Bachelor thesis
Computing Science

Radboud University

Comparing shallow autoencoders to normalization, standardization and PCA as preprocessing step for outlier detection

Author:
Gijs Thuis
s4490444

First assessor:
Prof. Data Science, Tom Heskes
t.heskes@science.ru.nl

Second assessor:
Dr. Data Science, Twan van Laarhoven
Twan.vanLaarhoven@ru.nl

Daily supervisor:
PhD student/candidate Roel Bouman
roel.bouman@ru.nl

June 28, 2022

Abstract

We compare normalization, standardization, PCA and shallow autoencoders as preprocessing step for outlier detection. All used outlier detection techniques are hyperparameter tuned and their performance is evaluated using AUC-ROC scores and 4-Fold cross validation. For both lower and higher dimensional datasets, PCA generally show better results compared to shallow autoencoders. Additionally, both autoencoders and PCA do not outperform robust standardization or normalization as preprocessing method in the OD pipeline, on both lower and higher dimensional data. Therefore we do not suggest to use shallow autoencoders as preprocessing method for outlier detection.

Contents

Chapter 1

Introduction

Outlier detection refers to the problem of finding patterns in data that do not conform to expected behavior[7]. Another often citated definition for outliers is Hawkins'[15] where an outlier is identified as "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism". It is a critical topic in machine learning.[8]

The interest in accurate outlier detection stretches across various application domains, of which examples follow shortly. Often outliers are of particular interest, where identifying them is an end goal in itself and sometimes outlier detection is used as a preprocessing step, where outliers may be irrelevant noise and thus should be removed during data cleaning. Some examples of the former include fraud detection[3], health supervision[24] and hydropower generation plant monitoring[2]. This shows that outlier detection has societal and commercial value.

A complicating factor for accurate outlier detection is the *curse of dimensionality*, which existence was first introduced by Bellman [5]. It refers to the problem occuring in cases where data is too high dimensional, for otherwise sufficient machine learning techniques, to accurately model data. For outlier detection the curse of dimensionality appears in cases where the data is too high dimensional for an outlier detection method to catch onto the underlying defining outlier patterns or *nature* of the data to which one can ascribe the outliers to. It is this problem of dimensionality that we partly try to address in this thesis.

The way of trying to overcome the curse of dimensionality is by reducing the number of dimensions. PCA (Principal Component Analysis) is regularly and successfully used to reduce the number of dimensions of input spaces. There are a lot of examples, but some are Kaya et al.[16] who use PCA prior

to clustering for brain tumor segmantation, Al-Bahri et. al.[4] who apply PCA as preprocessing step for image recognition and feature extraction and Sheinker[23] who leverages PCA for detection of visually obscured ferromagnetic objects.
Still it does not always provide a suitable means when used prior to outlier detection methods. As indicated by Harrou[14], PCA based approaches often tend to miss small or moderate anomalies.

In this thesis we investigate whether or not autoencoders can be a replacement for PCA as dimensionality reduction technique in the outlier detection pipeline (and also compare them to normalization and standardization, again as dimensionality reduction technique). The reason for believing in this alternate approach is that autoencoders are capable of catching nonlinear patterns, whereas PCA is not. The first to introduce autoencoders were Rumelhart et al.[21] It becomes more common to believe in autoencoders as a feasible dimensionality reduction technique as can be seen by the growing interest in them, as stated by San Martin et al. [22] We cover the theoretical differences between PCA and autoencoders in the preliminary sections(2.1.3 and 2.1.4).

To expand on the ground for believing that autoencoders could be a feasible preprocessing technique for outlier detection, especially when applied on high-dimensional data, we briefly introduce their core mechanisms. The main task for autoencoders is to reconstruct a given input to output, while constraining the dimensionality using a bottleneck layer. An autoencoders tries to do so by first encoding the input space into a lower dimensional - latent - space and then decoding this bottleneck representation into input space again. This decoding phase exists to prove that, after the encoding transformation, crucial substructures existing within the input space are kept intact in the denser bottleneck representation and are also more prominent as the space is smaller. In this smaller bottleneck representation, outliers may stand out more, as they would not adhere to the key structures leveraged by the autoencoder to reduce dimensions. The used outlier detection techniques should now be more accurate in identifying outliers.

It is because of the previous that we come to the following hypothesis. We suspect that for all used outlier detection techniques, autoencoders as preprocessing method outperform normalization, standardization and PCA as preprocessing method. The outlier detection methods that are used include LOF, IFOR and kNN for outlier detection(The full forms of the abbreviations for OD techniques are given in table 3.2).

Notably, this research is to some extent exploratory. Therefore the hypothesis, while being formulated strongly, should be a mere guide for examining whether or not autoencoders show promising signs for being a preprocessing

technique for outlier detection. Preprocessing for outlier detection is not a new research domain and so the extent to which this research can be labeled exploratory is limited. Nevertheless, autoencoders are not often used as preprocessing technique. Hence the label *exploratory*. Autoencoders are more often used as an outlier detection technique directly. This is shown by the considerable amount of research where autoencoders, or in some cases, variational autoencoders (VAE) are leveraged for outlier detection. Some examples are Gonzalez et al. [11], who succesfully use both basic and variational autoencoders to model anomalous behavior, such as wandering through the house, by electrical consumption by elderly in elderly homes and show better results for VAE's and Mujkic et al.[19], who leverage several autoencoder architectures for detecting anomalous functioning agricultural vehicles. Ultimately, the conclusion will be indicative of whether or not we believe further investigation of autoencoders as preprocessing technique for outlier detection is necessary.

After defining the research question and hypothesis, we now further demarcate the frame of this research by briefly introducing the environment in which the performance of combinations of preprocessing techniques and outlier detection techniques is evaluated. The used outlier detection techniques use unsupervised learning. LOF, IFOR and kNN do not use outlier labels when modeling the data. But we have outlier labels and still use them to evaluate the performance of each preprocessing-detection combination. We use hyperparameter tuning to find the set of hyperparameters for which the the combination models the outliers most accurately. As the evaluation of the model performance uses the outlier labels and we pick the best performing combination according to this evaluation, the model training is supervised. Conclusively, we use unsupervised outlier detection and supervised training.

Chapter 2

Preliminaries

This chapter contains some introductory knowledge about machine learning principles and techniques which are used in the research. First we introduce the preprocessing methods, specifically addressing PCA and autoencoders. Then we describe the basics of the methods of outlier detection being used. And after that we explain how the used performance metric, which is used to assign scores to the performance of pipelines (i.e. combinations of preprocessing methods and outlier detection methods, where each combination contains one from both) on specific data sets, works.

