Automatic Tuning of Machine Learning for Robust and Reliable Anomaly Detection

Rama Kassoumeh [RK] Shubham Gupta [SG] Sohith Dhavaleswarapu [SD] Khandakaer Abir Hossain [KAH] Imene Kolli [IK] Nishat Tasnim Ahmed Meem [NTAM] Kartik Kamboj [KK] Umid Kumar Dey [UKD] Upanishadh Iyer [UI]

Abstract

[UKD]Anomaly detection is a critical task in various domains, such as intrusion detection, fraud detection, and fault detection. Machine learning algorithms have shown significant potential in detecting anomalies. However, machine learning models require manual tuning to achieve optimal performance, which is a time-consuming and challenging task. In this paper, we propose an automatic tuning framework for machine learning models used in anomaly detection. Our framework optimizes the hyperparameters of machine learning models by leveraging grid search. We evaluate our approach on several real-world datasets, and the experimental results demonstrate that our framework achieves better performance than manually-tuned models, which is further enhanced by the preprocessing of input data. Our proposed framework can significantly reduce the manual efforts required for tuning machine learning models and improve the efficiency and effectiveness of anomaly detection. With a lot of data to work on, the found hyperparameters can then be generalised across the datasets to produce even faster results.

1. Introduction

[NTAM] Anomaly detection is an important task in the field of machine learning as it helps to find rare or unusual events in a dataset. By identifying the anomalies the performance of machine learning models can be improved which will lead to better decision-making. This approach can provide insights into a system's behavior in many real-world scenarios. For example, anomaly detection can identify fraudulent activities in credit card transactions. In industrial sectors, it can identify any defects or equipment malfunctions that may go unnoticed. However, there are several challenges that make anomaly detection a challenging task. The lack of labelled data makes it difficult to train a machine learning model to detect anomalies. Another challenge is defining the normal and anomalous behavior of the data as it can defer from person to person sometimes. Noise in the data makes it difficult to distinguish between real anomalies and random fluctuations.

[KAH] Unlike supervised approaches, unsupervised methods do not require labeled data to detect anomalies. Rather, unsupervised methods deal with the underlying structure of the data, from which any deviation is regarded as anomaly (Varun Chandola, 2009). For instance, when clustering algorithms like K-means or hierarchical clustering are concerned, anomalies are data points that do not fit into any cluster or establish their own cluster. The ability of unsupervised approaches to detect a wide range of anomalies using only intrinsic information about the data, makes it widely used for anomaly detection because labeled data may be expensive to obtain in many real-world applications.

[KAH] Hyperparameter tuning is a crucial part of both supervised and unsupervised algorithms to optimize the performance of models. When the number of hyperparameters is large for a model, it leads to an exponential increase of possible combinations, causing hyperparameter tuning complicated in general. However, it gets even more challenging in our context of anomaly detection as it comes to optimizing the unsupervised models without any prior knowledge or labeled data (Jonas Soenen, 2021). Interaction between hyperparameters causes another challenge in hyper parameter tuning. For instance, hyperparameters of a neural network include the number of layers, number of neurons, regularization, learning rate, etc. When the number of layers gets increased, it also leads to an increase in the model's complexity which may necessitate a higher learning rate that stops the model from getting trapped in a local minimum. Similarly, increased regularization strength helps the model avoid overfitting which may need a decreased learning rate to compensate for the reduced capacity of the model. Overall, it is intended to find an optimal combination of hyperparameters that also consider a trade-off between the complexity of the model and generalization performance.

[NTAM] The main goal of this case study is the automatic tuning of some unsupervised machine-learning approaches to detect anomalies. Various algorithms are applied to find the anomalies on 40 training datasets. A python library called Flaml is used to find the best values of the hyperparameters of every algorithm individually for each of these 40 datasets. The optimal hyperparameters for each algorithm are then found by using the measures of central tendency such as mean, median or mode on previously found 40 sets of hyperparameters. Finally, the algorithms are run on 10 test datasets with these optimal parameters and also with the default hyperparameters of the algorithm and the performances of these two approaches are compared using the AUC (area under the ROC curve) scores.

[NTAM] Section 2 discusses the related works and provides an overview of existing research for anomaly detection. The Solution Framework in section 3 outlines the proposed framework for the automatic tuning of machine learning for anomaly detection. Section 4 presents the specific algorithms used in the proposed framework and their implementation details. In Section 5, a comparison of the performance of these algorithms on various datasets is discussed to evaluate their effectiveness. Finally, the Summary section provides a brief summary of the key findings of this report and discusses the implications of the results and the Conclusion in section 7 concludes the report.

2. Related Work

[KAH] To assess the importance of latency space on autoencoders, some previous works related to the impact of a bottleneck for anomaly detection were analyzed. Some papers earlier suggested that, if a bottleneck is not imposed on autoencoder, it reconstructs any inputs perfectly and reconstruction error gets too low, making the model useless in anomaly detection (Ruff et al., 2018a). However, some recent works contradicted this conventional belief and demonstrated better performance with non-bottlenecked autoencoders. In the paper (Bang Xiang Yong, 2022), some techniques to remove the bottleneck were discussed initially. One of them was to expand the latency dimension which can be related to our case of autoencoder (over).

Implementing a bottleneck implies having a latency size smaller than input dimension $\dim(\mathbf{z}) < \dim(\mathbf{x})$. On the contrary, a bottleneck is removed if size of the latent dimensions gets expanded, expressed as $\dim(\mathbf{z}) \geq \dim(\mathbf{x})$. Another way to eliminate the bottleneck is using a long-range skip connection which allows the data flow of each layer to skip the bottleneck (Bang Xiang Yong, 2022). Extensive experiments were conducted using pairs of unrelated image datasets such as FashionMNIST (H. Xiao, 2017) vs MNIST (Deng, 2017), CIFAR (A. Krizhevsky, 2009) vs SVHN (Y. Netzer, 2011), eight datasets from the Outlier Detection Datasets (ODDS) (Rayana, 2016) including Cardio, Optdigits, Lympho, Pendigits, Ionosphere, Thyroid, Pima and Vowels as well as sensor data including ZeMA (T. Schneider, 2018) and STRATH (C. Tachtatzis, 2019) datasets. Performance of one bottlenecked and three non-bottlenecked models were analyzed using the metric area under the curve (AUC) of receiver operating characteristic graph.

Most non-bottlenecked models outperform the bottlenecked model with mean AUC score ≥ 0.8 (Bang Xiang Yong, 2022). It implies that even with a latency size higher than the number of features or using skip connections, the models perform well to detect anomalies. Furthermore, the non-bottlenecked models provided the best AUC scores on most datasets except in the case of FashionMNIST vs MNIST where the bottlenecked model performed better with a very close margin (Bang Xiang Yong, 2022). For the CIFAR vs SVHN pair, the non-bottlenecked models named as BAE-Ensemble and BAEB-∞ performed much better than what was achieved in some previous works (E. Nalisnick, 2019). It indicates that if the earlier works would consider non-bottlenecked models, they could achieve better AUC scores.

Some limitations of non-bottlenecked approaches here include the sole focus on unsupervised anomaly detection and ignoring other cases where the bottleneck might be necessary such as clustering and dimensionality reduction (Bang Xiang Yong, 2022). Different types of datasets may also demand the need for bottlenecked autoencoder. In brief, the paper demonstrates the possibility of achieving better performance with non-bottlenecked models and recommends not to be limited to only bottlenecked models in future works.

[UKD] (Dey et al., 2019) use grid search to find the best hyperparameters to analyse genetic expression data to predict type of cancer using Machine Learning Techniques. In this study, it was found out that using optimised hyperparameters leads to better prediction.

3. Solution Framework

3.1. Evaluation metric

[KK] The evaluation metric used in this case study is AUC score which is the area under the ROC curve, as shown in Figure 1, plotted with two parameters,

- TPR (true positive rates)
- FPR (false positive rates)

The AUC score ranges from 0 to 1. AUC score of 0.0 means model predictions are 100% wrong, while AUC score of 1.0 means model predictions are 100% correct (Sammut & Webb, 2011).

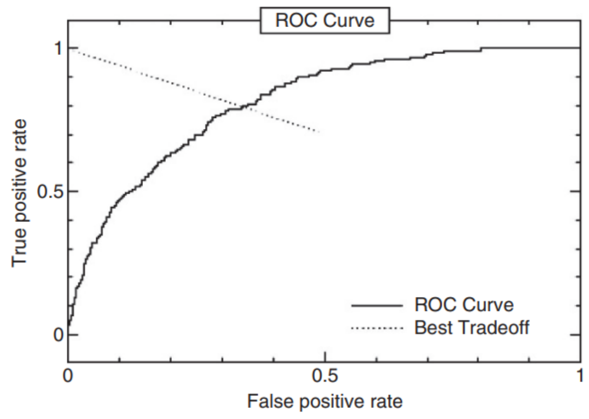


Figure 1. Area Under the ROC Curve. (Sammut & Webb, 2011)

3.2. Flaml: A Fast and Lightweight AutoML Library

[KK] Flaml (Fast and Lightweight Automated Machine Learning) is a Python package for automating and adjusting machine learning. It offers a rapid auto-tuning tool that is powered by a novel, cost-effective tuning approach that chooses ideal hyperparameters based on new training data. It can also optimize pipelines, mathematical or statistical models, algorithms, computing experiments, software setups, and so on.

It is one of the libraries used in large language models such as OpenAI GPT3 because it quickly finds quality models for user-defined data efficiently.

It has a built-in model called 'tune', which tunes the hyperparameters of the algorithm in three steps (tuning bjective, constraints, and search space of the hyperparameters) (Wang et al., 2021).

3.3. Framework

[KK] Firstly, the algorithm is performed on 40 training datasets and using the default settings of the hyperparameters, we obtain the AUC scores. In order to improve them, we tune the algorithm's hyperparameters using the Flaml library. The algorithm is now being run again on 40 training datasets with the customized configuration of the hyperparameters. As a result, each dataset generates a set of tuned hyperparameters and hence total 40 sets of tuned hyperparameters are generated.

Using metrics of central tendency (mean, median, or mode), we extract the best-tuned hyperparameters from the 40 sets generated above. Evaluate the best-tuned hyperparameters on 10 testing datasets and compare them with the default parameters. In further evaluation, instead of using any central tendency measures, we trained a decision tree and figure out which depth of the tree works better with the testing data in evaluation. Finally, the AUC scores of the best tuned hyperparameters are compared and analyzed with the standard parameter scores.

4. Algorithms

4.1. Isolation Forest (Iforest)

4.1.1. Algorithm

[RK] Isolation Forest is an unsupervised technique for anomaly detection. It is based on Random Forests that use decision trees. In contrast to decision trees, Isolation Forest constructs isolated trees, where each leaf node contains only one data point, rather than partitioning the data one feature at a time until each partition is homogeneous (Liu et al., 2012).

The concept behind the Isolation Forest algorithm is that anomaly points are "few and different", making them easier to isolate compared to normal points. The Isolation Forest builds an ensemble of isolated trees to target different anomalies and to improve the accuracy and robustness of the anomaly detection process. By combining the results of multiple isolation trees, the ensemble can better capture the structure of the data and detect anomalies that might not be identified by a single tree. The ensemble approach also helps to reduce the risk of overfitting and improves the generalization performance of the model (Liu et al., 2012).

To train the Isolation Forest algorithm, a dataset sample is taken and multiple trees are built. The trees randomly select a feature and partition it between the maximum and minimum values. Anomaly points are isolated much faster than normal points, as normal points tend to be close to each other and anomaly points tend to be far apart. Consequently, normal points occur in dense regions, while anomaly points occur in sparse regions. As a result, the points with the shortest path length are most likely to be anomalies (Liu et al., 2008).

For any anomaly detection method, an anomaly score is needed. the anomaly score $s(x, n)$ in the Iforest algorithm is calculated for each data point x as (Liu et al., 2012):

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{1}$$

Where $E(h(x))$ is the average of $h(x)$ from a collection of isolation trees, $h(x)$ is the path length from the root node to the point x, and $c(n)$ is a normalization factor that depends on the number of data points in the dataset. The higher the anomaly score, the more anomalous the data point is considered to be as shown in "Fig. 2", where the red points are anomaly points that have a shorter path to reach from the root. An appropriate threshold for the anomaly score can be chosen to determine which data points should be classified as anomalies.

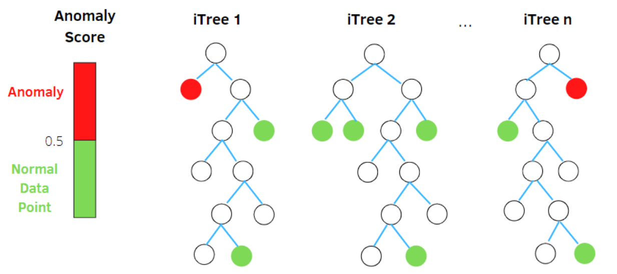


Figure 2. Isolation Forest. (Datrics)

4.1.2. Experiment settings and results

During the tuning process, four hyperparameters were selected for the Iforest algorithm, namely $n\_estimators$, $contamination$, $max\_features$, and $random\_state$. Table 1 presents both the default hyperparameters as well as the best hyperparameters used during the testing phase, along with the AUC scores obtained when using the default and optimal parameters. As all values are numerical, the mean value was chosen as the best hyperparameter. Consequently, there was no significant difference between the default and best hyperparameters, and the AUC score improved by 1%.

To sum up, utilizing the default hyperparameters of the Isolation Forest algorithm proved to be effective as the best-chosen hyperparameters in terms of complexity, time consumption, and memory requirements. Additionally, the algorithm is capable of handling high-dimensional data. However, it still faces challenges in dealing with categorical variables (Liu et al., 2012).

4.2. Generative Adversarial Network (GAN)

4.2.1. Algorithm

[RK] A generative adversarial network (GAN) is a highly successful technology that is widely used for generating high-resolution images. It is composed of two deep neural networks and is trained using an unsupervised approach. GAN has diverse applications in generative fields such as sampling and data generation, missing feature extraction, image reconstruction, representation learning, text generation, music generation, and outlier detection (Goodfellow et al., 2020).

