed above, considerable practical downsides when working with tabular data.

[15]MARKET_VALUE is the market value of the holdings; NOMINAL_VALUE is nominal or book value of the holdings; RAW_VALUE is the originally reported book value of holdings (i.e. original currency, etc). The INITIAL means that this was the value of the first reporting of the bank, which may have been subsequently changed. See Blaschke et al. (2020) for more details on the data.

Figure 11: Individual model explanations and corresponding feature values



Notes: The figure shows individual model explanations and corresponding feature values. Both figures show the SHAP values on the y-axis and their corresponding feature values on the x-axis. On the left, the SHAP values are not meaningfully different than zero, whereas for the subfigure on the right, larger values of the feature are associated with larger SHAP values. Data source: IFS Jan-2019

In terms of runtime, Table 12 summarises an experiment computing local explanations for the top one hundred instances of a data set by predicted outlier score using LIME and SHAP.

Table 12: Test runtimes for 100 individual local explanations

Method	Runtime
LIME	3m31s +/- 13s
KernelSHAP	11m22 +/- 44s

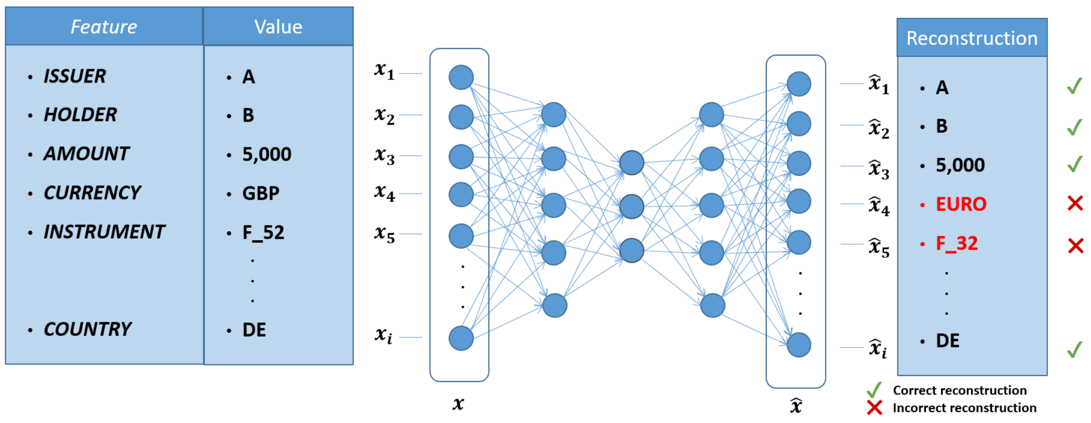
That KernelSHAP is computationally expensive is certainly an important consideration when deciding at what point in the model development process such explanations warrant the time to compute them. For early stages of model development, global surrogate models may suffice to give researchers rough intuition as to feature importance. For later stages, such as interfacing with business experts or regulatory stakeholders, SHAP and to a lesser extent LIME are a worthwhile tool to reach for.

9.3 Autoencoder neural network

In the last decades, neural-network-type models have shown a remarkable progress in various domains. However, understanding the decision making process of such complex models remains a challenging task for domain experts. In the following, we outline our approach to providing explanations for outliers that were detected with an autoencoder.

The goal of the Autoencoder Neural Network is to perform a lossy compression of the data into a lower dimensional space (encoder) and to reconstruct the data in its original dimensionality as accurately as possible (decoder). Because reconstruction is imperfect, the deviation of the reconstructed data from the original data – the reconstruction error – reflects the success of the model in reconstructing a sample. Because the reconstruction error is tightly linked to how well a sample fits into the structure of the data that is preserved in the lower-dimensional space, it provides us with a measure of the outlyingness of a sample. Consequently, we can use the reconstruction error to separate *inliers* that follow a common pattern (low reconstruction error) from *outliers* that deviate from the common structure of the data (high reconstruction error). One challenge of this approach is that the reconstruction error – a single scalar – is not sufficient to answer the question why a sample is flagged as an outlier. To provide an explanation why an observation was flagged as an outlier we study the reconstruction error and its properties in more detail. Instead of collecting the reconstruction errors on the instance level we collect them on the attribute level. In particular, we unfold the reconstruction error of an instance and study corresponding reconstruction errors per individual attribute. This way we are able to identify whether a particular field was reconstructed or not. The subset of fields that were not reconstructed trigger high reconstruction error of an instance. As a result this subset of entries are most likely contain structurally unusual pattern. Figure 12 depicts a schematic overview of the reconstruction on the attribute level. In this example two attributes (*CURRENCY* and *INSTRUMENT*) were reconstructed incorrectly. Moreover, given the values of the other attributes the model predicts that the *CURRENCY* should be 'EURO' instead of 'GBP' and *INSTRUMENT* should be 'F_32' instead of 'F_52'. Therefore, there is a high chance that these two fields contain an error and as a result have to be screened. Technically this is achieved by applying the softmax function on the final/output layer of the decoder network per categorical attribute. Since we use the one-hot encoded representation of a categorical attribute the result of the softmax provides the normalized scores for all categories of an attribute which can be used as a probability estimates. Finally the element with the highest probability score is selected as the prediction category. If such category differs from the corresponding category of the input instance then the field is flagged as incorrectly reconstructed. Such methodology also provides an opportunity to the domain expert to order potential reporting errors correspondingly and start the auditing process from such field(s) that most-likely contain the most *severe* error(s).