2.1 Preprocessing methods

2.1.1 Normalization

Let us normalize a feature vector $\boldsymbol{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$, where n is the number of samples in $\boldsymbol{x}$. Then for all $i \in \{1, ...n\}$ the normalized value v_i is given by:

$$v_i = \frac{x_i - min(\boldsymbol{x})}{max(\boldsymbol{x}) - min(\boldsymbol{x})}$$

When normalization is applied on a full matrix (representative of a dataset), then the formula is applied on above on all feature vectors in the matrix.

2.1.2 Standardization

Like normalization, standardization is a means of feature scaling. Let us standardize a feature vector $\boldsymbol{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$ where n is the number of samples in $\boldsymbol{x}$. Let σ denote the standard deviation of $\boldsymbol{x}$ and μ denote the mean value of $\boldsymbol{x}$. Then for all $i \in \{1, ...n\}$, the standardized value v_i is given by:

$$v_i = \frac{x_i - \mu}{\sigma}$$

2.1.3 Principal Component Analysis

Principal component analysis[20] is a technique that computes the principal components (PC's) from a given matrix. PC's are linear combinations that capture variability. When applied, PCA constructs principal components in descending order of the extent to which they capture variability (another way to put "capture variability" is: explain information) from a given input matrix. For each PC this is done by means of a projection on a linear combination of the original variables. Each principal component is orthogonal to all previous PC's. There are as much PC's as features, but as PCA reduces the number of dimensions only a smaller fraction is kept.

Given an input space $\boldsymbol{X}$:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \dots & x_{1P} \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{NP} \end{bmatrix}$$

PCA transforms $\boldsymbol{X}$ to another matrix $\boldsymbol{Z}$ where the shape is $N \times P$, equal to the one of $\boldsymbol{X}$.
Each columnar vector is a principal component of input space $\boldsymbol{X}$.

1. For each column vector we compute the mean and put this result in one mean vector: $\overline{\boldsymbol{x}} = \frac{1}{n} \sum_{i=1}^{n} x_i$
2. Define a $N \times P$ mean matrix $\overline{\boldsymbol{X}} = \overline{x}\mathbf{1}_n$
3. Compute the mean centered matrix: $\boldsymbol{Y} = \boldsymbol{X} - \overline{\boldsymbol{X}}$ *Note:* We mean centered X because we want to ensure the vector columns are zero mean gaussians for the next steps.
4. Decompose $\boldsymbol{Y}$ into singular values $\boldsymbol{Y} = \boldsymbol{U}\Sigma\boldsymbol{V^T}$ where the columns of $\boldsymbol{U}$ are left singular vectors and the columns of $\boldsymbol{V}$ are right singular vectors of $\boldsymbol{Y}$. They also form the orthonormal bases of On the diagonal of $\boldsymbol{\Sigma}$ we find the singular values for $\boldsymbol{Y}$. For the shapes of the decomposed matrix we have:
   - $\boldsymbol{U} : n \times d$
   - $\boldsymbol{\Sigma} : d \times d$
   - $\boldsymbol{V^T} : d \times p$
5. Compute principal components $\boldsymbol{Z} = \boldsymbol{U\Sigma}$, where:

$$\boldsymbol{Z} = \begin{bmatrix} z_{1P} & \cdots & z_{1P} \\ \vdots & \ddots & \vdots \\ z_{N1} & \cdots & z_{NP} \end{bmatrix}$$

The first vector (z_1) in this matrix represents the first principal component and is the component capturing the most variance. For the i-th principal component we have:

$$z_i = \begin{bmatrix} z_{1i} & \cdots & z_{1P} \end{bmatrix}$$

Notably, in step 3, 4 and 5 we made mere linear transformations, and PCA can thus be characterized as a projection method. This is important for for our research, as the transformations made in PCA are all linear.

Historically, PCA was previously been executed by means of eigendecomposition (by computing eigenvalues and eigenvectors), but nowadays the more general singular value decomposition (SVD) approach is common.

2.1.4 Autoencoders

Autoencoders are a specific type of neural networks, which are trained to reconstruct given input, after representing it compactly in its *bottleneck* layer, which is the middle layer. Typically autoencoders have k layers, where $k \geq 3$ and k is odd. The first layer has n neurons. The next $\frac{n-1}{2}$ layers have less neurons than their predecessor and after that the final $\frac{n-1}{2}$ layers each have more neurons than their predecessor. The decrease in neurons per layer towards mirror the increase in neurons per layer away from the bottleneck layer.

The first $\frac{n-1}{2}$ layers of the full autoencoder is called the encoder, the final $\frac{n-1}{2}$ layers together is called the decoder and as we saw the layer inbetween the bottleneck layer. Furthermore, the first is also called *input* layer, the last layer *output* layer and every layer inbetween *hidden* layers.

A generic neural network architecture representation for 5 layers is presented in Figure 2.1.

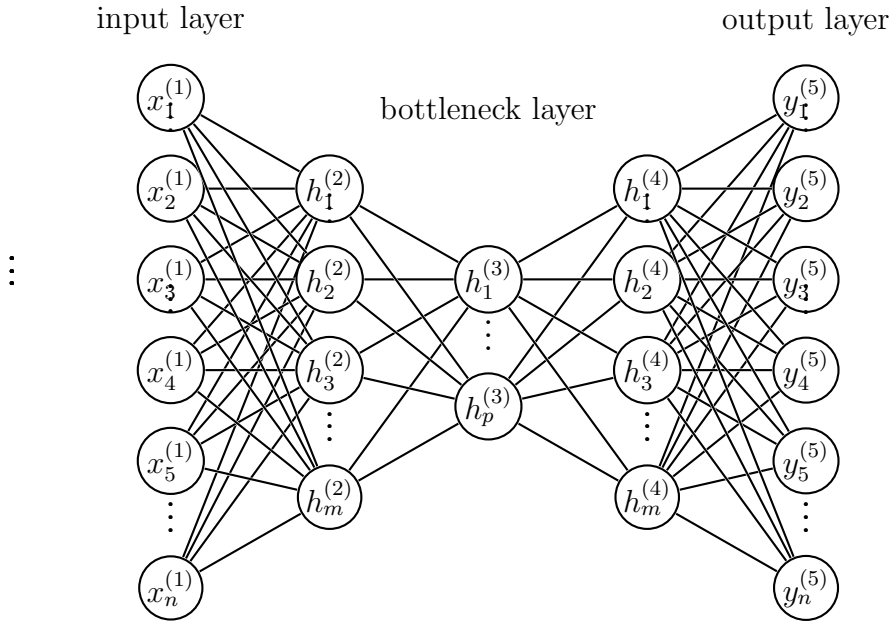

Figure 2.1: Illustrative image of the layer architecture of an autoencoder, consisting of 5 layers. Neurons are denoted by either x (input neuron), h (hidden neuron) or y (output neuron). The superscript denotes the layer in which the neuron is located and the subscript the index position in that layer. For the layer lengths we have n > m > p. In this case the bottleneck layer is the second hidden layer. Its layers are defined by:

$$
\begin{aligned}
x^1 &:= \{x_1^{(1)}, x_2^{(1)}, ..., x_n^{(1)}\} \\
h^2 &:= \{h_1^{(2)}, h_2^{(2)}, ..., h_m^{(2)}\} \\
h^3 &:= \{h_1^{(3)}, h_2^{(3)}, ..., h_p^{(3)}\} \\
h^4 &:= \{h_1^{(4)}, h_2^{(4)}, ..., h_m^{(4)}\} \\
h^5 &= \{y_1^{(5)}, y_2^{(5)}, ..., y_n^{(5)}\}
\end{aligned}
$$

Then these autoencoder components are indentified by their layers:

$$
\begin{aligned}
\text{encoder} &:= \{x^1, h^2\} \\
\text{decoder} &:= \{h^4, y^5\} \\
\text{bottleneck layer} &:= \{h^3\}
\end{aligned}
$$

In the outlier detection pipeline, the first half of autoencoders, the encoder, is used for preprocessing. Before the encoder is extracted out, the weights between all layers in the autoencoder are trained. This is done by minimizing the reconstruction loss using a optimization method, e.g. gradient descent. The reconstruction loss is computed by a loss function. This reconstruction loss or error defines how accurate the decoder decodes the bottleneck representation back into the given input. After extracting the encoder (and thus its weights) from the autoencoder, the application of the encoder on a data set is a means for nonlineair dimensionality reduction, which makes it potentially suitable for preprocessing.

In neural networks, each neuron contains a numerical value. Neurons in the input layer are, by definition, assigned values provided by the input. Subsequently, we define the values for neurons in all next layers. The value of the i-th neuron in the j-th layer, which is of length n will be defined. Given is the length m of the $(j-1)$-th layer. Also given is the activation function σ. Then, the the neuron value $a_i^{(j)}$ defined by equation 2.1 and 2.2.

$$a_i^{(j)} = \sigma(w_{i,1} \cdot a_0^{(j-1)} + w_{i,2} \cdot a_1^{(j-1)} + ... + w_{i,1} \cdot a^{(j-1)} + b) \tag{2.1}$$

$$a_i^{(j)} = \sigma(\sum_{k=1}^{m} w_{k,1} \cdot a_k^{(j-1)} + b) \tag{2.2}$$

where $w_{q,r}$ defines the weight between the q-th neuron in the $(j-1)$-th layer and the r-th neuron in the j-th layer.

Notably, in equations 2.1 and 2.2, we put the symbol 'a' in $a_j^{(i)}$ to express the value of a neuron, not knowing if it occurs in the input layer, any hidden layer or the output layer. Furthermore for the whole j-th layer we define its neuron values by equations 2.3 and 2.4:

$$a^{(j)} = \begin{pmatrix} a_1^{(j)} \\ \vdots \\ a_n^{(j)} \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{1,0} & \dots & w_{1,m} \\ \vdots & \ddots & \vdots \\ w_{n,0} & \dots & w_{n,m} \end{pmatrix} \begin{pmatrix} a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \mathbf{b} \right] \tag{2.3}$$

$$a^{(j)} = \sigma(\mathbf{W}^{(j-1)} a^{(j-1)} + \mathbf{b}) \tag{2.4}$$

2.2 Outlier detection methods

2.2.1 kNN for outlier detection

kNN is known for its use in classification and regression. First we briefly discuss how kNN classification works, after which we explain how an alteration of the algorithm is used for outlier detection.

Using the kNN classification algorithm to classify any data point p in a data set S, the class labels of all other data points in S are known and used to determine a class for p. A distance metric is used defining the distance between each pair of data points. All pairwise distances between p and any other point in S are computed and the data points are ordered in ascending order of pairwise distance. The k first elements in that list are called the k nearest neighbors. Then p is assigned to the class that is present most often among these k nearest neighbors.

The kNN outlier detection algorithm (kNN OD) has a whole different purpose than kNN for classifcation as they have a different type as output. kNN OD computes outlier scores instead of determining class membership assignments. Assume we want to compute the outlier score (a score between 0 and 1 expressing outlierness of a data point) of a data point p in data set S. Similar to kNN for classification, the k data points in S being the nearest to p are computed. Again, this is done by ordering all other datapoints in S on pairwise distance to p, and then collect the first k. Three main methods exist to assign outlier scores and they all use the pairwise distances of the k nearest neighbors of p in S. The first method is to take the absolute distance to the k-th neighbor. The second method is to take the mean among the k nearest neighbors. And the third is to take the median value among the k neighbors.

Although the kNN OD algorithm is now finished, one might want to make a binary classification (i.e. identify outliers and inliers). In this case a threshold value can be used to part these two classes. Let $I \subseteq S$ denote the set of inliers, and $O \subseteq S$ the set of outliers. Now for a given threshold value θ, the data points from set S do have set membership according to 'equation' 2.5.

$$\forall x \in S \begin{cases} x \in I, & \text{if } \sigma(x) \leq \theta \\ x \in O, & \text{otherwise} \end{cases} \quad (2.5)$$

2.2.2 Local Outlier Factor

Local outlier factor, or LOF, is a distance based approach to anomaly detection. It is a local method. LOF calculates a score for each data point that resembles the extent to which it is irregular, compared to its neighbors. For each data point a local density estimation is computed. It uses the distances to its neighbors for the density and compares the density with these neighbors. If the local density estimation of a data point is substantially lower than the local density estimations of its neighbors, it is classified as outlier. Various variables are important for any good application of LOF. Firstly, to find a proper way of establishing that two data points are neighbors. Secondly, the choice of distance measure to use. Third, the choice of density diffentiation when comparing to neighbors and thus determining which are actual outliers. kNN is often used in LOF for obtaining the neighbors of data points. Here picking a suiting k is key. It differs from KNN in the sense that the outlier score in kNN is determined directly from the distance to the k nearest neighbor where in LOF the outlier score is based on a comparison of the density measure to the ones of its neighbors. LOF was first introduced by Breunig et al.[6]

2.2.3 Isolation Forest

The isolation forest approach to outlier detection tries to isolate atypical data points. To isolate an observation, first it partitions the dataset and then randomly selects an attribute and a split value for that attribute and splits the partitions based on that. As outliers tend to be isolated quicker than the *normal* data points, they are recognized by having a small number of splits before isolation (at a small depth of the tree). Isolation Forest as an approach to anomaly detection was initially proposed by Liu et al.[18]

2.3 Evaluation metrics

2.3.1 ROC-AUC

ROC-AUC is an evaluation metric that represents the area under the curve of the receiveing operating characteristic. The true positive rate(TPR is plotted against the false positive rate(FPR). This ROC curve gives insight in the course between TPR and FPR while the cut-off threshold for observations to be classified inlier or outlier changes. The ROC-AUC then is the area under this curve. As the TPR and the FPR are both in the range $[0, 1]$, the ROC-AUC value also is in the range $[0, 1]$. A random classifier has a baseline of 0.5.