The Generative Adversarial Network (GAN) utilizes two deep neural networks to generate data. These two networks, the generator and the discriminator compete against each other in a game-like training process. The generator network takes random noise and tries to generate data samples that resemble the real data distribution, On the other hand, the discriminator takes in both real data samples and the fake samples generated by the generator and tries to differentiate between them and provides a probability of whether the input is real or not. The two networks are trained simultaneously, where the generator tries to fool the discriminator by generating samples that look like real samples. In contrast, the discriminator tries to classify the real and fake samples correctly. By repeatedly this training process, GAN can generate new data instances that resemble the real ones (Han et al., 2021).

The anomaly problem is as the follows for AnoGAN type, which is used in this paper: X of N normal data samples from the unknown distribution $p_{data}(x)$, $X = \{x_n \in \mathbb{R}^d : n = 1, ..., N\}$ and used as training samples in d dimensions. The generator denotes $G(z)$ where z is the random noise of distribution $p_z(z)$ and tries to resemble the data x by learning x distribution, while the discriminator denotes as $D(.)$ (Han et al., 2021) (Schlegl et al., 2017). The generator's and the discriminator's networks are trained simultaneously with a two-player minimax game of the function $V(G, D)$ (Schlegl et al., 2017):

$$\min_G \max_D V(G, D) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (2)$$

The discriminator trains to maximize the probability assigned to the real sample and at the same time the generator trains to minimize $\log(1 - D(G(z)))$, i.e fools the discriminator via pushing the discriminator's output of the fake sample toward 1.

During the training and testing, a measurement is needed to distinguish the anomaly samples from the normal ones. In this case, an "Anomaly score" should be defined, and the higher an anomaly score is, the more likely the sample is an anomaly, The Anomaly score is calculated as (Kim et al., 2021):

$$A(x) = (1 - \lambda).R(x) + \lambda.D(x) \quad (3)$$

where λ is a learning rate and the residual score is defined as $R(x) = \sum |x - G(z)|$, the difference between the original data and the generated one, and the discriminator score is defined as $D(x) = \sum |f(x) - f(G(x))|$, where $f(.)$, the intermediate feature representation of the discriminator, is used instead of the scalar output of the discriminator (Kim et al., 2021).

4.2.2. Experiment settings and results

There are various types of GANs, each with distinct architectures for the generator and discriminator neural networks. For this particular case study, the AnoGAN type was selected for both the tuning and the testing (Schlegl et al., 2017). Ten hyperparameters were chosen for the GAN algorithm, the hidden activation function, output activation function, number of layers for the generator, number of layers for the discriminator, number of neurons for each layer in the generator and discriminator, learning rate, dropout rate, number of epochs, and batch size.

Tuning was performed on 40 distinct datasets, and the subsequent testing was conducted on 10 additional datasets. Knowing that the tuning and the testing were time-consuming depending on the dataset size and the number of features, especially for tuning the number of layers and the number of nodes that are combined together in one list as shown in table ”Table 2”. ”Table 2” displays both the default hyperparameters (Zhao et al., 2021), the best hyperparameters that were utilized in the testing phase, and the AUC scores of the test using the default and the optimal parameters. The most frequently occurring values were selected as the best hyperparameters for string values, whereas mean values were selected for numerical values. As a result, no significant difference is observed between the default and best hyperparameter, even though there is a 4% enhancement in the AUC score.

However, after tuning the parameters, interesting findings have been noticed. The most frequent activation function for the hidden layers was the ”None” activation function, as demonstrated in ”Fig. 13”. Without the activation function, each neuron in the hidden layers would simply perform a linear transformation on its input and pass it to the next layer. The activation functions play a critical role in enabling neural networks to learn complex patterns and relationships in the data, and not using them for the hidden layer would limit the capabilities of the network. In the testing phase, the hidden layer activation function was evaluated using both the ”None” activation function (i.e., no activation function) and the ”relu” activation function, but no significant difference is observed. As a result of the testing, it was determined that using either no activation function or the ”relu” activation function resulted in similar AUC scores.

The number of nodes per layer also exhibits an interesting trend, as illustrated in ”Figure 14” and ”Figure 15”. The gray lines in these figures indicate the optimized number of nodes for each dataset, while the black line represents the mean of all optimized results. This trend of fluctuating node numbers per layer, where fewer nodes are initially used, followed by a higher number of nodes, is noteworthy. This is particularly significant for deep neural networks, as hidden layer nodes must be sufficiently numerous to capture multiple features and learn complex patterns.

These unexpected results from the hyperparameter tuning process required further investigation in future studies, in order to obtain a more in-depth understanding of the trend.

4.3. Local Outlier Factor

4.3.1. Algorithm

[SG] The existing algorithms detect an object as an outlier or not an outlier. However, this algorithm is inspired by considering the non-binary property of an outlier. The local outlier factor (LOF) (Breunig et al., 2000) is the degree to which an object is called an outlier, and it searches for local outliers rather than global outliers.

According to the Hawkins-Outlier definition (Hawkins, 1980), an object is an outlier that deviates from other observations and arouses doubt that it might come from another mechanism. On the other hand, in the Distance-based outlier detection method (DB(pct, d_{min})) (Knorr & Ng, 1998), an object p is called an outlier if at least the percentage (pct) of the objects in D lies greater than the distance (d_{min}) from object p. ”Figure 3” shows a two-dimensional dataset with two clusters: sparse cluster C1 and dense cluster C2. There are 400 objects in cluster C1 and 100 objects in cluster C2. In addition, there are two more objects: o1 and o2. According to Hawkins-Outlier, both o1 and o2 are outliers. However, according to distance-based outliers, only o1 is an outlier; because if o2 is an outlier, then it might be the case that some of the objects in C1 are also outliers. There are no global parameters for which o1 and o2 are separated from C1 and C2.

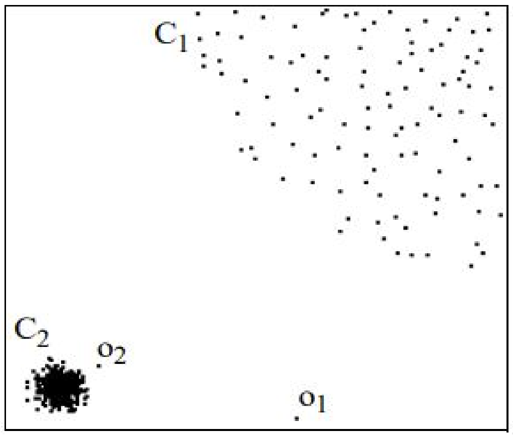


Figure 3. Two-dimensional dataset (Breunig et al., 2000).

In this algorithm, an object's local outlier factor focuses on the object's k-nearest neighbor distance relative to the k-nearest neighbor distances of these k neighbors of the object. LOF considers the surrounding neighborhood density and can be defined as follows :

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|} \quad (4)$$

where, p and o are the points in dataset (D), lrd is the local reachability density, and $N_k(p)$ is the k-distance of p.

The value of LOF is one if the point is deep inside the dense cluster, and the value of LOF is more than one if it is an outlier. A few hyperparameters affect the value of LOF, and one of the most important hyperparameters is MinPts. Let us consider "Figure 16" in Appendix; there are three clusters in the left subfigure. S1, S2, and S3 have 10, 35, and 500 objects, respectively. In the right subfigure, the x-axis represents Minpts, and the y-axis represents the LOF value. For MinPts = 10 to 50, objects in S3 are not outliers because all minimum points are within the cluster. Therefore, its LOF value is always close to one. In S2, the LOF value is close to one until MinPts = 45; after that, its LOF value is high. It is because S2 starts to include points from S1 from MinPts = 36, and at MinPts = 45, it starts to include points from S3. Finally, in the case of S1, it has outliers for MinPts = 10 to 35. It is observable that MinPts have a significant influence on LOF value.

4.3.2. Experiment settings and results

There were 40 train datasets and 10 test datasets in this case study. Firstly, the hyperparameter tuning was conducted on 40 train datasets. For the optimal hyperparameter, the mean value was chosen for the continuous hyperparameter. Consequently, the testing was executed on 10 test datasets with the optimal tuned hyperparameter. The evaluation metric was the AUC score. "Table 3" in Appendix summarizes the default and optimal hyperparameter values with AUC scores. In the LOF algorithm, the "n_neighbors" (i.e. MinPts) hyperparameter was tuned. The default value was 20, and the optimal tuned value was 48. As a result, the AUC score was increased by 6%.

Interestingly, it has also been observed that the LOF algorithm performs well in high-dimensional datasets. For instance, the "fashion0.npz" and "har.npz" datasets have 784 and 561 dimensions respectively. The AUC score was 0.88 and 0.80 for the standard hyperparameter; whereas 0.89 and 0.72 were for the best hyperparameters. However, any conclusion can only be established after further research; because the distribution and quality of the dataset play essential roles in anomaly-detecting algorithms.

4.4. Normalizing Flows

4.4.1. Algorithm

[SG] The normalizing flows (Dinh et al., 2014) is a generative-based modeling algorithm, and it models the distribution of the data. It starts with a simple probability distribution and transforms it into a complex probability distribution through a composition of functions, as shown in "Figure 4".

$$z \sim p_\theta(z) = \mathcal{N}(z; 0, I) \quad (5)$$

$$x = f_\theta(z) = f_k \circ \ldots \circ f_2 \circ f_1(z) \quad (6)$$

where, z is a simple random variable with a simple gaussian distribution, x is a complex function, and for any x there is only one z.

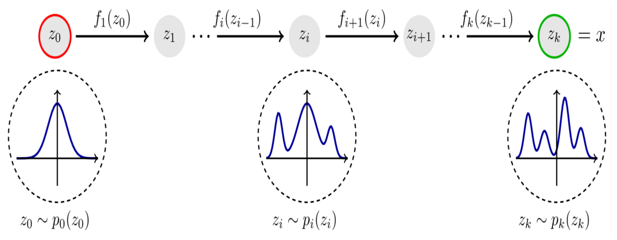


Figure 4. Transforming a simple distribution to a complex one (nf, 2023).

Each function shall be invertible in the sequence. The probability density property states, $\int p(x)\, dx = 1$. The log loss is used while fitting the normalizing flows in the data distribution. The density of x can be calculated by using the change of variable theorem as follows :

$$p_\theta(x) = p_\theta(f^{-1}(x)) \left| det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right| \quad (7)$$

where, $\left| det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$ is the magnitude of the determinant of the Jacobian. This is computationally expensive. The solution approach towards anomaly detection has been suggested in the following ways (Klüttermann) :

- **Translation :** The mean of the gaussian can be changed in this transformation by keeping the unit integral property. It moves the whole distribution to another point.

$$\begin{aligned} 1 = \int p(x)\, dx &= \int p(x+\alpha)\, dx \\ &= \int p(x+\alpha)\, d(x+\alpha) \end{aligned} \quad (8)$$

- **Scaling :** The standard deviation of the gaussian can be changed in this transformation. It rescales the width while adjusting the height of the gaussian to keep the unit integral property.

$$1 = \int p(x)\,dx = \int p(\beta \cdot x)\,dx = \frac{1}{\beta}\int p(\beta \cdot x)\,d(\beta \cdot x) \tag{9}$$

- **Splitting :** Single distribution can be transformed into multiple distributions by keeping the unit integral property. Their location and scale can be changed independently.

$$1 = \int p(x)\,dx = \gamma \cdot \int p(x)\,dx + (1-\gamma)\cdot \int p(x)\,dx, \quad \forall\, 0 \le \gamma \le 1 \tag{10}$$

These three transformations are sufficient to model any distribution; however, many infinite splits are not a practical approach. Therefore, three more extensions have been suggested as follows (Klüttermann) :

- **Mixture Layers :** This transformation rotates the entire output space instead of correlating gaussians.

$$1 = \int p(x)\,dx^d = \int p(A \cdot x)\frac{1}{|A|}\,dx^d \tag{11}$$

- **Non-linearity :** The mixture of translation and scaling transformations is present in the non-linearity transformation.

$$f(x) = N(A \times x + b) \tag{12}$$

- **Realisation :** There are three realisation functions present in this algorithm: gauss, biased, and box. The ”gauss” is an intuitive option for a realisation function; however, when the data distribution is skewed, then the ”biased” realisation function is utilized. Sometimes, the data is distributed with sharp edges and vertices; a ”box” realisation function is a better choice in this scenario.

4.4.2. Experiment settings and results

This case study has been conducted on 40 train datasets and 10 test datasets. Firstly, the hyperparameter tuning was conducted on 40 train datasets. For optimal hyperparameters, the mean value was chosen for the continuous hyperparameters; whereas the mode value was selected for categorical hyperparameters. Consequently, the testing was executed on 10 test datasets with optimally tuned hyperparameters. The evaluation metric was the AUC score. ”Table 4” in Appendix summarizes the default and optimal hyperparameter values with AUC scores. In normalizing flow algorithms, six hyperparameters were tuned including batch size, number of splits, epochs, realisation, mixture, and non linear. There is no significant difference observed in the ”number of splits” hyperparameter after tuning. The ”mixture” hyperparameter also remains the same. However, the AUC score was increased by 3% with the tuned optimal hyperparameters.

Interesting behavior was noticed in high-dimension data. Let us consider ”Figure 17” and ”Figure 18” in the Appendix for both train and test (default and best hyperparameters) datasets. The x-axis represents the number of dimensions, and the y-axis represents the AUC scores. It was observed that as the number of dimensions increases, the AUC score is almost always 0.5 for both train and test (default and best hyperparameters) datasets. The AUC score = 0.5 suggests that the NF algorithm randomly detects the anomaly in high dimensions. This surprising behavior requires further in-depth analyses with different quality and distributions of datasets.