Figure 12: Correct and incorrect reconstructions



Notes: The figure provides a schematic overview of the correct and incorrect reconstructions on attribute level of the Autoencoder Neural Network. Each field is flagged correspondingly based on the reconstruction errors collected per attribute field.

Such technique allows us to flag a set of attribute fields that affect the reconstruction quality of an instance at most. In other words, the combination of attributes that were reconstructed incorrectly appears to become an anomaly pattern. Most likely this set of attribute values is what makes the sample anomalous and as a result some of these fields might contain an error. We believe that such features provide more detailed explanation of a particular decision(s) made by the Autoencoder Neural Network and could serve as an important supplement to the domain experts' toolbox.

10 Conclusion

Steadily rising data volumes and an increasing complexity of statistical reporting of micro data led to a surge in the interest of statistics departments to employ statistical learning methods from the fields of data science and machine learning to provide data user with the highest possible data quality. Reducing the burden on the reporting agents in the data quality management process and achieving an overall higher operational efficiency of statistics departments with minimal (costly) human input are equally important goals when moving to data science and machine learning methods.

In this paper, we outlined the steps that we took to implement a prototype that is capable (i) to detect outliers on an unsupervised basis (ii) to provide explanations of data points that seems suspicious, and (iii) to incorporate the feedback of domain experts in the process. We apply our pipeline to data sets that are collected by the Bundesbank and cover the structure and format of a wide range of financial data, including interest rates, money market statistics, sectoral securities holdings, and investment fund holdings. In addition, since we had information on previous reporting errors of all the aforementioned statistics, we were able to evaluate the performance of the various unsupervised algorithms in detecting unusual data points.

We conclude that unsupervised learning algorithms, applied to granular financial data that was collected by a central bank, are not only suitable to detect incorrect reporting and thereby improve the data quality, but, in conjunction with explainable AI, can also provide explanations for what distinguishes outliers from inliers. However, our work also stresses that a production pipeline that is largely automated and that provides the possibility to actively incorporate (human) feedback is at least as important as a proper selection of algorithms.

References

Aggarwal, C. C. (2012). Outlier ensembles: position paper. *SIGKDD Explorations*, 14(2):49–58.

Aggarwal, C. C. (2015). Outlier analysis. In *Data mining*, pages 237–263. Springer.

Aggarwal, C. C. and Sathe, S. (2015). Theoretical foundations and algorithms for outlier ensembles. *SIGKDD Explor. Newsl.*, 17(1):24–47.

Aggarwal, C. C. and Sathe, S. (2017). *Outlier Ensembles: An Introduction*. Springer Publishing Company, Incorporated, 1st edition.

Ahmed, M., Naser Mahmood, A., and Hu, J. (2016). A survey of network anomaly detection techniques. *J. Netw. Comput. Appl.*, 60(C):19–31.

Alvarez-Melis, D. and Jaakkola, T. S. (2018). Towards robust interpretability with self-explaining neural networks. *CoRR*, abs/1806.07538.

Angiulli, F. and Pizzuti, C. (2002). Fast outlier detection in high dimensional spaces. In *European conference on principles of data mining and knowledge discovery*, pages 15–27. Springer.

Antwarg, L., Shapira, B., and Rokach, L. (2019). Explaining anomalies detected by autoencoders using SHAP. *CoRR*, abs/1903.02407.

Aytekin, C., Ni, X., Cricri, F., and Aksu, E. (2018). Clustering and unsupervised anomaly detection with l_2 normalized deep auto-encoder representations. In *Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6.

Bade, M., Doll, H. C., Hirsch, C., Hubrich, A., and Schulz, F. (2019). Money Market Statistical Reporting, Data Report 2019-08 – Metadata Version MMSR-Data-Doc-v1-0. *Deutsche Bundesbank, Research Data and Service Centre*.

Bade, M. and Krueger, M. (2019). MFI interest rate statistics, Data Report 2019-05 – Metadata Version 3. *Deutsche Bundesbank, Research Data and Service Centre*.

Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys Tutorials*, 16(1):303–336.

Blaschke, J. and Haupenthal, H. (2020). Investment Funds Statistics Base, Data Report 2020-05 – Metadata Version 3-1. *Deutsche Bundesbank, Research Data and Service Centre*.

Blaschke, J., Sachs, K., and Yalcin, E. (2020). Securities Holdings Statistics Base plus, Data Report 2020-14 – Metadata Version 3-1. *Deutsche Bundesbank, Research Data and Service Centre*.