Chapter 3

Research

3.1 Methods

We are in the process of finding out whether autoencoders can outperform PCA, normalization and standardization as preprocessor method for outlier detection. To do so we build pipelines consisting of preprocessing method(s) and an outlier predictor. Stratified K-fold cross is used for validation and hyperparameter tuning for pipeline optimization. Features without variability (having 0 variance) are removed beforehand. Figure 3.1 displays the outlier detection process that has been used.

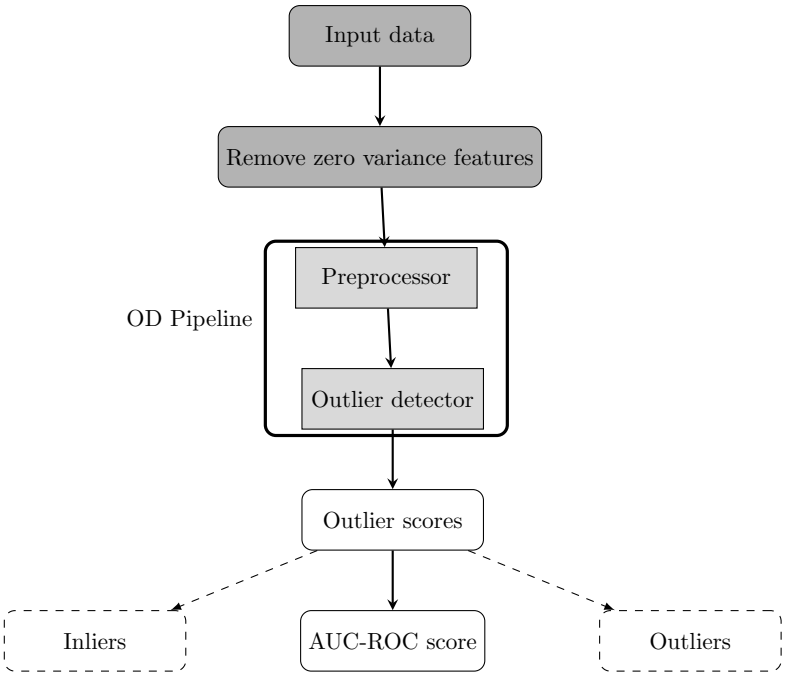

Figure 3.1: Illustration of the outlier detection process from input to evaluation of the outlier detection pipeline. For all data points an outlier score is computed. From these scores, and the outlier labels we compute the AUC-ROC score for a pipeline. We directly evaluate by means of the AUC-ROC score and the dashed nodes only display that a binary classification could be made from the outlier scores, given a threshold value, but also that we do not explicitly do this. Nevertheless, when computing AUC-ROC scores for OD pipelines, classifying outliers and inliers needs to be done multiple times.

3.1.1 Outlier Detection Pipelines

Outlier detection pipelines used in this research are built by combining each of the preprocessors with each of the outlier detection methods. All 12 pipeline combinations are built where the preprocessing methods are given in Table 3.1 and the outlier detectors in Table 3.2. Notably we use *robust* standardization. This is standardization as explained in section 2.1.2 with two adjustments. First, the median is removed. Secondly, the mean and standard deviation used to standardize are computed over a subrange of the input data. In our case, the used subrange is the interquartile range.

Table 3.1: This table displays the used preprocessing methods and their abbreviations.

Robust Standardization + Principal Component Analysis	Stand-PCA
Robust Standardization + Autoencoder	Stand-AE
Robust Standardization	Stand
Normalization	Norm

Table 3.2: This table displays the used outlier detection methods and their abbreviations.

Local Outlier Factor	LOF
Iolation Forest	IFOR
K-Nearest Neighbor Outlier Detection	kNN

3.1.2 Evaluation Metric

The predictor assigns an outlier score to each sample x in the test set where the outlier score $\sigma_x \in [0, 1]$. To evaluate preprocessing methods we use a metric to score pipelines in which these preprocessing methods are embedded in. We compute the area under the receiver operating chareristic (ROC-AUC) by using the predicted outlier scores. We use 4-fold cross validation and compute the average performance over the 4 determined ROC-AUC scores. For each fold the ROC-AUC score is computed by predicting the performance of the hyperparameter tuned model on the outer holdout fold. The hyperparameter tuned model is selected by training on two of the three inner folds and estimating the performance on the inner hold-out fold.

3.1.3 Hyperparameter Tuning

Hyperparameter tuning is a machine learning technique which selects the best model based on hyperparameter optimization. Hyperparameters are model parameters which are not trained within a model itself but still impact the structure and behaviour of a model and thus its performance. Examples include distance metrics and number of layers in a neural network. The values for these parameters are given beforehand. The model is partly predefined and restricted by them. We use the gridsearch method to tune hyperparameters. We use gridsearch for hyperparameter tuning. When gridsearch is applied to tune hyperparameters, all possible combinations of provided hyperparameters are used exhaustively and the performance of each pipeline configuration is evaluated.

We apply hyperparameter tuning on pipelines. As pipelines consist of one or multiple preprocessing method(s) and a outlier detection method, each pipeline configuration gets hyperparameters used in the preprocessing method(s), as well as hyperparameter used in the OD method. The hyperparameter settings that we use are given in Table A.2.

For an image representation of the folding process for hyperparameter tuning, see Figure 3.2 First, an *outer split* is made. There is one *test fold* and $k-1$ *outer train folds*. Then the outer train split gets split into $k-2$ *inner train folds* and a *inner test fold*. For each combination that follows from the sets of hyperparameters the performance is tested on this inner test fold.

Each of the $k-1$ folds in the outer training split is used for cross validation in the gridsearch. This results in k hyperparameter tuned models that are different and they each are tested on the outer test fold. It is with this test that we assign the AUC-ROC score of the OD pipeline. In the end we have k resulting ROC AUC values that come from 4 different models.