4.5. K - Nearest Neighbors

4.5.1. Algorithm

[UI]The K-Nearest Neighbors (KNN) algorithm is a popular machine learning algorithm used for classification and regression problems. However, KNN algorithm can also be used for anomaly detection, which is the process of identifying abnormal data points in a dataset. Anomaly detection is an important problem in many domains such as fraud detection, network intrusion detection, medical diagnosis, and sensor monitoring(Chandola et al., 2009).

The key idea of using KNN for anomaly detection can be defined as - ’The anomaly score of a certain observation in a dataset can be given by its distance to the k^{th} nearest neighbor in the same dataset. Various distance metrics can be used to find the distance between two data observations, including but not limited to Euclidean, Manhattan and Mahalonobis distances. The key parameters in training and using KNN are the k and the distance metric used to assign the distances between the points. Generally, a thresholding process needs to be applied to the test score before a test instance can be declared anomalous(Chandola et al., 2009).

4.5.2. Experiment Settings and Result

40 distinct datasets were used to tune the hyperparameters using the Flaml library. The set of hyperparameters with the best AUC for that training dataset was stored as the best set for the dataset. The hyperparamaters chosen to tune us-

ing this method for KNN were $n_neighbors$, which depicts the number of neighbors considered for checking the distance and making a cluster and $method$. Arithmetic mean of the best $n_neighbors$ hyperparameter generated from the 40 training datasets was used to generate the overall best $n_neighbor$. The same was done for $method$, bit instead of arithmetic mean, the mode of the $method$ was used tp generate the best hyperparameter as $method$ is a categorical hyperparameter. 10 test datasets were used to check if the optimized parameters improved the performance of the algorithm. In order to make this comparison, the same algorithm was executed on the test datasets using default hyperparameters and their AUCs were stored.

The default parameters for the KNN algorithm were 1 for the $n - neighbors$ and $mean$ for the $method$ hyperparameter. The optimited parameters for the KNN algorithm were 3 for the $n - neighbors$ and $median$ for the $method$ hyperparameter. The optimized parameters improved the AUC scores in some cases but in other cases, the performance was unchanged or degraded. The algorithm struggles in high-dimensional spaces as the datapoints become spread out in a sparse manner which makes distance based identification of outliers somewhat difficult.

4.6. RandNet

4.6.1. Algorithm

[UI]Autoencoders are a type of neural network that learns to compress and decompress data through a process called encoding and decoding. The goal of an autoencoder is to learn a compact representation of the input data that captures its most important features, while still being able to reconstruct the original data with minimal loss of information. The nodes in the hidden layers are smaller in number than the inputs. This leaves the network with no choice but to pick up the weights of each of the inputs so that the middle layers and their outputs depict reduced representations of the input. This creates a scenario where outliers are much harder to reconstruct than the normal data points and this makes autoencoders a good way to identify and classify outliers as such.

In spite of their many powers and advantages, autoencoders have not seen a lot of publicized success in the outlier detection space despite other similar dimensionality reduction methods becoming pretty successful at the same. One of the reasons for this is that the outliers in a certain dataset are included in the training process. This makes the autoencoders vulnerable to over-fitting(Chen et al., 2017).

The authors propose an ensemble methodology to overcome this issue. Ensemble learning involves combining results from various base components to create a more robust model. Predictive algorithms possess a lot of natural variance in their outputs. This variance is dependent on the observations in the dataset or model chosen. In order to capture all of the variance possible in the training scenario, one can run the model multiple times and take a central estimator of the consequent results to obtain much better performance of the model. While, in theory this idea does seem to work pretty well, there is not guarantee that it will work in practice as well. There is a possibility that the best detector amongst the various ensemble components might perform better than the averaging the results the from all the components. One way to avoid this is to make sure that the various components take into account all of the diverse parts of the dataset involved in the training process. This increases the chance that the aggregated model will be better than the individual components (Chen et al., 2017).

The algorithm RandNet, combines ensemble learning along with autoencoders. But, combining a bunch of fully connected autoencoders will not achieve the required diversity amongst the individual components. This is because when the same network structure is used - the results obtained from them will be somewhat similar. This is will be helpful from the point of view of variance reduction. In order to combat this, some connections inside the neural network are dropped randomly. This introduces some diversity in the structures of the individual autoencoders and helps increase the overall performance of the algorithm. The figures below show the difference between and fully connected autoencoder and an autoencoder with its connections dropped randomly(Chen et al., 2017).

The structure of the autoencoder is symmetric with equal number of input and output nodes, which is equal to the number of the dimensions of the data. The number of nodes in the layers following the input layer is set to an ratio of α to the previous layer. The structure is mirrored after the bottle neck layer of the neural net to maintain symmetry. The minimum limit for the number of nodes is set to 3 so that excessive compression of the data will be avoided.

Activation functions are used to add non-linearity to the linear computation carried out by the nodes in any hidden layer. In the hidden layer immediately after the input layer, sigmoid function is chosen - which is given by,

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

. For all other layers, the ReLU function is used as an activation function, given by

$$f(x) = max(0, x)$$

. Balancing the usage of these two activation functions negates the disadvantages of both of them. The presence of two sigmoid function layers at the ends of the network prevents a large gradient from flowing through the ReLU

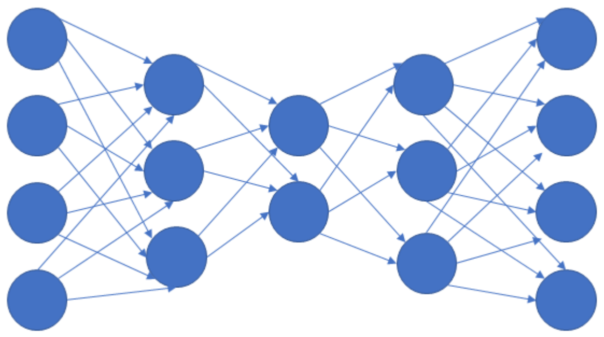


Figure 5. Fully Connected Autoencoder.

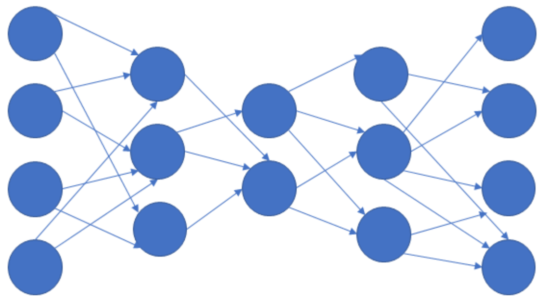


Figure 6. RandNet Autoencoder with randomly dropped connections.

units in the middle. This helps is solving or mitigating the 'Dying ReLU' problem and the ReLU units in the center help prevent the 'Vanishing Gradient' problem (Chen et al., 2017).

In order to drop random connections in the neural network, the following steps are executed for each pair of layers. Suppose there are two layers with l_1 and l_2 number of nodes respectively. Between them, $l_1.l_2$ number of connections are possible. From these possible connections, $l_1.l_2$ connections are sampled with replacement. This means there will be certain connections which will be included twice in the sample and certain connections will be that will be omitted in the sample. The latter of these categories will be the connections that will be dropped in the neural network(Chen et al., 2017).

The ensemble learning part of the happens as follows. Assume that there are m ensemble components, with training set of n data points and d dimensions. We denote the $i-th$ ensemble component's j_{th} record using $x_{ij} \in R^d$. The $outlierscorevector$ for the i_{th} component is generated by

$$[OS_i]_j = \Sigma_{k=1}^{d}([x_{ij}]_k - [o_{ij}]_k)^2 \in 1...m, j \in 1...n.$$

This outlier score vector is then normalized to a standard deviation of one unit. The final outlier score vector is calculated by computing the median of different ensemble components.

For optimizing the error, the RMSProp method is used. It involves maintaining a running average of the magnitudes of the recent gradients. This average is then used to normalize the new gradient generated in the next step of the back-propagation. The running average $(RA)_t$ is given by,

$$RA(t) = \rho.RA(t-1) + (1-\rho).g_t^2.$$

Here, g_t^2 is the gradient computed at time t. The parameter ρ is used to control the rate of adjustment. For the purposes of this scenario, it was chosen to be 0.9. The gradient update is given by

$$\theta_{t+1} = \theta_t - \frac{\nu}{\sqrt{RA(t)+\epsilon}}.$$

The ν is a static learning rate and ϵ is used to avoid division by zero and such ill-conditioned problems in the update process(Chen et al., 2017).

Layer-wise pre-training is used to speed up the training process. This involves training the weights of two symmetrically selected layers along with a bottle-neck middle layer to learn the reduced representation in a three layer network. This representation is then used to train the next level of hierarchical reduced representation in another three layered neural network. $\frac{1}{3}^{rd}$ of the training sample is used to perform this layer wise pre-training for each hierarchical ensemble component (Chen et al., 2017).

4.6.2. Experiment Setting and Results

The same setup for tuning as in in KNN. There were 40 datasets on which tuning of the hyperparameters occured.The optimized parameters from these tuning results were then applied to a distinct set of 10 'test-datasets'. The default set of hyperparameters was used on the test-dataset to calculate how much improvement the tuned parameters brought about, if any. The hyperparameters selected to tune were the learning rate lr and α which is used to determine the ratio at which the number of nodes will be created in the layers following the input layer and the layers preceding the output layer. The modes of both the hyperparameters as found in the 40 datasets was used as the optimized versions of themselves on the test datasets. The default value of the learning rate and α was 0.01 and 0.5 respectively. The optimized values were 0.6 and 0.03 respectively.

The optimized parameters usually produced an improvement in the AUC scores on the test datasets. This improvement might also be because of and increase in the α parameter of the algorithm. An increase in α means an increase in the number of nodes in the hidden layers of the autoencoder. This amounts to an increased model capacity which may account for the increase in AUC score. Increasing α too much will make the model prone to overfitting. 0.6 seems like a good balance between avoiding overfitting and increasing the performance of the model.

4.7. LODA

4.7.1. Algorithm

[SD] Loda is abbreviated as a Lightweight Online Detector of Anomalies. It is a fast unsupervised machine-learning algorithm for anomaly detection with less space complexity. Loda is an ensemble-based algorithm designed with a collection of weak detectors to achieve a higher anomaly detection rate. In domains like real-time continuous data from sensors or online streaming data, Loda can detect oddly behaving information from a stream of large data samples. Despite missing variables in input data, Loda can efficiently identify the anomalies within the data stream. Applications of the algorithm are widely classified in the domains like requests from fraudulent network bots, malware-infected computers in the network, misclassified sensory information and many more.

Loda mainly detects the anomalies using the joint probability of the data-generating process. The process involves density estimation as a training phase and is followed by the classification of anomalies. Input data is processed into a set of one-dimensional histograms and later each set of histograms is described as projection vectors. The collection of all the one-dimensional histograms is ensembled into a strong classification method for anomaly detection (Aggarwal, 2013).

Loda is operated as a two-parameter algorithm which includes the number of one-dimensional histograms and a number of bins. These hyperparameters jointly describe the performance of the algorithm for certain input data resulting in finding anomalies. Additionally, Loda can also be solely independent of hyperparameters values and dependent on the data-generating process by defining default values for hyperparameters. Despite of default anomaly detection, the algorithm can be further optimized by efficient hyperparameter tuning based on evaluation metrics.

Let $(h_i)_{i=1}^{k}$ be the k one-dimensional histograms and $\{w_i \in \mathbb{R}^d\}_{i=1}^{k}$ be the single projection vector projected by the approximate probability density of each histogram.

Let $f(x)$ be the Loda function on sample input x. The output is represented as the average of the logarithm of probabilities estimated on individual projection vectors.

$$f(x) = \frac{-1}{k} \sum_{i=1}^{k} \log \hat{p}(x^T w_i) \tag{13}$$

Where $\hat{p}$ is the probability estimated by i^{th} histogram and w_i be the corresponding projection vector.

In the training phase, the n data sample $\{x_i \in \mathbb{R}^d\}_{i=1}^{n}$ with initializing set of sparse d dimensional $(d^{-\frac{1}{2}})$ non zero random projection vectors. $\{w_i\}_{i=1}^{k}$ return the updated set of histograms $\{h_i\}$ and corresponding projection vectors $\{w_i\}_{i=1}^{k}$ upon sample training data. This process also ensures the missing variables in data by initializing spare projections.

The classification phase proceeds with a set of histograms $\{h_i\}$ and projection vectors $\{w_i\}_{i=1}^{k}$ from the training phase and returns the probabilities of whose logarithms are averaged by the function 13.

An individual score is generated by ranking the sample's anomalousness by identifying the sparse projection of each histogram by randomly generated sub-spaces. The score is ranked high when the anomaly feature is present when compared to no feature identified (Rondina et al., 2014).

4.7.2. Experiment settings and results

The Loda can be performed with default parameters with the number of bins in histograms as 10 and the number of random cuts as 100 (Zhao et al., 2019). The process of tuning involves starting with 40 data sets with default parameters and followed by tuned parameters. The evaluation metric as AUC score is generated upon each dataset. The overall performance is compared against default and tuned parameters. Figure 19 and 20 illustrate frequency distribution after tuning. The average number of bins and the number of random cuts resulted in 569 and 1311 respectively.

Algorithm performance is evaluated based on 10 test datasets upon default parameters, mean parameters and decision tree suggested parameters. The table 5 gives us the overall results with AUC scores. We observe *wbc* dataset AUC score for mean and suggested parameters with an improvement of 0.9 to 1.0 upon default parameters indicating all correct predictions. An overall increase in AUC is observed in all datasets upon mean parameter values except for the *breastw* dataset. The AUC score is reduced from 0.97 to 0.20. The reason for the difference is not clear and also explains considering mean as an optimal tuning approach or not for the Loda algorithm.

A similar trend of results was observed in the comparison of the decision tree suggested parameters and default parameters. In figure 21, we observe an overall increase up to 15 percent over suggested parameters. Performance is reduced for *breatw* and *Liver_1* datasets but slightly better compared to the mean of parameters value.