Breiman, L. (1996). Stacked regressions. *Mach. Learn.*, 24(1):49–64.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104.

Chalapathy, R. and Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey. *CoRR*, abs/1901.03407.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15.

Dang, X. H., Micenková, B., Assent, I., and Ng, R. T. (2013). Local outlier detection with interpretation. In Blockeel, H., Kersting, K., Nijssen, S., and Železný, F., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 304–320, Berlin, Heidelberg. Springer Berlin Heidelberg.

Das, S., Wong, W.-K., Dietterich, T., Fern, A., and Emmott, A. (2020). Discovering anomalies by incorporating feedback from an expert. *ACM Trans. Knowl. Discov. Data*, 14(4).

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240.

Ernst, M. and Haesbroeck, G. (2017). Comparison of local outlier detection techniques in spatial multivariate data. *Data Min. Knowl. Discov.*, 31(2):371–399.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231.

Goldstein, M. and Dengel, A. (2012). Histogram-based Outlier Score (HBOS): A Fast Unsupervised Anomaly Detection Algorithm. *Poster and Demo Track of the 35th German Conference on Artificial Intelligence (KI-2012)*, pages 59–63.

Goldstein, M. and Uchida, S. (2016). A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173.

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126.

Junttila, J. (2001). Structural breaks, arima model and finnish inflation forecasts. *International Journal of Forecasting*, 17(2):203–230.

Khoa, N. L. D. and Chawla, S. (2010). Robust outlier detection using commute time and eigenspace embedding. In Zaki, M. J., Yu, J. X., Ravindran, B., and Pudi, V., editors, *Advances in Knowledge Discovery and Data Mining*, pages 422–434, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kriegel, H. ., Kroger, P., Renz, M., and Wurst, S. (2005). A generic framework for efficient subspace clustering of high-dimensional data. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–.

Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. *CoRR*, abs/cmp-lg/9407020.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

Lundberg, S. and Lee, S. (2017). A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874.

Molnar, C. (2019). *Interpretable Machine Learning*. lulu.com. `https://christophm.github.io/interpretable-ml-book/`.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198.

Pasillas-Díaz, J. and Ratté, S. (2016). An unsupervised approach for combining scores of outlier detection techniques, based on similarity measures. *Electronic Notes in Theoretical Computer Science*, 329:61–77.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Patino, C. M. and Ferreira, J. C. (2018). Internal and external validity: can you apply research study results to your patients? *Jornal brasileiro de pneumologia*, 44:183–183.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pelleg, D. and Moore, A. W. (2005). Active learning for anomaly and rare-category detection. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 1073–1080. MIT Press.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45.

Prieditis, A. and Russell, S. J., editors (1995). *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*. Morgan Kaufmann.

Ramaswamy, S., Rastogi, R., and Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438.

Rokach, L. (2010). Ensemble-based classifiers. *Artif. Intell. Rev.*, 33(1–2):1–39.

Rubens, N., Kaplan, D., and Sugiyama, M. (2011). *Active Learning in Recommender Systems*, pages 735–767. Springer US, Boston, MA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.

Sarfraz, M. S., Sharma, V., and Stiefelhagen, R. (2019). Efficient parameter-free clustering using first neighbor relations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8934–8943.

Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. (1999). Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 582–588, Cambridge, MA, USA. MIT Press.

Schreyer, M., Sattarov, T., Borth, D., Dengel, A., and Reimer, B. (2017). Detection of anomalies in large scale accounting data using deep autoencoder networks. *CoRR*, abs/1709.05254.

Seabold, S. and Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Smyth, P. and Wolpert, D. (1999). Linearly combining density estimators via stacking. *Machine Learning - ML*, 36:59–83.

Tissot, B., Widjanarti, A., Zulen, A. A., Ari, H. D., and Wibisono, O. (2018). The use of big data analytics and artificial intelligence in central banking. *IFC Bulletin*, 50.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241 – 259.

Zhang, Y., Meratnia, N., and Havinga, P. (2007). A taxonomy framework for unsupervised outlier detection techniques for multi-type data sets. *Rap. tech., Centre for Telematics and Information Technology University of Twente*.

Zhang, Y., Meratnia, N., and Havinga, P. (2010). Outlier detection techniques for wireless sensor networks: A survey. *Communications Surveys and Tutorials, IEEE*, 12:159 – 170.

Zhao, Y. and Hryniewicki, M. K. (2018). DCSO: Dynamic Combination of Detector Scores for Outlier Ensembles. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection De-constructed (ODD v5.0)*.

Zhao, Y., Hryniewicki, M. K., Nasrullah, Z., and Li, Z. (2018). LSCP: locally selective combination in parallel outlier ensembles. *CoRR*, abs/1812.01528.

Zhao, Y., Nasrullah, Z., and Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7.

Zimek, A., Campello, R. J., and Sander, J. (2014). Ensembles for unsupervised outlier detection: Challenges and research questions a position paper. *SIGKDD Explor. Newsl.*, 15(1):11–22.