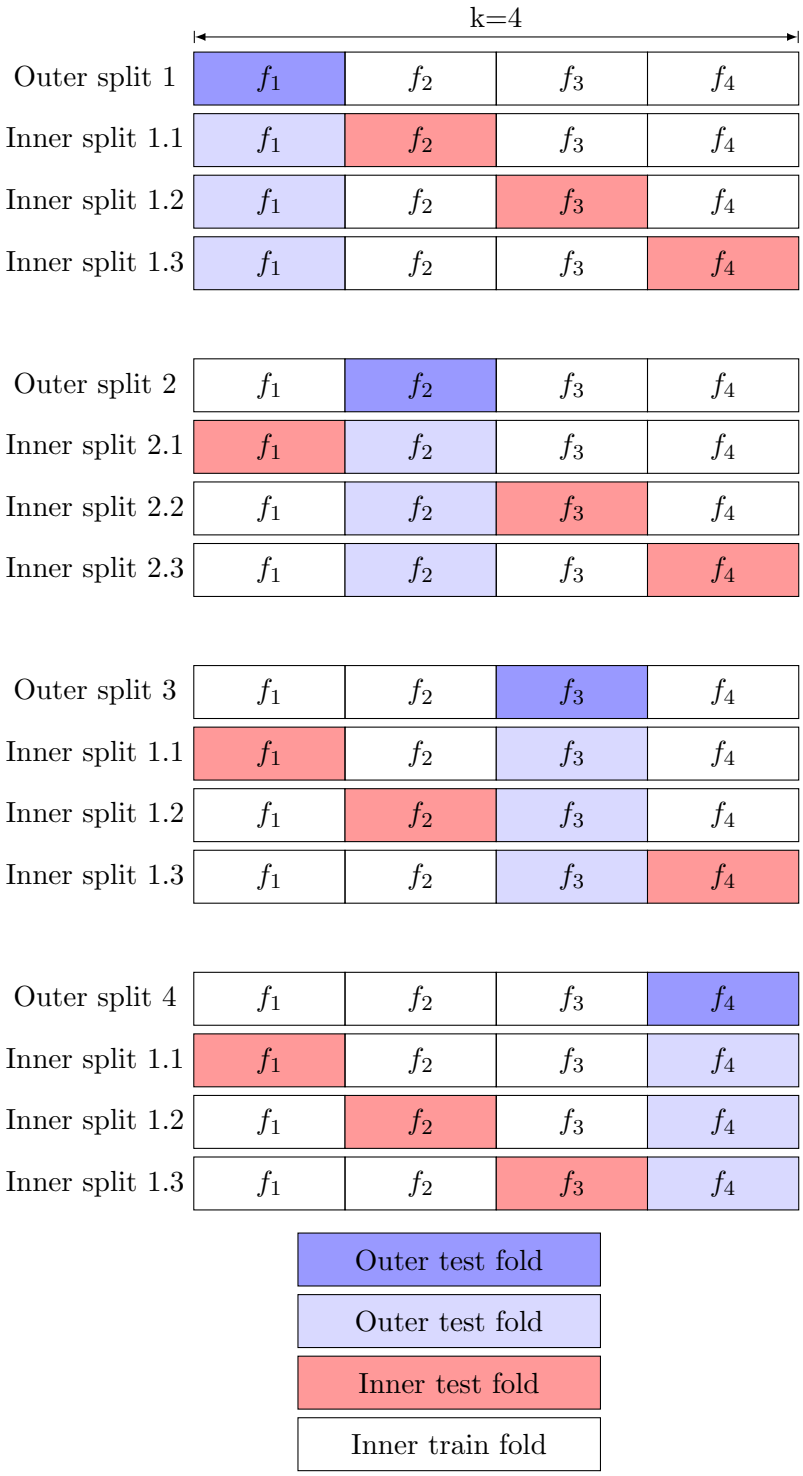

Figure 3.2: An illustration of the splits and folds used during the hyperparameter tuning process.

3.2 Results

In this section the results are presented and discussed in two phases, one for lower dimensional and one for higher dimensional data. Stripplots are used to visualize the results. In the plots, the dots displaying the AUC-ROC scores for the *standardization + autoencoders* method have a horizontal line crossing them to augment it and make it easier to compare the score with scores for other preprocessing methods. A table with all the AUC-ROC scores in one view is given in Appendix A.1.

3.2.1 Lower dimensional data sets

Here we see three stripplots with AUC-ROC scores for three lower dimensional data sets. These three data sets, `Breastw`, `Cardio` and `Thyroid` have 9, 21 and 6 faetures respectively. Comparing the scores for outlier detection pipelines with PCA and scores of outlier detection pipelines with autoencoders, by looking at the right of each figure, we see that most times PCA performs better and in some cases even seem to perform much better than autoencoders.

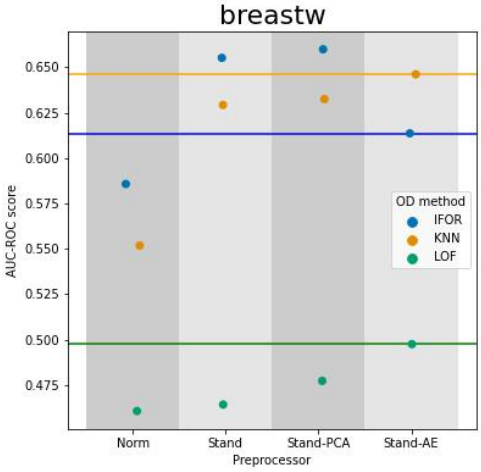

Figure 3.3: Stripplot of the AUC-ROC scores for the used OD pipelines applied to the Breastw dataset.

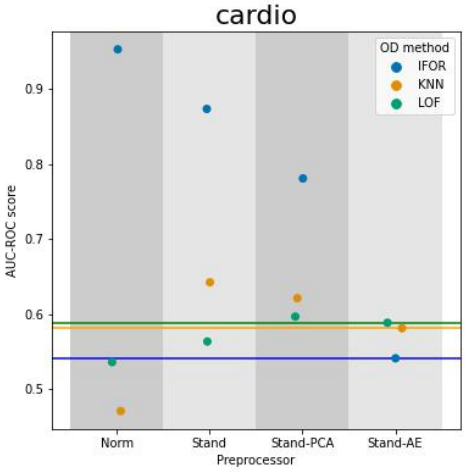

Figure 3.4: Stripplot of the AUC-ROC scores for the used OD pipelines applied to the Breastw dataset.