4.8. Deep SVDD

4.8.1. Algorithm

[SD] Deep Support Vector Data Description (Deep SVDD) is a deep learning-based neural network algorithm for anomaly detection. It is a combination of one-class classification and Support Vector Data Description. A neural network is trained while generating a hypersphere to classify the input data from anomalies. The main objective of this algorithm is to minimize the volume of the hypersphere with the sphere separating normal data and common factors of variation. The primary application of Deep SVDD are detecting adversarial attacks, financial transactions and identifying defective products in manufacturing. This algorithm can be applied in domains where data is unsupervised and labels are limited. (Ruff et al., 2018b).

In this approach, the neural network takes the input as high dimensional data space $\mathcal{X} \subseteq \mathbb{R}$ is transformed with an activation function $\phi(;\mathcal{W})$ with weights $\mathcal{W}$ to an output space $\mathcal{F} \subseteq \mathbb{R}^n$.

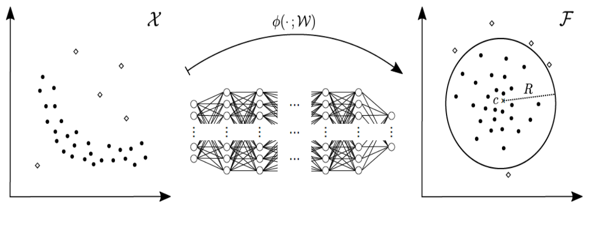


Figure 7. DeepSVDD workflow from input data transformation to minimizing the volume of hypersphere (Ruff et al., 2018b).

The hypersphere is represented by a centre $\boldsymbol{c}$ and with radius $\mathcal{R}$ such that volume of the hypersphere is minimized to separate the normal flow of data point mapping inside the sphere and anomalies mapped outside the sphere.

The objective of DeepSVDD is defined as

$$\min_{\mathcal{W}} \frac{1}{n} \sum_{i=1}^{n} \|\phi(\boldsymbol{x}_i;\mathcal{W}) - \boldsymbol{c}\|^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} \left\|\boldsymbol{W}^{\ell}\right\|_F^2 \tag{14}$$

where some input space $\mathcal{X} \subseteq \mathbb{R}^n$ and output space $\mathcal{F} \subseteq \mathbb{R}^p$ and $\phi(;\mathcal{W}) : \mathcal{X} \to \mathcal{F}$ neural network function with set of weights $\mathcal{W} = (\mathcal{W}^1, ..., \mathcal{W}^l)$ and $\mathcal{W}^l$ are the weight of layers $l \in (1, ..., L)$. Deep SVDD makes use of quadratic loss for penalizing the distance of every network representation $\phi(\boldsymbol{x}_i;\mathcal{W})$ to $\boldsymbol{c} \in \mathcal{F}$. The network weight decay regularization on the network parameter $\boldsymbol{W}$ with hyperparameter $\lambda>0$ and $\|\cdot\|$ denoted the Frobenius norm.

The anomaly score $s()$ is defined for an input test data point $\boldsymbol{x} \in \mathcal{X}$ by calculating the distance between the centre of hypersphere $\boldsymbol{c}$ to the point.

$$s(\boldsymbol{x}) = \|\phi(\boldsymbol{x};\mathcal{W}^*) - \boldsymbol{c}\|^2 \tag{15}$$

(Ruff et al., 2018b)

4.8.2. EXPERIMENT SETTINGS AND RESULTS

DeepSVDD implementation takes the following parameters which include $\boldsymbol{c}$ as the centre which is default calculated based on network initialization's first forward pass. *use_ae* is auto encoder type of DeepSVDD which reverses neurons from hidden neurons and by default set it as 'True'. *hidden_neurons* by default as [64, 32] are the number of neurons per hidden layer. *hidden_activation* an activation function to use for hidden layers. All hidden layers are forced to use the same type of activation which default takes as 'relu'. *output_activation* a activation function to use for output layer (default='sigmoid'). *optimizer* is the name of the optimizer and in most of the deep learning models 'adam' is considered by default. *epochs* are number of iterations to train the model and defaults to 100. The number of samples per gradient is updated as *batch_size* which is default considered as 32. *dropout_rate* (default= 0.2) is to be used across all layers. *l2_regularizer* (default=0.1) is the strength of the activity regularizer applied on each layer. The percentage of data to be used for validation *validation_size* which is default equals 0.1. To standardize data, *preprocessing* is considered by default. *random_state* is the seed used by the random number generator and by default set to 'None' (Zhao et al., 2019).

In our current analysis over 40 training datasets are considered for the process of tuning hyperparameters. Choice of "relu", "sigmoid", and "tanh" are considered for *hidden_activation* and *output_activation*, *no_of_epochs* to be ranging from 1 to 1000 and *dropout_rate*, *l2_regularizer* and *validation_size* from 0 to 1. *preprocessing* is said to be considered "True" or "False". Due to the complexity of execution and resources, rest of the parameters are considered to be default in the analysis.

During the tuning process, it is observed that "sigmoid" is an ideal activation function for *hidden_activation* and "relu" for *output_activation*. The average number of epochs is observed to be 241, similarly for *dropout_rate* is 0.16, *l2_regularizer* is 0.588, *validation_size* is 0.66. In most of the training datasets, enabling the *preprocessing* leads to better results.

Tuned hyperparameters are considered to be optimal for datasets based on the features and samples for the decision tree algorithm. During the testing process, default parameters AUC scores are compared against the tunned parameters AUC scores which are ideally suggested by the decision tree algorithm.

In figure 22 we observe DeepSVDD performance for most of the datasets is increased significantly for tuned parameters over the default parameters. Datasets *wbc* and *liver_1* exhibit less AUC scores compared to default parameters AUC scores. The reason for this drop is expected to be both these datasets have less number of samples and are highly dimensional. Additionally, *cover* dataset which has a huge number of samples performes very efficiently over tuned hyperparameters. From this, we interpret that tuning hyperparameters for DeepSVDD can show optimal results when the dataset has a good ratio of a sample size to its features.

4.9. Principal Component Analysis

4.9.1. Algorithm

[KAH] Principal component analysis is an unsupervised approach that reduces the dimensionality of a dataset where a large number of variables are interrelated, while preserving the maximum variation present in the original dataset. High dimensional dataset is transformed into a set of new variables named as the principal components. The principal components are linear functions of the original variables and capture most of the variance contained in the original variables. These components are uncorrelated and ranked according to the amount of variation explained by them. Hence, the largest amount of variation is captured by the first principal component, the second largest variance by the second principal component, and so on (Mohammed J. Zaki, 2020).

As the aim is to find the best r-dimensional approximation to the dataset, the input data $\mathbf{D} \in \mathbb{R}^{n \times d}$ is centered initially by subtracting the mean μ.

$$\overline{\mathbf{D}} = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}^T \tag{16}$$

Eigenvectors and eigenvalues are then computed from covariance matrix $\mathbf{\Sigma}$ which is expressed as:

$$\mathbf{\Sigma} = \frac{1}{n}\left(\bar{\mathbf{D}}^T \bar{\mathbf{D}}\right) \tag{17}$$

(Mohammed J. Zaki, 2020).

The eigenvalues $(\lambda_1, \lambda_2, \ldots, \lambda_d)$ as well as the eigenvectors $(u_1, u_2, \ldots, u_d)$ are computed from the covariance matrix $\mathbf{\Sigma}$ using eigendecomposition method which satisfies $\mathbf{\Sigma}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = 1, 2, \ldots, d$.

The largest directions of variation in the data are defined as eigenvectors, and the amount of variance explained by each eigenvector is called an eigenvalue. The eigenvalues with a very small contribution to variance are discarded in PCA. Therefore, after obtaining the eigenvalues $(\lambda_1, \lambda_2, \ldots, \lambda_d)$ as well as the eigenvectors $(u_1, u_2, \ldots, u_d)$, the smallest set of dimensions r is chosen which can explain at least a desired variance threshold α of the total variance

$$f(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq \alpha \tag{18}$$

where $f(r)$ is fraction of variance for all $r = 1, 2, \ldots, d$ (Mohammed J. Zaki, 2020). Generally, α is considered as 0.9 or more to make sure at least 90% fraction of variance is captured by the reduced dataset. Eventually, the coordinates of the data points are computed for the new data matrix $\mathbf{A} \in \mathbb{R}^{n \times r}$ in the r-dimensional subspace of principal components.

Furthermore, PCA is very sensitive to anomalies. A large influence on the first few principal components as well as on variance can be observed when there exist some anomalies in the dataset. One approach to detect the anomalies using PCA is to evaluate the distance between the center of the principal component space and each data point where data points with a certain standard deviation distant from the center are regarded as anomalies. Figure 8 depicts the influence of an outlier on the first principal direction where the green arrow represents the first principal direction, the red square shows an anomaly, and the clustered blue circles indicate the normal data.

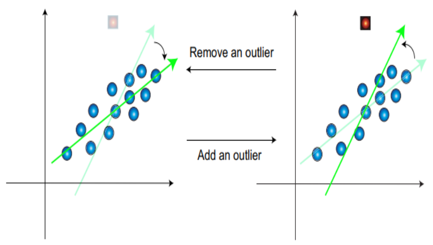


Figure 8. Illustration of the impact of an anomaly on first principal direction (Yi-Ren Yeh & Lee, 2009).

It is noticeable that, when any deviated instance or outlier is added, the first principal direction gets affected and makes a bigger angle, creating a larger impact on the principal direction. Similarly, removing the outlier or adding any normal point ends up with a much smaller angle (Yi-Ren Yeh & Lee, 2009). The extent of variance represented by the principal direction for each new instance determines whether it is an anomaly or not.

4.9.2. Experiment Settings and Results

For PCA algorithm, the number of principal components ($n\_components$) was the only hyperparameter considered for the tuning process. If the number of components is too few, some of the variances of the original dataset are lost. On the other hand, too many components cause the model to overfit the data. Therefore, it is necessary to tune this hyperparameter while considering a trade-off between the dimensionality reduction and capturing as much meaningful information as possible. The hyperparameter ($n\_components$) was tuned on 40 training datasets and the mode of the optimal hyperparameters was chosen as the suggested parameter for 10 test datasets.

Figure 29 portrays that the dimension of the original dataset has a huge influence on the corresponding optimal number of components. It is observed that, the optimal number of components is high for high dimensional datasets. For instance, optimal $n\_components$ is found 1060 for the *bioresponse.npz* dataset with shape $(1424, 1776)$, 99 $n\_components$ for *musk.npz* with shape $(2868, 166)$, 199 $n\_components$ for *speech.npz* with shape $(3564, 400)$ etc. However, it is also noticeable that the dimensionality of the dataset is not the only factor of the optimal number of components as the line of optimal $n\_components$ in the figure 29 does not always follow the peaks of the number of features line especially in *gas-drift.npz* dataset with shape $(1796, 128)$, *hill-valley.npz* with $(425, 100)$ which have optimal $n\_components$ 2 and 3 respectively. Some other factors affecting this hyperparameter may include the underlying patterns of the dataset, the amount of noise present in the data, etc. Table 9 demonstrates the AUC scores of PCA using the default parameter and best-suggested parameter, implying that using the mode of tuned hyperparameter does not ensure any significant improvement.

4.10. Autoencoder (Over)

4.10.1. Algorithm

[KAH] Unlike PCA, autoencoders deal with non-linear representations, allowing them to capture more complex relationships. An autoencoder is composed of two functions which are called encoder and decoder. The encoder function maps the input into a space with smaller dimensionality in general compared to the original space, also known as latency space. Subsequently, the decoder function needs to symmetrically map the encoder back to the initial input without any information being lost. In other words, considering an encoding function $E(x)$ and a decoding function $D(x)$, $D(E(x)$ needs to return to the input x. Hence, the optimization problem (Roberto Aurelia & Brociekb, 2021) can be written as follows:

$$\min_{\theta_D, \theta_E} \|x - D(E(x)\|^2 \tag{19}$$

Here, θ_D, θ_E represent the parameters of the $D(x)$ and $E(x)$ respectively. The parameters are learned while performing the optimization in order to minimize the reconstruction error $\|x - D(E(x)\|^2$. The reconstruction error, estimated by the difference between the input data and reconstructed output can be utilized to detect anomalies. The input data with reconstruction error above a certain threshold can be classified as an anomaly. The threshold value may be selected depending on the distribution of reconstruction errors on the training dataset (Roberto Aurelia & Brociekb, 2021). Contrary to conventional autoencoder, autoencoder (over) has a latency size larger than the number of features.

4.10.2. Experiment Settings and Results

In our case study, two types of autoencoder were compared which are autoencoder (under) and autoencoder (over). Both differ in latency size where autoencoder (under) has a latency size less than the number of features and autoencoder (over) has a fixed latency size which is 10 times the number of features. For both of them, the remaining hyperparameters were optimizers including '$adam$', 'sgd','$rmsprop$', '$adagrad$', '$adadelta$', loss functions including '$binary\_crossentropy$', '$mean\_squared\_error$', '$mean\_absolute\_error$', number of epochs, learning rate and batch size. Moreover, Rectified Linear Unit (ReLU) activation function was implemented in both encoded and decoded layers. In addition, sigmoid function was used in the output layer in both autoencoders. After tuning the hyperparameters on 40 training datasets, mode of each hyperparameter was chosen as the suggested hyperparameters for 10 test datasets.

Having compared these two types of autoencoders, figure 30 depicts that autoencoder (under) performs better with higher AUC scores when it comes to best found parameters on 40 training datasets. One potential reason is that the latency size of 10 times the number of features, simply copies the original input and pastes it as output while ignoring the less important or non-representative features in the input, causing autoencoder (over) to perform poorly in anomaly detection. On the other hand, Autoencoder (over) performs better with 4.61% higher AUC scores overall using the suggested parameters on 10 test datasets as shown in Table 10. Due to a fixed latency size, fewer number of parameters are tuned in this case where autoencoder (under) deals with the tuning of latency size as well. Tuning less number of hyperparameters makes the optimization process more efficient, implying that the model is less likely to overfit, and may perform better in generalization to new data. The results will be discussed in detail in the comparison part of the report.