For example, PCA outperforms autoencoders considerably for IFOR on `Cardio`, and `Thyroid`. Also for LOF on `Thyroid` PCA performs much better. The performance by autoencoders+IFOR is particularly bad, as it scores much worse than the two benchmark methods normalization and standardization. On none of the three data sets autoencoders are part of the best performing OD pipeline. Where autoencoders based OD pipelines perform mostly worse than standalone standardization, OD pipelines with PCA are also not an improvement upon them. Finally, we observer that in general these figures display that PCA performs better than autoencoders as preprocessing step on lower dimensional data.

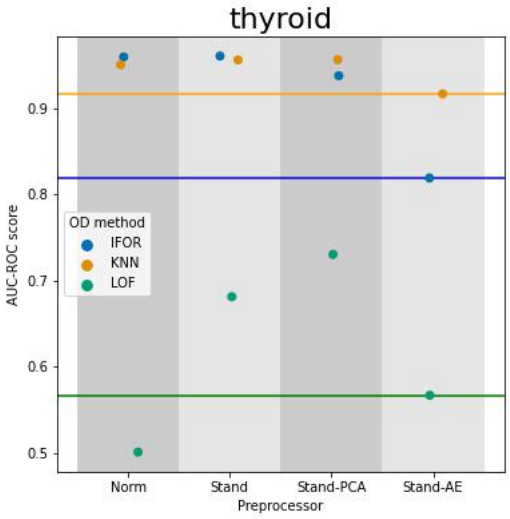

Figure 3.5: Stripplot of the AUC-ROC scores for the used OD pipelines applied to the Thyroid dataset.

3.2.2 Higher dimensional data sets

Here we see plots for scores of the three data sets, `Musk`, `Arrhythmia` and `Speech` which have 166, 274 and 400 faetures respectively.

Comparing autoencoder with PCA we see for `Musk` that IFOR perform better if PCA is used a preprocessing step compared to autoencoders as preprocessing step. kNN and LOF show comparable results. In the context of this data set being higher dimensional it is remarkable that the scores for Isolation forests are best for the benchmark methods normalization and standardization. The initial hypothesis was that dimensionality reduction techniques, PCA and Autoencoders, would proof to be favourable. Here should be noted that isolation forest is known to do well on higher dimensional data, when compared to distance based metrics, like LOF and kNN. The same pattern holds for IFORs and kNN on the `Speech` sets, where we also see the dimensionality reduction techniques perform worse.

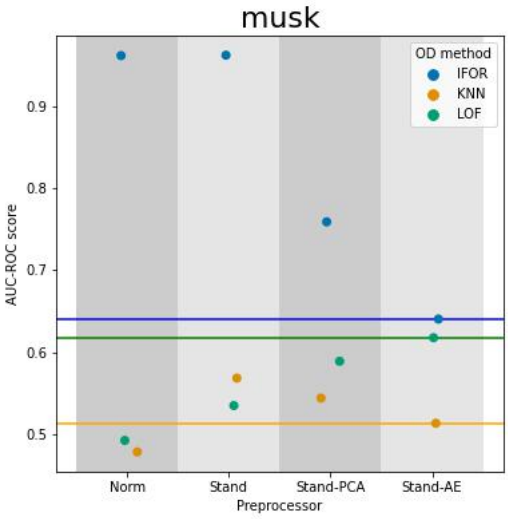

Figure 3.6: Stripplot of the AUC-ROC scores for the used OD pipelines applied to the Musk dataset.

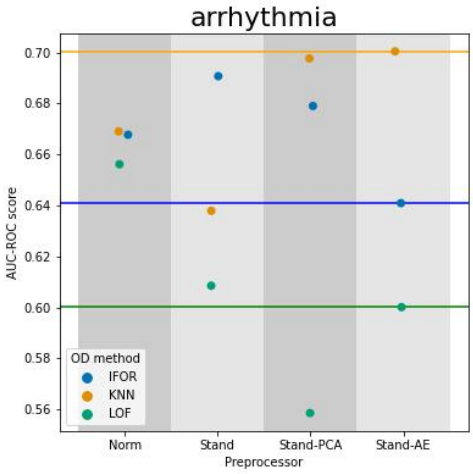

Figure 3.7: Stripplot of the AUC-ROC scores for the used OD pipelines applied to the Arrhythmia dataset.

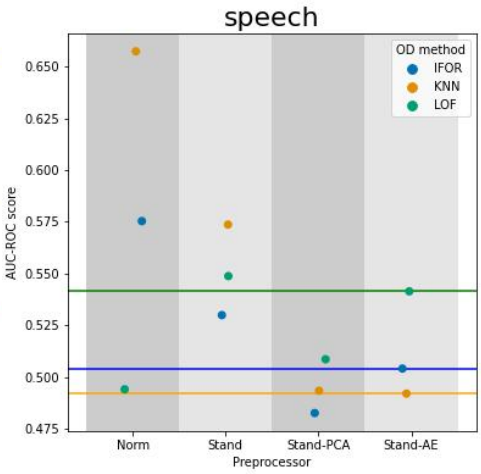

Figure 3.8: Stripplot of the AUC-ROC scores for the used OD pipelines applied to the Speech dataset.

3.3 Discussion

Contrary to our expectations, our research does not suggest that autoencoders provide a feasible way of preprocessing data for outlier detection. Therefore the hypothesis is rejected. We discuss this rejection in sections 3.3.1 and 3.3.2. After that, we defend why we used the OD techniques and evaluation metric we used, in sections 3.3.3 3.3.4. Finally, we discuss the supervised context in which we evaluate the performance of OD pipelines in section 3.3.5.

3.3.1 Number of datasets

Although we have found some implications that autoencoders are probably not a feasbile preprocessing method for outlier detection, we have to admit there is some uncertainty in those conclusions. From the results shown we can not derive that the differences are significant. We only used 6 datasets, 3 lower and 3 higher dimensional. Therefore we can not be sure the implications of the results, in the way they are formulated, generalize across more datasets and in particular across data sets with different types of distributions and dimensionality.

3.3.2 Autoencoder depth

The autoencoders used have a rather small number for the bottleneck layer. The depth of the autoencoders is low. We can therefore not be sure that our conclusion generalizes across deeper autoencoders.