4.11. LUNAR : Learnable Unified Neighbor-based Anomaly ranking

4.11.1. Algorithm

[KK] Many local outlier methods, such as the Local Outlier Factor (LOF), K-Nearest Neighbor (KNN) and DBSCAN, are commonly utilized for anomaly detection. These algorithms have pre-defined parameters such as nearest neighbors (k), distance (ϵ), and number of points ($minPts$). These approaches are also a subset of Graph Neural Networks' (GNNs) message passing system (message, aggregation, and updation). The GNN is a graph-based representation of neural networks (Goodge et al., 2022) (Hamilton, 2020).

All of these algorithms, however, lack learnability, which means they do not modify the parameters based on the training data in order to attain higher anomaly scores. LUNAR, a graph neural network-based anomaly detection technique, is suggested to address the problem of learnability in the message passing framework. It is a GNN with a single layer and a message-passing mechanism. The target node is linked to its source nodes (nearest neighbors), and information or messages are transmitted between them via the connecting edges. LUNAR is designed as follows (Goodge et al., 2022):

Graph Formulation : Let there be a data sample x_i with x_j as the k-nearest neighbor. So the edge (j,i) connecting target node i with source node j is defined as $e_{j,i}$, which is an Euclidean distance.

$$e_{j,i} = \begin{cases} dist(x_i, x_j) & \text{if } j \in \mathcal{N}_i \\ 0 & \text{otherwise,} \end{cases}$$

where, $\mathcal{N}_i$ is the set of nearest neighbors nodes. Once the graph is formed, the message passing framework works as follows,

- **Message :** Message passed from j to i along edge (j,i) is equal to $e_{j,i}$ or Euclidean distance between the points.

$$\phi^{(1)} := e_{j,i}.$$

- **Aggregation :** Instead of the optimal method of max-pooling or fixed average, we use learnable aggregation in this approach. A k-dimensional vector, $e^{(i)}$ is formed by concatenating the distance between the k neighbors and the sample x_i.

$$e^{(i)} := [\, e_{1,i}, ..., e_{k,i}] \in \mathcal{R}.$$

  Then, $e^{(i)}$ is mapped with the weights (Θ) to form a neural network, $\mathcal{F}$.

$$h^{(1)}_{\mathcal{N}_i} := \mathcal{F}(e^{(i)}, \Theta)$$

- **Update :** Atlast's update function outputs this learned aggregate message.

$$\gamma^{(1)} := h^{(1)}_{\mathcal{N}_i}$$

The GNN is trained using a loss function to return a score of 0 for normal nodes and 1 for abnormalities. Since all of the training samples are normal samples, the GNN outputs only 0 for training samples hence negative sampling is performed on the training samples to avoid this problem. Negative sampling is classified into three types: uniform, subspace, and mixed. Uniform sampling produces negative samples through uniform distribution with a small, positive constant, ϵ.

$$x^{(negative)} \sim \mathcal{U}(-\epsilon, 1+\epsilon) \in \mathcal{R},$$

where, $x^{(negative)}$ are the negative samples generated. The other approach, subspace sampling, adds Gaussian noise to the normal training samples, $x_i^{(train)}$, with a small, positive constant, ϵ and a vector of binary random variables, $\mathrm{M} \in \mathcal{R}^d$ (Goodge et al., 2022).

$$z \sim \mathcal{N}(0, I) \in \mathcal{R}^d,$$

$$x_i^{(negative)} = x_i^{(train)} + \mathrm{M} \circ \epsilon z.$$

The mixed sampling is the mixture of uniform and subspace sampling.

4.11.2. Experiment settings and results

The algorithm's hyperparameters are tuned to obtain 40 sets of optimal hyperparameters using 40 training datasets. The mode is used to select the best value of the optimal hyperparameters from these sets for two categorical hyperparameters ($model\_type$ and $negative\_sampling$) and the mean for numerical hyperparameters ($n\_neighbors$, $epsilon$, n_epochs, lr and $proportion$). Figure 23 depicts the evaluation of the algorithm on the 10 test datasets using the default settings and best value of the hyperparameters as given in Table 7. We can see in the Figure 23 that there is hardly any change in the AUC scores of the datasets. The AUC scores for the datasets 'har' and 'steel-plates-fault' do not change. The dataset 'Liver' rises from 0.647 to 0.658, while the dataset 'Diabetes_present' rises from 0.736 to 0.744. The datasets 'optdigits' and 'wbc' have obtained AUC scores of 1.0 with the best hyperparameters. Overall, there is a 2.6% improvement in the outcomes of the AUC scores or accuracy.

4.12. Gaussian Mixture Model

4.12.1. Algorithm

The Gaussian Mixture Model (GMM) is a clustering algorithm that is part of the unsupervised machine learning technique (Zhao et al., 2019). It is a distribution-based model that is an extension of the K-means technique. GMM would determine the likelihood of each data point and assign it to a cluster or distribution for the provided data points. Clusters are modeled as Gaussians distributions rather than simply by their means, as in K-means. So not just means also a covariance that describes there ellipsoidal shape. To fit the model, we need to maximize the likelihood of the observe data. This can be obtained by using Expectation-Maximization mechanism (Fortran et al., 1992).

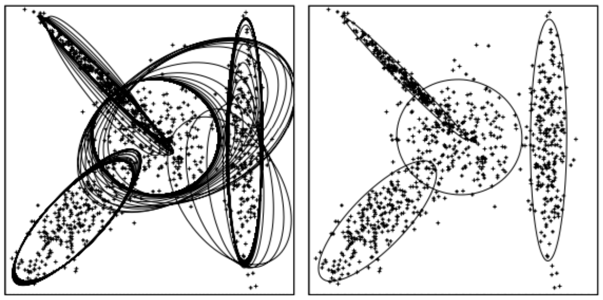


Figure 9. Gaussian Mixture Model in M dimensions, N = 1000 samples and K Gaussian Clusters (Fortran et al., 1992).

The Expectation-Maximization Algorithm work as follows: Assume there are $x_i's$, $i = 1, ..., n$ observations in M dimensional space, K Gaussian-clusters, following a normal distribution, $\mathcal{N}$, with mean as μ_k, covariance matrix as Σ_k and π_k be the mixing probabilities and the collection of all parameters is denoted by, $\Theta := \pi_1, \mu_1, \Sigma_1, ..., \pi_k, \mu_k, \Sigma_k$, where k = 1, ..., K. Each cluster is represented by one component of the mixture distribution, $\phi(x_k; \mu_k, \Sigma_k)$ (Fortran et al., 1992).

- Evaluate the initial log likelihood of the probability of the data x_n given Θ.
- **Expectation (E) :** Estimate the probability of individual observation x_i that it belongs to cluster j using the current parameters of the supplied data and denote it as γ_{ij}.

$$\gamma_{ij} = \frac{\pi_j \phi(x_i; \mu_j, \Sigma_j)}{\sum_{k=1}^{K} \pi_k \phi(x_i; \mu_k, \Sigma_k)}$$

- **Maximization (M) :** Using the estimated (γ_{ij}) probability distribution from the previous step, maximize the function to estimate the model parameters.

$$\mu_j = \frac{\sum_n \gamma_{ij} x_i}{\sum_n \gamma_{ij}}, \quad \Sigma_j = \frac{\sum_n \gamma_{ij} (x_i - \mu_j)^2}{\sum_n \gamma_{ij}}$$

$$\pi_j = \frac{1}{n} \sum_n \gamma_{ij}$$

Essentially, you make initial predictions for the number of gaussian distributions, assign random values for mean, covariance, and density, and then alternate between E and M steps until your estimates converge. The EM algorithm is depicted in Figure 9 where the left part of the figure shows Gaussian models with iterations of E and M steps and the converged result is shown on the right part. (Fortran et al., 1992).

4.12.2. Experiment settings and results

Running the method on 40 training datasets with three optimal hyperparameters, i.e., $n\_components$, $covariance\_type$ and $max\_iterations$ yielded 40 sets of it. The best value of $n\_components$ and $max\_iterations$ is taken as the mean of 40 sets, which is 5 and 38, respectively, and $covariance\_type$ is taken as the model which is full. With the default settings and best values of hyperparameter as given in Table 8, the algorithm is evaluated on the 10 test datasets and plotted as shown in Figure 24. The figure depicts that the datasets 'Liver_1', 'breastw' and 'steal-plates-fault' show no change in their AUC scores. The dataset 'optdigits' shows an increase in AUC scores from 0.782 to 0.965. The 'wbc' dataset yields an AUC score of 1. Though there are some ups and downs within the AUC scores of datasets, as depicted in Figure 24, there has been only 1.1% improvement in the scores overall.

4.13. Isolation-based Nearest Neighbor Ensembles - iNNE

[IK] The iNNE model is an unsupervised anomaly detector that combines the strengths of both isolation and nearest neighbour methods. Unlike tree-based models or density estimators, iNNE isolates instances by building hyperspheres around them. The radius of these hyperspheres is determined by the nearest neighbour distances, making dense regions create smaller hyperspheres than sparse ones. This relativity between instances and their neighbourhoods allows the model to detect both global and local anomalies. The output of the model is an anomaly score that ranges between 0 and 1, where 1 represents the perfect outlier instance and 0 the perfect normal instance. (Bandaragoda et al., 2014)

4.13.1. ALGORITHM

The iNNE algorithm generates a set of t sets of hyperspheres from $(S_i)_{i=1,..,t}$ subsamples of size ψ, selected randomly without replacement from the data space D. The number of estimators t and the subsample size ψ are the only hyperparameters defined by the user. Each generated hypersphere $\{B(c) = \{x : ||x - c||_2 < \tau(c)\}|c \in S_i\}$ with $x \in R^d$, is centered at c and with radius $\tau(c) = ||c - \eta_c||_2$, where η_c is the nearest neighbour of c, making it the largest hypersphere to isolate $c \in S_i$ from the rest of the instances in S_i. (Bandaragoda et al., 2014)

After the generation of hyperspheres, a second stage is implemented to evaluate the anomaly score $I_i(x) \in [0, 1]$ of each test instance $x \in R^d$ based on S_i.

$$I_i(x) = \begin{cases} 1 - \frac{\tau(\eta_{cnn(x)})}{\tau(cnn(x))} & \text{if } x \in \bigcup_{c \in S} B(c) \\ 1, & \text{otherwise} \end{cases} \tag{20}$$

where $cnn(x) = argmin_{c \in S}\{\tau(c) : x \in B(c)\}$

For each instance $x \in R^d$, the set of t anomaly scores is then averaged. (Bandaragoda et al., 2014)

$$\bar{I}(x) = \frac{1}{t}\sum_{i=1}^{t} I_i(x)$$

4.13.2. PERFORMANCE

Because iNNE does not rely on density estimation, it is able to define the boundaries around neighbourhoods using only a few samples. These boundaries are then used to assign the maximal isolation score of 1 to the instances that fall behind all of them, and neighbourhood relative scores to the other instances. The algorithm is also robust against irrelevant dimensions, because it uses all the features of the data to construct the boundaries. (Bandaragoda et al., 2014)

4.13.3. EXPERIMENT SETTINGS AND RESULTS

To test the effect of the sample size in iNNE, we conduct the same experiment on the training and testing data sets. We compare the AUC scores achieved of each model trained with the true (tuned) sample size ψ_{true_i}, with the AUC scores achieved with the standard ($\psi_{standard} = 8$) sample size, as well as the chosen sample sizes: the mean and median of the true sample sizes of the 40 training data sets ($\psi_{mean} = 73$ and $\psi_{median} = 49$). Figure 25 (see appendix) shows that overall, the standard and chosen sample sizes perform almost as good as the true parameters in most data sets. However, we observe a relation between the dimensions of the data sets and the performance of the model across the different settings of the sample size. It's interesting to see that the model trained with the standard sample size outperformed the other models in the small and low dimensional data sets: *Liver_1(86, 6) , pima(300, 8), breastw(266, 9)*. While the models trained with the chosen sample sizes outperformed the standard model in the large and high dimensional data sets: *har(926, 561) , cover(5494, 10), optdigits(300, 62)*. This positive effect of the sample size on the performance of iNNE in large or high dimensional data sets makes the algorithm efficient and more easily auto-tuned. That is because the search range of ψ is now smaller.

4.14. Deep Ensemble Anomaly Detection - DEAN

[IK] The DEAN model is a one-class anomaly detection model that employs ensemble methods for deep neural networks. It is designed to detect anomalies in a data set using only normal instances for training. The model leverages the deep architecture of its sub-model neural networks to capture complex relationships between features. The variability of the predictions of these sub-models is also required, to encourage learning diverse relations. Each DEAN sub-models is initialized differently and trained on a random subset of features, where the size of this subset, referred to as bag, is a hyperparameter of the model. The individual performance of the sub-models is subordinate to the depth and variability of the learned relationships, which are combined to form the DEAN ensemble. Furthermore, the scalability of the sub-models contributes to the robustness of the ensemble's predictions. That is because the algorithm's capacity to generate numerous sub-models rapidly can be attributed to their simple loss function. (Böing et al.)