Partly, the reason for picking a small bottleneck layer is, although we use gpu acceleration for performance enhancement, it still costs significant computing time to train one individual model. As we perform hyperparameter tuning for picking the right hyperparameters for each pipeline, we have to train a lot of possible configurations and training time becomes quite extensive. For example, let's look at an autoencoder model with kNN as the final step in the pipeline. For the autoencoder we define 2 activation functions, 3 different values for the branching factor and 2 different values for the bottleneck index layer. Then for the kNN parameter set we have 30 values for k, 3 ways of computing the outlier score (mean, largest and median distance of or towards k nearest neighbor(s)) and 2 distance metrics. All this together comes down to (2*3*2)*(30*3*2) = 2175 models to be trained within a single inner fold. As we do 4-fold crossvalidation on the outer split and inside have 3 inner folds for hyperparameter tuning, this amounts to 4*3*2175=26100.
A last remark on autoencoders used for OD. They might not be an improvement as preprocessing step for outlier detection, but autoencoders, either basic or variational, may be a good fit as direct outlier detection method.

3.3.3 OD techniques

We have evaluated the performance of three outlier detection techniques. These were kNN, IFOR and LOF. We selected these techniques on basis of the comparative work of Bouman[1][1]. Bouman compares 18 outlier detection techniques on 34 multivariate datasets. In this comparison, kNN overall performs best. On the subset of datasets, for which outlier detection techniques designed to find local outliers, generally score better, an IFOR derived method (EIF, extended isolation forest) performs best. For the "global" subset of datasets, COF performs best. Since these methods perform best, we use kNN and IFOR. We also use LOF. We use LOF instead of COF. LOF is similar to COF as outlier detection technique in the sense that they are both local methods, both comparing some local density estimations to the local density estimations of data points nearby. COF is a variation on LOF, and they differ in the way they find the k nearest neighbors used for the local density estimation. COF makes use of chain distance as local density estimation. kNN and LOF leverage local density estimations and IFOR takes a total different approach. Therefore we cover multiple bases.

3.3.4 Evaluation metric

Picking ROC-AUC as outlier detection scorer is based upon the conclusions about the metric in the work of Zimek et al.[25] In their survey, advocate using ROC-AUC as it inherently makes up for the class imbalance problem because it not only plots the true positive rate against the false positive rate, but does this while increasing the threshold at which observations are classified as outlier versus inlier.In the context of the class imbalance problem, ROC-AUC is a sensible metric choice for evaluation of outlier detection models. Paraphrasing Zimek, ROC-AUC is the best we have but this does not make it perfect.
In many cases specificity and sensitivity are not equally important. For example, in the domain of medical screening we can accept false positives, as it "only" results in stress and more medical costs for further investigation. False negatives however could do a lot bigger harm as not treating the illness can inflict increased suffering and worse. As indicated by Halligan et al.[13] ROC-AUC does not differ between specificty and sensitivity and therefore is not always a suitable metric for outlier detection. Although there are concerns with the metric, we think the benefits outweigh the costs and use it as our evaluation metric.

[1] submitted for peer review

3.3.5 Supervised evaluation

For assigning scores to outlier detection pipelines on a data set the average ROC-AUC score over the folds is used. For each split a the best scoring model on that split is used to predict on the held out fold. This pushes the score upward a bit, compared to scoring all parameter settings for a given split on the data and selecting the average best performing set of parameters over all splits for the final predicted score. On the contrary, we use a hold out fold twice (one in the outer split and once in the inner split) and that has a negative impact on the score. Subsequently, the context in which outlier detection models are evaluated is one in which we pick the best performing configuration for each OD pipeline and evaluate its performance on the hold out set. This makes our evaluation of OD pipelines a supervised learning approach. If, for all OD pipelines, the average performance over all configurations of an OD pipeline was used to evaluate its performance, then the evaluation would be unsupervised. We may assume that our choice for best perfomer affects our results.

Chapter 4

Related Work

David and Clark[10] review preprocessing for network intrusion detection. They identify that faeture scaling techniques normalization and stardization are being used often.Furthermore, they observe that some of the papers they review apply PCA to reduce the number of dimensions.

Chen et al.[9] leveraged a KNN-model to classify outliers in the Wisconsin Breast Cancer dataset and retrieved full coverage (100%) of the 39 predefined set of outliers with a cut-off of 14% where 8% were predefined to be outliers. They do not mention specific preprocessing techniques that they have applied but remove some malignant instances from the set and thus remove some of the outliers preceding model training.

Zimek et al.[25] identify a dichotomy of outlier detection methods when it comes to addressing data in high dimensional Euclidean space. For the definition of outliers one class does consider subspaces and the other does not. In the same survey they explain that some methods exclusively need normalization and some OD methods would be too biased to certain features when standardization is not applied. This would happen in cases where a single or a few value(s) in a feature are a lot greater than the other, effectively dominating the scale.

Ahmed et al.[2] use three outlier detection methods. These are FBOD (Feature Bagging for Outlier Detection, Lazarevic and Kumar[17], SOD (Subspace Outlier Detection) and LOF. By means of the latter Ahmed et al. indentify two important threshold values for the hydropower generation plants anomalies, being oil temperature and bearing harmonics values.

Chapter 5

Conclusions

We have applied PCA and autoencoders as preprocessing step for three outlier detection methods on six OD benchmark data sets. On data sets with a low number of dimensions, PCA mostly outscores autoencoders slightly and in a few cases considerably. Autoencoders, again in most cases, perform worse than the benchmark methods normalization and robust standardization.

On data sets with a high number of dimensions, we see that isolation forests perform better when they are preceded by PCA instead of being preceded by autoencoders. Contrary to our expectations the dimensionality reduction methods are often outperformed for all three tested outlier detection methods, by standardization and normalization. Finally we conclude that, PCA often outperforms autoencoders.

Considering the prior, we reject the hypothesis that autoencoders outperform normalization, standardization and PCA as preprocessing step for outlier detection. Our work does not suggest that the capacities of autoencoders as preprocessing method for outlier detection provide a bright future. Nevertheless, we suggest that the performance of autoencoders as preprocessing step for outlier detection are further investigated in future research. Also, in order to gather more certainty on the poor performance of shallow autoencoders as preprocessing step for outlier detection, we suggest that our used method is applied on more datasets in future research.

Bibliography

[1] A comparison of anomaly detection methods, 2022.

[2] Imtiaz Ahmed, Aldo Dagnino, Alessandro Bongiovi, Yu Ding, 2018 IEEE 14th International Conference on Automation Science, and Germany 2018 Aug. 20 . 2018 Aug. 24 Engineering (CASE) Munich. *Outlier Detection for Hydropower Generation Plant*, pages 193–198. 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE). IEEE, 2018.

[3] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

[4] Ibtisam Mohammed Al-Bahri, Sallam Osman Fageeri, Aiman Moyaid Said, and G. Mary Amirtha Sagayee. A comparative study between pca and sift algorithm for static face recognition. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pages 1–5, 2021.