4.14.1. ALGORITHM

The DEAN algorithm requires each sub-model to search for the most significant relations $g(x)$ between the randomly sampled bag of features within the training set. The sub-model operates under the assumption that these relations are the most constant ones where $g(x) = k.(1 \pm \delta)$ and δ being the lowest. To find these relations, the algorithm first standardizes them into a comparable scale:

$$f(x_{train}) = \frac{g(x_{train})}{k} \approx 1 \tag{21}$$

Then, the algorithm minimizes the loss function of each sub-model f_i to find the lowest δ:

$$l_{DEAN_i} = (\frac{g_i(x_{train})}{k_i} - 1)^2 = \delta^2 \tag{22}$$

When evaluating a new instance x_{test} during testing, a consistent scoring function is employed across all sub-models to determine its anomaly score. This consistency makes the combination of the sub-models into an ensemble possible:

$$Score = |f(x_{test}) - q| \tag{23}$$

where $q = mean(f(x_{train}))$

The DEAN ensemble anomaly score of x_{test} then becomes the average of the sub-model scores. (Böing et al.)

$$F(x_{test}) = \frac{1}{n}\sqrt{\sum_{i=0}^{n} f_i^2(x)} \tag{24}$$

4.14.2. EXPERIMENT SETTINGS AND RESULTS

To assess the impact of varying *bag* sizes on the performance of DEAN, we conducted a comprehensive comparison of the achieved AUC scores across multiple *bag* settings. Our analysis, as presented in Figure 26 (see appendix), revealed that the AUC scores are highly sensitive to the hyperparameter settings, with the DEAN model demonstrating significant variance when the *bag* size is altered. Interestingly, we observed that lower values of *bag* size tend to enhance the overall performance of the model. To determine the optimal *bag* size for a given dataset, we sought to examine the relation between the *bag* size and the number of features p in the dataset D. Our findings in Figure 27 (see appendix) demonstrate that while high-dimensional data require a minimum *bag* size to capture deep relations in the training data, low-dimensional data sets only perform well when the *bag* size is lower than 50% of the number of features. The reason for this can be attributed to the dominance of the random feature bagging selection in the anomaly detection process (Böing et al.). Specifically, if the bag size exceeds 50% of the data dimension, many irrelevant features tend to contribute equally alongside the crucial features. Ultimately, the performance of the anomaly detector decays.

Knowing that DEAN is a one-class anomaly detectors, we sought to examine another relation between the optimal *bag* size and our built similarity measure between the training and testing set of each of the 10 data set. This is because in one-class anomaly detection, particularly in our scenario where half of the test data is anomalous, when the test dataset contains instances that are significantly different from those in the training dataset, the learned patterns may not be applicable to the new data. As a result, these detectors may be more sensitive to the differences between the normal and anomalous instances in the test dataset, leading to enhanced detection performance. In contrast, when the test dataset is similar to the training dataset, the one-class anomaly detector may have already learned the patterns and characteristics of the normal data, and may struggle to differentiate between normal and anomalous instances in the test dataset. This can lead to reduced detection performance, as the detector may fail to identify previously unseen anomalies that differ slightly from the normal patterns. (Böing et al.)

In Figure 26, our observations indicate that the true AUC score for both (*wbc* and *optdigits*) data sets decreased by 29% with a 25% increase of the *bag* over features ratio. Interestingly, we also found that the achieved true AUC score on two data sets with the same ratio (*Liver_1* and *Diabetes_present*) decreased by 13%. We sought then to determine the correlation between a quantified resemblance measure of the training and testing data against the AUC scores obtained by the tuned DEAN model (true). To this end, we conducted feature-wise Kolmogorov-Smirnov tests (adjusted using Bonferroni correction) between training features $(x^{(i)}_{train})_{i=1,\ldots,p}$ and testing features $(x^{(i)}_{test})_{i=1,\ldots,p}$. Subsequently, we determined the resemblance between the sets by computing the ratio of features that were not drawn from significantly difference empirical cumulative distribution functions. This resemblance measure $s(D_{train}, D_{test})$ attains the highest possible score of 1 when the training and testing sets are most similar. ((kol, 2008), (Haynes, 2013))

$$s(D_{train}, D_{test}) = \frac{\sum_{i=1}^{p} \mathbb{1}(x^{(i)}_{train} \overset{d}{=} x^{(i)}_{test})}{p} \tag{25}$$

The results in Figure 28 (see appendix) show the strong and negative correlation ($\rho = 0.89$) between the tuned model's AUC scores and our resemblance measure. We attribute the overall decreasing performance of DEAN across the 10 data sets when the resemblance is increasingly higher than 0.5 to the one-class nature of the model. We analyse the two data sets (*Liver_1* and Diabetes_present) with the same *bag* over features ratio (0.5) and increasing resemblance (12.5%). Our findings indicate a 13% decrease in the achieved AUC scores for both data sets as a result of this change. Our analysis also explored the 29% decrease in the achieved AUC scores for the *wbc* and optdigits data sets. Upon examination, we discovered that the resemblance of both data sets fell below our threshold (0.5), yet the achieved score for optdigits was significantly lower. Furthermore, the resemblance between the two data sets was also quite small (0.05). Based on our findings, we conclude that a bag size larger than half of the number of features, particularly for low-dimensional data sets with a high resemblance score, correlates with a significant decrease in the achieved AUC scores of DEAN. This negative correlation is crucial to consider when searching for the optimal bag hyperparameter for the model, as it helps to narrow down the search range.

4.15. Kernel Density Estimation

4.15.1. Algorithm

[UKD]The first step in KDE is to choose a kernel function, which is a non-negative, symmetric function that integrates to 1 over its support. Common choices of kernel functions include the Gaussian (normal) kernel and the Epanechnikov kernel. The Gaussian kernel is given by:

$$K(u) = \frac{1}{\sqrt{2}} e^{\frac{1}{2}u2}$$

where u is the distance from the observation point. The Epanechnikov kernel is given by:

$$K(u) = \frac{3}{4}(1u^2)$$

for —u— 1, and 0 otherwise.

Estimating the PDF

Given a set of n observations $x_1, x_2, ..., x_n$, the kernel density estimate of the PDF at a point x is given by:

$$\bar{f}(x) = \frac{1}{nh}\Sigma_{i=1}^{n} K(\frac{xxi}{h})$$

where h is the bandwidth parameter, which determines the width of the kernel function, and K is the kernel function.

The bandwidth parameter h controls the trade-off between bias and variance in the estimate. If h is too small, the estimate will have high variance and will be sensitive to noise in the data. If h is too large, the estimate will have low variance but will be biased towards the shape of the kernel function. There are several methods for selecting the bandwidth parameter, including cross-validation, rule-of-thumb methods, and plug-in methods.

4.15.2. Experimental Settings and Results

We performed a hyperparameter tuning experiment for a KDE (Kernel Density Estimation) model using the ROC AUC score as the performance metric. The aim of this experiment was to determine the optimal values for the hyperparameters of the KDE model, including leaf_size, metric, algorithm, and contamination. A configuration space was created to search over using the a dictionary, which included leaf_size as a random integer between 5 and 100, metric as a choice between euclidean, l1, and l2, algorithm as a choice between ball_tree and kd_tree, and contamination as a choice between 0.1, 0.2, 0.3, and 0.4. We conducted the experiment using the tune.run method from the Flaml library with the following parameters: opt as the method to optimize, score (ROC AUC score) as the metric to optimize for, max as the direction of optimization, the params dictionary as the configuration space, 4 CPU cores as the computational resources allocated to each trial, a verbosity level of -1, 10 trials, and a time budget of 60 seconds. The output of the experiment was a dictionary containing the scores and evaluation costs of each trial. The optimal hyperparameters were determined based on the highest ROC AUC score achieved in the experiment. After the initial run, the best parameters were generalised for all the datasets based on mode.

The results were rather surprising as the default parameters yielded better results 7 out of 40 times in the training sets and 9 out of 10 times on the test sets, as shown in figure 33 and figure 34, suggesting that generalised parameter optimisation may not be the best for the datasets provided. Even with data preprocessing, somewhat similar findings were discovered as experiments on only 10 out of 40 training sets yielded better results are preprocessing. For the test sets, the numbers are 0 out of 10. Interestingly, the median AUC for the 40 train datasets is 0.72 with optimised hyperparameters while 0.68 without it. While more data is needed to make a definite conclusion, it appears that KDE runs better on its own, or is at least more accustomed to the datasets trained and tested on here, than with hyperparameter tuning (or even data preprocessing for that matter).

4.16. Autoencoder (Under)

4.16.1. Algorithm

[UKD]An autoencoder is a type of neural network that is used for unsupervised learning. It is primarily used for dimensionality reduction, feature extraction, and data compression. The main idea behind autoencoders is to take a high-dimensional input and compress it into a low-dimensional representation, which is then reconstructed back into the original input space. Autoencoders are made up of two parts: an encoder and a decoder. The encoder takes the high-dimensional input and maps it to a lower-dimensional representation, while the decoder takes the low-dimensional representation and maps it back to the original high-dimensional space. The encoder and decoder are usually implemented as deep neural networks, with multiple layers of neurons that learn to map the input to the output.

The architecture of an autoencoder is defined by the number of layers in the encoder and decoder. The simplest autoencoder architecture has three layers: an input layer, a hidden layer, and an output layer. The input layer takes the high-dimensional input, the hidden layer performs the compression, and the output layer reconstructs the original input. The number of neurons in the hidden layer is typically much smaller than the number of neurons in the input and output layers. This is what causes the compression and dimensionality reduction.There are also more complex autoencoder architectures, such as convolutional autoencoders, recurrent autoencoders, and variational autoencoders, which have additional layers and specialized architectures for handling different types of data.

To define it mathematically, suppose we have an input vector x of dimension n. We want to compress this vector into a lower-dimensional representation z of dimension m, where m ¡ n. We can achieve this by using an encoder function f(x) that maps the input vector to the lower-dimensional space:

$$z = f(x)$$

The encoder function f(x) is usually implemented as a neural network with multiple layers. Each layer is defined by a weight matrix W and a bias vector b. The output of each layer is computed by taking the dot product of the input with the weight matrix, adding the bias vector, and passing the result through an activation function:

$$h = g(Wx + b)$$

where h is the output of the layer, g is the activation function, and x is the input vector. The encoder function f(x) is defined as the composition of multiple layers:

$$f(x) = h_m = g(W_m g(W_{m1} ... g(W_1 x + b_1) ... + b_{m1}) + b_m)$$

where m is the number of layers in the encoder, W and b are the weight matrix and bias vector of each layer, and g is the activation function. Once we have the compressed representation z, we can use a decoder function g(z) to reconstruct the original input vector x:

$$\bar{x} = g(z)$$

The decoder function g(z) is also implemented as a neural network with multiple layers. The output of each layer is computed in the same way as the encoder function:

$$y = g(W^{z+b^)}$$

where y is the output of the layer, W' and b' are the weight matrix and bias vector of each layer, and g is the activation function. The decoder function g(z) is defined as the composition of multiple layers:

$$g(z) = f(W'_k f(W'_{k1} ... f(W'_1 z + b'_1) ... + b'_{k1}) + b'_k)$$

where k is the number of layers in the decoder, and W' and b' are the weight matrix and bias vector of each layer. The function f is the activation function, which is typically a non-linear function such as sigmoid, ReLU, or tanh. The goal of training an autoencoder is to minimize the difference between the original input x and the reconstructed output y, which is also known as the reconstruction error. The reconstruction error can be measured using a loss function, such as mean squared error (MSE):

$$L(x, y) = \frac{1}{n} \Sigma_{i=1}^{n} (x_i y_i)^2$$

where n is the dimension of the input and output vectors, and x_i and y_i are the i-th elements of the input and output vectors, respectively. To train the autoencoder, we use back-propagation to adjust the weights and biases of the encoder and decoder networks, to minimize the reconstruction error. This is done by computing the gradient of the loss function with respect to the weights and biases, and updating them using an optimization algorithms.

4.16.2. Experimental Settings and Results

The input layer is defined using the Keras API with a shape that is determined by the number of input features. This value is selected dynamically by the Flaml hyperparameter optimization framework and can range between 1 and 128. The encoder layer is defined with a Dense layer in Keras, and its number of neurons is set by Flaml. The activation function used in the encoder is Rectified Linear Unit (ReLU). Two additional encoder layers are defined, each with half the number of neurons as the previous layer, using the ReLU activation function. The decoder layers are defined with the same number of neurons as the previous encoder layer. The first decoder layer uses the ReLU activation function, while the remaining decoder layers use the ReLU activation function as well. The output layer of the autoencoder uses the Sigmoid activation function. The autoencoder is then compiled with an optimizer chosen by Flaml, such as Adam or SGD, a loss function such as Binary Cross-Entropy or Mean Squared Error, and other hyperparameters like epochs and batch size. The autoencoder is then trained on the training data for a certain number of epochs, which is also selected by Flaml. The validation data is used to evaluate the performance of the autoencoder during training. The trained autoencoder is then used to predict the reconstruction of the test data. The reconstruction error is calculated as the Mean Squared Error between the input and output of the autoencoder. Finally, the performance of the autoencoder is evaluated using the Receiver Operating Characteristic Area Under the Curve (ROC AUC) score. The hyperparameters for the autoencoder, including the number of neurons, optimizer, loss function, epochs, and batch size, are all dynamically selected by Flaml during the hyperparameter optimization process. The combination of hyperparameters that produces the best ROC AUC score on the validation set are then generalised (in this case, using mode, i.e. choosing the most repeated parameter suggestions) is selected as the final model.

As it turns out, using optimised hyperparameters improves the performance on 22 of the 40 training datasets but only 1 out of 10 test datasets as shown in figure 35 and figure 36. Furthermore, when the dataset is preprocessed, 21 out of the 40 training sets do better with optimised hyperparameters while 3 out of 10 do better on the test sets. It seems pre-processed data when coupled with individualistic best settings (i.e., best parameters of each individual dataset) does better than than when coupled with generalised best settings as in the case of the former, 36 out of the total 50 datasets had better AUC scores than when it was not preprocessed and the best individual parameters were applied.
As it turns out, using optimised hyperparameters improves the performance on 22 of the 40 training datasets but only 1 out of 10 test datasets. Furthermore, when the dataset is preprocessed, 21 out of the 40 training sets do better with optimised hyperparameters while 3 out of 10 do better on the test sets. It seems pre-processed data when coupled with individualistic best settings (i.e., best parameters of each individual dataset) does better than than when coupled with generalised best settings as in the case of the former, 36 out of the total 50 datasets had better AUC scores than when it was not preprocessed and the best individual parameters were applied.

4.17. Angle-Based Outlier Detection (ABOD)

[NTAM] Angle-based outlier detection or ABOD is a geometric approach to detect outliers in a dataset based on the variances of the angles a datapoint makes with any two other datapoints in the dataset. The general idea behind this approach is the variance of the angles between a point that is in a cluster and not an outlier and every other pair of points in the dataset will be higher. As Figure 10 shows the point which is far away from the clusters of points makes angles with the other two points showing low variance.

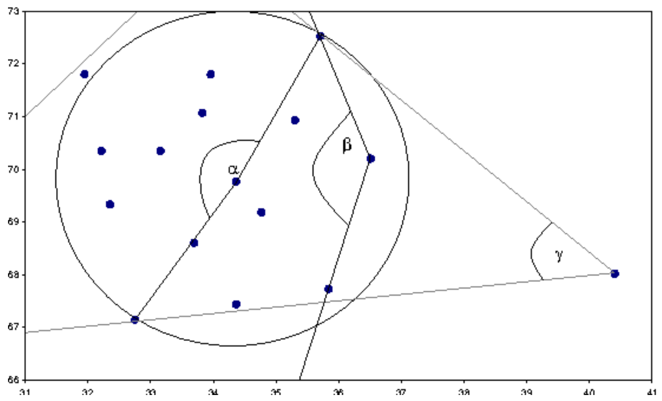


Figure 10. The angles a point makes with other two points (Kriegel et al., 2008)

In ABOD, a factor called angle-based outlier factor or *ABOF* is calculated for each datapoint. ABOF is defined as the variance over the angles a point makes with all pairs of points in the dataset. For a point $\vec{A}$ in the dataset $\mathcal{D}$ is defined as,

$$ABOF(\vec{A}) = VAR_{\vec{B},\vec{C}\in\mathcal{D}} \frac{(\overline{AB}, \overline{AC})}{||\overline{AB}||^2 \cdot ||\overline{AC}||^2}$$

Here, $\vec{B} \in \mathcal{D}\backslash\{\vec{A}\}$ and $\vec{C} \in \mathcal{D}\backslash\{\vec{A}, \vec{B}\}$. $ABOF$ is calculated for all points in the dataset and the points are sorted according to their $ABOF$. Figure 31 in Appendix shows the rank of the points shown in Figure 10. The outlier point is ranked as 1.

In ABOD for each point, every single pair of points in the dataset are considered to calculate the variance of their angles which makes the complexity $O(n^3)$. To reduce this complexity, a 'Fast' version of ABOD is introduced where only k-nearest neighbors of a point are considered while calculating the angles. This fast version reduces the complexity to $O(n^2 + n \cdot k)$. Hence, the fast version is useful for a large dataset (Kriegel et al., 2008).

4.17.1. Experiment Settings and Results

The fast ABOD is used for this experiment. The hyperparameter tuned for ABOD is the number of neighbors k which is the k nearest neighbors to be considered for a point while calculating the angles. The default value of k is 10. However, the mean of the best value of this hyperparameter on 40 datasets is found to be 24. Hence, the best hyperparameter value is found to be much higher than the default value.

4.18. Variational Autoencoder (VAE)

4.18.1. Algorithm

[NTAM] Autoencoder is a model built on two parts - an encoder and a decoder. The encoder takes the input and converts it into a lower-dimensional representation of latent space. The decoder then takes the latent space representation and converts it trying to keep it as close as possible to the input.

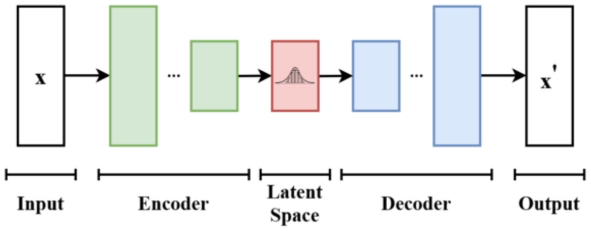


Figure 11. Structure of an autoencoder (Variational autoencoder)

Variational autoencoder or VAE is a type of autoencoder where the encoder and the decoder perform a bit differently. The encoder is probabilistic which takes the input and produces the parameters - the mean and the standard deviation of the input which are then fed into the decoder to reconstruct the input.

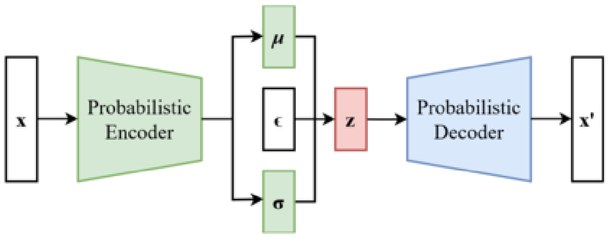


Figure 12. Structure of a variational autoencoder (Variational autoencoder)

Let The latent representation be z and the prior of the latent variable be $p_\theta(z) = \mathcal{N}(\boldsymbol{z}, \boldsymbol{0}, \boldsymbol{I})$. The posterior with multivariate Gaussian is defined as

$$log q_\phi(z|x^{(i)}) = log\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$

where μ is the mean and σ^2 is the standard deviation. As both the prior $p_\theta(z)$ and $q_\phi(z|x)$ are Gaussian, the loss can be computed with KL divergence as follows,

$$\mathrm{L}(\theta, \phi; x^(i)) \simeq \frac{1}{2}\sum_{j=1}^{J}\Big(1 + log((\sigma_j^{(i)})^2) - ((\mu_j^{(i)})^2) - ((\sigma_j^{(i)})^2)\Big) + \frac{1}{L}\sum_{l=1}^{L} log p_\theta(x^{(i)}|z^{(i,l)})$$

(Kingma & Welling, 2022)

4.18.2. Experiment Settings and Results

Table 6 in Appendix shows the hyperparameters of variational autoencoder which are tuned, their default value, their suggested value after tuning with flaml and the measure of central tendency used to get the corresponding suggested hyperparameter value. Mean is used to get the suggested value of encoder neurons, epochs, dropout rate and L2 regularizer while the mode is used for the optimizer, output activation, hidden activation and loss function. The default value of encoder neurons does not differ much from the suggested values of this. The suggested value of the number of epochs is 415 which is about four times higher than the default value 100. The suggested dropout rate 0.72 is much higher than the default value 0.2 and there is a 50% increase in the L2 regularizer's suggested value compared to its default value 0.1. Figure 32 in Appendix shows the provided values for optimizer, output activation, hidden activation and loss function for flaml tuning and their proportion in 40 training datasets after the tuning. The suggested optimizer is Adagrad and the suggested output activation is Relu. These two differ from their default value and they are the best ones after flaml tuning for almost half of the training datasets. The default hidden activation is Relu but for all 40 training datasets, Sigmoid performed best after flaml tuning. The default loss function is mean squared error or mse and the suggested one is also mse. Interestingly, mse is found as the best loss function for all training datasets after flaml tuning. Hence, the suggested loss function stays the same for all training datasets after flaml tuning while the hidden activation function is found to be completely different for all training datasets.

5. Algorithms' Comparison

[RK] Various types of tests and comparisons of AUC values are presented in this section. These comparisons have been made between all the algorithms using the tuned, default, and best hyperparameters.

5.1. Comparison the Ranking of Algorithms using CD

[RK] The first comparison examines the ranking between the algorithms. To do this, a critical difference (CD) plot was used. This is a robust statistical tool for comparing the results of experiments over multiple observations. The CD plot contains vertical lines representing the means of the results of the experiments being compared, and horizontal lines indicating whether or not the compared entities have significant differences (Demšar, 2006).

”Figure 37” shows the ranking of all algorithms using the tuned hyperparameters. In (a), the ”kde” algorithm has the best ranking with a score of 5.5, closely followed by ”inner” with a score of 5.6 and ”knn” with a score of 6.7. Meanwhile, in (b), the ”knn” algorithm had the highest percentage of the top 1 algorithms among the 40 datasets, ranked number 1 for 16. However, for the remaining datasets, ”knn” was ranked lower than the other algorithms. In addition, the ”kde” algorithm was ranked as number 1 for only two of the datasets but performed better on average than the other algorithms.

”Figure 38” and ”Figure 39”, use the default and the best hyperparameters, show that the four best performing algorithms are ”lunar”, ”ifor”, ”kde” and ”knn”. Although the ”ifor” algorithm has never been the best-performing algorithm across the 10 test datasets, ”ifor” is still considered one of the best-performing algorithms, with a ranking between 5.5 and 6.5. Conversely, the ”abod” algorithm was the best-performing algorithm only twice out of the 10 test datasets, and performed poorly on the remaining datasets, resulting in a ranking between 10 and 10.5.

Moreover, the rankings across the figures ”Figure 38”, and ”Figure 39” consistently demonstrate that there is no significant difference among all the algorithms, as evidenced by the horizontal line in the figures. In addition, the more complex algorithms such as ”GAN”, ”NF”, ”DeepSVDD”, and ”LOF” have the worst rankings compared to the simpler algorithms such as ”lunar”, ”knn”, ”ifor”, ”kde”, and ”inne”, which consistently perform the best.

5.2. Behavior of Algorithms in higher dimensions

[SG] The behavior of the 18 algorithms was analyzed in high dimensions. Let us consider ”Figure 40” in Appendix, the x-axis represents algorithms, and the y-axis represents the AUC score for the algorithms. We took two datasets: ”fashion0.npz” has 784 dimensions, and ”har.npz” has 561 dimensions. The AUC scores in default and best hyperparameter cases were compared. It is observable that most of the algorithms show the same pattern for both datasets. Interestingly, the algorithm ”NF” has an AUC score of 0.5 in all cases. The ”abod” algorithm has a high AUC score for the ”har.npz” dataset; however, it has a low AUC score for the ”fashion0.npz” dataset. On the contrary, the algorithm ”gan” has a contrasting pattern.

5.3. Comparison of Algorithms with Best and Default hyperparameters

[SG] The AUC scores for the 18 algorithms were analyzed via the Wilcoxon test for default and best hyperparameters. The Wilcoxon test (R. Lyman Ott, 2015) compares two groups to check if they are significantly different from each other. The null hypothesis suggests that there is no significant difference between the two groups, while the alternate hypothesis suggests that there is a significant difference between the two groups. The risk of increased type I error has occurred because of multiple comparisons. The value of α (the level of significance) is considered 0.05. The family-wise error rate (FWER) (R. Lyman Ott, 2015) is the probability of creating at least one type I error in a set of hypothesis tests. The calculated value of the family-wise error rate is FWER = $1 - (1 - \alpha)^n$ = 0.6028. It means that the probability of occurring a type I error is approximately 60.28%. The correction method adjusts the p-value while keeping the α-value the same. Let us consider ”Figure 41” in Appendix, the x-axis represents algorithms, and the y-axis represents the corrected p-values for the algorithms. Most of the p-values are near to one because of the adjustment of the p-values. Only the ”randnet” algorithm has a p-value less than α, so the null hypothesis can not be accepted for this algorithm. It indicates that there is a significant difference between the AUC scores in default and best hyperparameters for the ”randnet” algorithm. However, this is not the case for the other algorithms.

5.4. Comparison of Autoencoders

[SG] Finally, the Autoencoder (under) and Autoencoder (over) were compared. The AUC scores for default and best hyperparameters were analyzed using boxplots and the Wilcoxon test. The value of α (the level of significance) is considered 0.05. The calculated p-value is 0.08 for the default hyperparameters, so the null hypothesis can not be rejected. It indicates that there is no significant difference between the Autoencoder (under) and Autoencoder (over) in the case of default hyperparameters. However, The calculated p-value is 0.04 for the best hyperparameters, so the null hypothesis can not be accepted. It indicates that there is a significant difference between the Autoencoder (under) and Autoencoder (over) in the case of best hyperparameters. ”Figure 42” in Appendix displays that the boxplots were generated for both cases as well. The right subfigure also shows that the median for Autoencoder (over) is higher than for Autoencoder (under). It reveals that the Autoencoder (over) performs better than the Autoencoder (under) in this distribution of data and scenario.

6. Summary

[IK] Anomaly detection is a vital component of many applications, including fraud detection, network intrusion detection, and fault detection. In recent years, various algorithms have been proposed for anomaly detection, including nearest neighbour methods, neural networks, and Isolation Forest. A research has been conducted to compare the performance of these algorithms across different hyperparameter values and data sets.The study finds that algorithms based on nearest neighbour methods, such as iNNE, KNN, Lunar, and KDE, outperform most of the neural network models in detecting anomalies. While the variance of AUC scores of most neural network algorithms, across the different hyperparameter setting, was higher than the AUC scores of Nearest Neighbour algorithms. This finding indicates that it is harder to autotune neural networks than Nearest Neighbour algorithms, which can lead to sub-optimal performance. However, few neural network algorithms, including VAE, and DEAN, are able to perform among the best algorithms when the hyperparameter values are tuned. Especially, in large and high dimensional data sets. During our efforts to minimize the number of potential hyperparameter value combinations, we discovered that the default hyperparameter values did not produce optimal results across all data sets, primarily due to their varying dimensionalities. Consequently, we needed to adjust the hyperparameter values to suit the specific data set under consideration. Our study emphasizes the importance of understanding the data set's structure and quality, as it enables us to efficiently narrow down the search range for optimal hyperparameter values. Similarly, nearest neighbor models, including LOF, KNN, and ABOD, display a positive correlation between the data set dimensions and the model's complexity, which is determined by the number of nearest neighbors considered. Comparable observations can be made regarding iNNE, which employs nearest neighbor techniques, where we find a direct relation between the sample size hyperparameter and the data set size. Finally, ensemble methods and tree-based algorithms, such as DEAN and Isolation Forest, that utilize feature bagging selection are also susceptible to the influence of irrelevant features in the data, which can dominate and distort the anomaly score due to the random nature of the selection process. Through our research, we have discovered that a bag size larger than the thresholds we determined during our empirical analysis had negatively impacted the model's performance, emphasizing the need for meticulous selection of hyperparameter values.

[IK] To this end, we conclude that the data set's properties, including its dimensions and quality, play a critical role in determining the hyperparameter values for an anomaly detector. Choosing the right hyperparameters based on the data set's characteristics can result in better model performance and faster training times. Conversely, selecting inappropriate hyperparameters can lead to poor model performance and overfitting or underfitting. Thus, understanding the dataset's structure and quality is essential for identifying the optimal hyperparameters for an anomaly detector.