[5] Richard E Bellman. Adaptive control processes. In *Adaptive Control Processes*. Princeton university press, 2015.

[6] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jorg Sander. Lof identifying density-based local outliers. *ACM SIGMOD Record*, 29(2):93–104, 2000. 93.

[7] Banerjee A. Chandola, V. and V. Kumar. Anomaly detection: A survey. acm comput. surv. 41. volume 3, pages 1–58, 2009. Article 15 (July 2009).

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[9] Yumin Chen, Duoqian Miao, and Hongyun Zhang. Neighborhood outlier detection. *Expert Systems With Applications*, 37(12):8745–8749, 2010. 8745.

[10] Jonathan J. Davis and Andrew J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers Security*, 30(6):353–375, 2011.

[11] Daniel Gonzalez, Miguel A. Patricio, Antonio Berlanga, and Jose M. Molina. Variational autoencoders for anomaly detection in the behaviour of the elderly using electricity consumption data. *Expert Systems*, 39(4), 2022.

[12] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. 2009.

[13] Steve Halligan, Douglas G Altman, and Susan Mallett. Disadvantages of using the area under the receiver operating characteristic curve to assess imaging tests: a discussion and proposal for an alternative approach. *European radiology*, 25(4):932–939, 2015.

[14] Fouzi Harrou, Ying Sun, and Sofiane Khadraoui. Amalgamation of anomaly-detection indices for enhanced process monitoring. *Journal of Loss Prevention in the Process Industries*, 40:365–377, 2016. 365.

[15] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[16] Irem Ersöz Kaya, Ayça Çakmak Pehlivanlı, Emine Gezmez Sekizkardeş, and Turgay Ibrikci. Pca based clustering for brain tumor segmentation of t1w mri images. *Computer Methods and Programs in Biomedicine*, 140:19–28, 2017.

[17] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 157–166, New York, NY, USA, 2005. Association for Computing Machinery.

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[19] Esma Mujkic, Mark P. Philipsen, Thomas B. Moeslund, Martin P. Christiansen, and Ole Ravn. Anomaly detection for agricultural vehicles using autoencoders. *Sensors (Basel, Switzerland)*, 22(10), 2022.

[20] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[22] Gabriel San Martin, Enrique Lopez Droguett, Viviane Meruane, and Marcio das Chagas Moura. Deep variational auto-encoders: A promising tool for dimensionality reduction and ball bearing elements fault diagnosis. *Structural Health Monitoring*, 18(4):1093, 2019. 1093.

[23] Arie Sheinker and Mark B. Moldwin. Magnetic anomaly detection (mad) of ferromagnetic pipelines using principal component analysis (pca). *Measurement Science and Technology*, 27(4), 2016.

[24] Arijit Ukil, Soma Bandyoapdhyay, Chetanya Puri, and Arpan Pal. Iot healthcare analytics: The importance of anomaly detection. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 994–997, 2016.

[25] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5):363–387, 2012. 363.

Appendix A

Appendix

A.1 Tables

	Arrhythmia	Breastw	Cardio	Musk	Speech	Thyroid	Average
Norm-IFOR	0.67	0.59	0.95	0.96	0.58	0.96	0.78
Norm-KNN	0.67	0.55	0.47	0.48	0.66	0.95	0.63
Norm-LOF	0.66	0.46	0.54	0.49	0.49	0.50	0.52
Stand-IFOR	0.69	0.66	0.87	0.96	0.53	0.96	0.78
Stand-KNN	0.64	0.63	0.64	0.57	0.57	0.96	0.67
Stand-LOF	0.61	0.46	0.56	0.53	0.55	0.68	0.57
Stand-PCA-IFOR	0.68	0.66	0.78	0.76	0.48	0.94	0.72
Stand-PCA-KNN	0.70	0.63	0.62	0.54	0.49	0.96	0.66
Stand-PCA-LOF	0.56	0.48	0.60	0.59	0.51	0.73	0.58
Stand-AE-IFOR	0.64	0.61	0.54	0.64	0.50	0.82	0.63
Stand-AE-KNN	0.70	0.65	0.58	0.51	0.49	0.92	0.64
Stand-AE-LOF	0.60	0.50	0.59	0.62	0.54	0.57	0.57

Table A.1: AUC-ROC scores with all pipeline combinations in the rows and datasets as columns

Method	Hyperparameter	Hyperparameter sets
Norm	Norm__norm	['l2']
	Norm__copy	[True]
Stand	Stand__scaling	[True]
	Stand__with_centering	[True]
	Stand__quantile_range	[(25.0, 75.0)]
PCA	PCA__n_components	[2,4,6,8,10]
	PCA__random_state	random_state*
AE	AE__act	['sigmoid', 'relu']
	AE__frac	[1/3,1/2,2/3]
	AE__bttl	[2,3]
IFOR	IFOR__n_estimators	[1000]
	IFOR__max_samples	[128,256]
	IFOR__max_features	[1.0]
	IFOR__bootstrap	[True,False]
	IFOR__random_state	random_state*
LOF	LOF__n_neighbours	[10,...,30]
	LOF__metric	distanceMetrics*
KNN	KNN__n_neighbours	[10,...,30]
	KNN__method	["mean", "largest", "median"]
	KNN__metric	distanceMetrics*

Table A.2: Overview of all hyperparamater settings used for hyperparameter tuning in this research. * random_state and distance metric values can be found in table A.2.AE__frac defines the fraction of neurons each layer left to the bottleneck layer has as a fraction of its predecessor. AE__bttl defines the bottlneck layer. If AE__bttl = 2, the second layer is the bottleneck layer. AE__act is the activation function for the neurons in the autoencoder. We always used Adam as optimizer for training of the autoencoders.

Parameter set name	Parameter set value(s)
random_state	1173838490
distanceMetrics	[”manhattan”, ”euclidean”]

Table A.3: Helper table for sets that are used multiple times in A.2. random_state is used as a seed for folding but when comuting SVD it may be the case that not the full exact SVD is computed. Computing the exact full SVD is the default, but Halko et al.[12] is used in cases where the number of features and samples exceed 500 and more than 80% of the input features is kept, since this method is more efficient.

A.2 Code

Code is written in python. We use the packages *scikit-klearn, tensorflow* and *keras*. The sklearn library GridsearchCV is used for hyperparameter tuning of Pipelines (scikit-learn objects) which consist of preprocessors (transformers) and outlier detection methods (fitters). For the autoencoder we wrote our own estimator by subclassing the CustomModel of Keras neural networks. We implemented the methods

```
set_params(), get_params() and clone()
```

properly so that GridsearchCV actually tests for different parameters such that it sets the parameters to new, to be tested, values.