[UI]The process of tuning the hyperparameters also brought out some curious features in the execution of certain algorithms and their behaviours in certain scenarios. It was seen that the Normalizing Flows algorrithm behaves strangely in that it consistently achieves an AUC score of 0.5 in high dimensional data. The AnoGAN algorithm, the number of nodes jump up and down in a zigzag pattern with peaks for the number of nodes occurring in the same layer for a number of datasets. Further analyses of these behaviours might be warranted before any significant conclusions can be made about their performances.

7. Conclusion

[UKD] A lot of algorithms were used in this study and the consensus with all of them is that hyperparameter tuning improved the performance of all the algorithms on almost all the datasets. It is evident that the selection of appropriate hyperparameters is a critical aspect of machine learning, particularly for anomaly detection. The hyperparameters control the behavior of the algorithms, and the optimal set of hyperparameters can significantly impact the performance of the models. Therefore, it is essential to tune the hyperparameters to achieve the best results. This statement, however, comes with a caveat: when tuning in flaml, the best parameters for each dataset varied and we tried to generalise those multiple parameters by either averaging them, choosing the median, or picking the most repeated one (mode). It was found out that the generalising the best parameters found for all the datasets often leads to worse results, which could indicate that there is no 'one shoe fits all' solution when it comes to hyperparameter tuning and that each dataset may needs it own set of tuned hyperparameters which suit its structure better (or, if generalising is indeed necessary, then finding another way to generalise instead of picking the mean, median, or mode could be better).

While this is not always true, there were some instances where a higher number of features led to better results in most cases after normalising the data. This, however, might just be a case of corelation rather than causation. Delving a bit deeper, for a large number of samples **and** features, density-based models Kernel Density Estimation and Gaussian Mixture Model performed brilliantly (above 0.9 AUC score) for all training datasets and, except for GMM in Fashion0 dataset, also put up good AUC numbers (above 0.8). Considering that these models are density-based, it is perhaps an indication that these models do better with bigger dataset with a number of features.

Similar behaviour was also encountered with ensemble models. With the exception of the dataset Annthyroid, every single large dataset with a lot of features from both test and train gave good results (greater than 0.9 AUC score) when an ensemble model was trained on them. Given that ensemble models are amalgamations of multiple different models, it is not that suprising that a high number of data points and features leads it to learn better and, therefore, give better results. As mentioned earlier, the general trend with the datasets was somewhat 'more the merrier', in that higher number of features coupled with more samples was helpful to the models in providing higher scores but this is *especially* true with density-based and ensemble models, since the other cluster of models (outlier detection, generative models, and autoencoders) used here didn't all perform unanimously well with dataset of high dimensionality and points. In short, with hyperparameter tuning, density-based, ensemble, and nearest neighbour models performed the best (AUC above 0.8) when compared to autoencoders, generative, and outlier-based methods. In fact, ironically, outlier-based methods performed the worst (AUC 0.73) after generative models (AUC 0.72).

Apart from that, it is also noteworthy that the performance of nearly all the algorithms is poor for Pima, Liver 1, optdigits, and Diabetes Present. This could indicate towards bad features being selected. This could also mean that the number of normal and anomalous data points in Pima, optdigits and Steel Plates datasets may be imbalanced, which can lead to biased performance metrics. Anomaly detection algorithms may have difficulty detecting anomalies in datasets where the number of anomalies is much smaller than the number of normal data points. This leads us right to the next point: dataset preprocessing.Since all the algorithms used here are simply from imported libraries, there is little scope to change the internal structures of the algorithms (one can, of course, fiddle around with the package files but we didn't do any of that here). As a result, we looked at the effect of manipulating the data that is being fed to the algorithms for the autoencoder algorithm.

Unsurprisingly, merely normalising the data led to better results for nearly all the datasets, but especially for the aforementioned four datasets. Indeed, with the data normalised, all but Pima, shot up to over 0.75 AUC score, with optdigits marking the most remarkable improvement, going from under 0.6 to 0.99 AUC score (it must also be added that optdigits seemed like the most volatile dataset, having a range of results with different algorithms). Finally, it is important to note that the performance of the algorithms was heavily dependent on the quality and quantity of data available. All the algorithms were run on merely 50 datasets in total, which is not much in the world of Machine Learning, where big data is necessary to yield the best results (Lee et al., 2019).

In conclusion, hyperparameter tuning, especially catered towards the dataset and not generalised, leads to better scores, so does pre-processing the dataset to a certain extent (one can only imagine how much better the scores could have been had the pre-processing been done after looking at the inherent structure of the table and not just a general normalisation). Furthermore, some type of models work better than other types of models (and it seems like models made specifically for outlier detection are not the best performing ones in this study). Finally, more training on more different datasets is needed to get a better understanding of how these models could work as only 50 datasets (in total) is perhaps not as much when it comes to tuning Machine Learning algorithms.

Acknowledgements

We would like to thank Prof. Dr. Emmanuel Müller (Chair of Data Science and Data Engineering, Technical University of Dortmund) for providing the opportunity to study this topic. We also want to thank our supervisor, Simon Klüttermann, who guided us throughout this case study.

References

Kolmogorov–Smirnov Test, pp. 283–287. Springer New York, New York, NY, 2008. ISBN 978-0-387-32833-1. doi: 10.1007/978-0-387-32833-1_214. URL https://doi.org/10.1007/978-0-387-32833-1_214.

Flow-based generative model. https://https://en.m.wikipedia.org/wiki/Flow-based_generative_model, 2023. Accessed on 2023-02-25.

A. Krizhevsky, G. H. Learning multiple layers of features from tiny images. 29:141–142, 2009.

Aggarwal, C. C. Outlier ensembles: position paper. *ACM SIGKDD Explorations Newsletter*, 14(2):49–58, 2013.

Bandaragoda, T. R., Ting, K. M., Albrecht, D., Liu, F. T., and Wells, J. R. Efficient anomaly detection by isolation using nearest neighbour ensemble. In *2014 IEEE International Conference on Data Mining Workshop*, pp. 698–705, 2014. doi: 10.1109/ICDMW.2014.70.

Bang Xiang Yong, A. B. Do autoencoders need a bottleneck for anomaly detection? *arXiv preprint*, 2022.

Böing, B., Klüttermann, S., and Müller, E. Post-robustifying deep anomaly detection ensembles by model selection. In *22nd IEEE International Conference on Data Mining (ICDM; In Press)*.

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

C. Tachtatzis, G. Gourlay, I. A. O. P. Sensor data set radial forging at afrc testbed v2, 2019. URL https://doi.org/10.5281/zenodo.3405265.

Chandola, V., Kumar, V., and Banerjee, A. Anomaly detection : A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

Chen, J., Sathe, S., Aggarwal, C., and Turaga, D. Outlier detection with autoencoder ensembles. *Society for Industrial and Applied Mathematics (SIAM)*, 6(1):90–98, 2017.

Datrics. Isolation forest model. https://wiki.datrics.ai/isolation-forest-model.

Demšar, J. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29:141–142, 2017.

Dey, Islam, U. K., and Sajjatul, M. Genetic expression analysis to detect type of leukemia using machine learning. volume 59, pp. 492–499. ACS Publications, 2019.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

E. Nalisnick, A. Matsukawa, Y. W. T. D. G. B. L. Do deep generative models know what they don't know?, in: International conference on learning representations. 2019.

Fortran, I., Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. Numerical recipes. *Cambridge, UK, Cambridge University Press*, pp. 842–847, 01 1992.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Goodge, A., Hooi, B., Ng, S. K., and Ng, W. S. Lunar: Unifying local outlier detection methods via graph neural networks. 2022.

H. Xiao, K. Rasul, R. V. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint*, 2017.

Hamilton, W. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14:48–49, 09 2020. doi: 10.2200/S01045ED1V01Y202009AIM046.

Han, X., Chen, X., and Liu, L.-P. Gan ensemble for anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4090–4097, 2021.

Hawkins, D. M. *Identification of outliers*, volume 11. Springer, 1980.

Haynes, W. *Bonferroni Correction*, pp. 154–154. Springer New York, New York, NY, 2013. ISBN 978-1-4419-9863-7. doi: 10.1007/978-1-4419-9863-7_1213. URL https://doi.org/10.1007/978-1-4419-9863-7_1213.

Jonas Soenen, Elia Van Wolputte, L. P. V. V. W. M. J. D. H. B. The effect of hyperparameter tuning on the comparative evaluation of unsupervised anomaly detection methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, pp. 1–4, 2021.

Kim, D., Cha, J., Oh, S., and Jeong, J. Anogan-based anomaly filtering for intelligent edge device in smart factory. In *2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, pp. 1–6. IEEE, 2021.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes, 2022.

Klüttermann, S. Unpublished work.

Knorr, E. M. and Ng, R. T. Algorithms for mining distance-based outliers in large datasets. In *Very Large Data Bases Conference*, 1998.

Kriegel, H.-P., Schubert, M., and Zimek, A. Angle-based outlier detection in high-dimensional data. KDD '08, pp. 444–452, New York, NY, USA, 2008. Association for Computing Machinery.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lee, J., Cheng, T., Zhu, W., Zhang, Y., and Li, Y. Data size matters: Effectiveness of large data for machine learning model selection in drug discovery. *Journal of Chemical Information and Modeling*, 59(1):492–499, 2019.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *2008 eighth ieee international conference on data mining*, pp. 413–422. IEEE, 2008.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

Mohammed J. Zaki, W. M. J. *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge University Press, 2020.

R. Lyman Ott, M. L. *An Introduction to Statistical Methods and Data Analysis, Seventh Edition*. Cengage Learning, Boston, USA, 2015.

Rayana, S. Odds library, 2016. URL http://odds.cs.stonybrook.edu.

Roberto Aurelia, Nicolo Brandizzia, G. D. M. and Brociekb, R. A customized approach to anomalies detection by using autoencoders. *Scholar's Yearly Symposium of Technology, Engineering and Mathematics*, 2021.

Rondina, J., Oliveira, L., Marquand, A., Dresler, T., Leitner, T., Fallgatter, A., Shawe-Taylor, J., and Mourão-Miranda, J. Scors—a method based on stability for feature selection and apping in neuroimaging. *Medical Imaging, IEEE Transactions on*, 33:85–98, 01 2014. doi: 10.1109/TMI.2013.2281398.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. Deep one-class classification. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4393–4402. PMLR, 10–15 Jul 2018a. URL https://proceedings.mlr.press/v80/ruff18a.html.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. Deep one-class classification. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4393–4402. PMLR, 10–15 Jul 2018b. URL https://proceedings.mlr.press/v80/ruff18a.html.

Sammut, C. and Webb, G. *Encyclopedia of Machine Learning*. Springer, New York, USA, 01 2011.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *Information Processing in Medical Imaging: 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*, pp. 146–157. Springer, 2017.

T. Schneider, S. Klein, M. B. Condition monitoring of hydraulic systems data set at zema, 2018. URL https://doi.org/10.5281/zenodo.1323611.

Variational autoencoder.

Varun Chandola, Arindam Banerjee, V. K. Anomaly detection : A survey. *ACM computing surveys*, pp. 1–72, 2009.

Wang, C., Wu, Q., Weimer, M., and Zhu, E. FLAML: A Fast and Lightweight AutoML Library, 2021.

Y. Netzer, T. Wang, A. C. A. B. B. W. A. Y. N. Reading digits in natural images with unsupervised feature learning, in: Nips workshop on deep learning and unsupervised feature learning. 2011.

Yi-Ren Yeh, Z.-Y. L. and Lee, Y.-J. *New Advances in Intelligent Decision Technologies*. National Taiwan University of Science and Technology, Taipei,Taiwan, 2009.

Zhao, Y., Nasrullah, Z., and Li, Z. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019. URL http://jmlr.org/papers/v20/19-011.html.

Zhao, Y., Chalapathy, R., Rehman, S., and Kumar, V. Pyod documentation. https://pyod.readthedocs.io/en/latest/pyod.models.html, 2021. Accessed on 2023-02-25.

A. Tables

Table 1. Hyperparameters for Iforest Algorithm

Hyperparameters	Default (Zhao et al., 2021)	Best
n estimators	100	563
max features	$n\_samples$	min(12, $n\_samples$)
contamination	0.1	0.02
random state	random value	224
AUC Score	0.82	0.83

Table 2. Hyperparameters for AnoGAN Algorithm

Hyperparameters	Default (Zhao et al., 2021)	Best
Hidden Layers Activation Function	tanh	None
Output Layers Activation Function	None	softmax
# Layers in Generator	5	6
# Layers in Discriminator	3	9
# Nodes for each Layers in Generator	[20, 10, 3, 10, 20]	[6, 45, 2, 41, 2, 7]
# Nodes for each Layers in Discriminator	[20, 10, 5]	[6, 14, 3, 2, 21, 5, 3, 26, 26]
Learning Rate	0.001	0.02
Dropout rate	0.2	0.03
epochs	500	722
Batch Size	32	404
AUC Score	0.49	0.53

Table 3. Hyperparameters and AUC scores for Local Outlier Factor Algorithm

Hyperparameters	Default Value (Zhao et al., 2021)	Optimal Value
$n\_neighbors$	20	48
AUC Score	0.62	0.68

Table 4. Hyperparameters and AUC scores for Normalizing Flows Algorithm

Hyperparameters	Default Value (Klüttermann)	Optimal Value
batch size	30	62
number of splits	10	11
epochs	1000	98
realisation	gauss	biased
mixture	No	No
non linear	False	True
AUC Score	0.57	0.60

Table 5. Comparison results from Loda algorithm performance based on test datasets

dataset	auc score	mean auc score	DT auc score	mean-default	DT-default
breastw	0.977726	0.205834	0.184917	-0.77189	-0.79281
cover	0.846046	0.971018	0.975973	0.124972	0.129927
Diabetes_present	0.6064	0.692311	0.6972	0.085911	0.0908
fashion0	0.883728	0.92235	0.918784	0.038622	0.035056
har	0.830493	0.875859	0.873774	0.045366	0.043281
Liver_1	0.5841	0.515414	0.442942	-0.06869	-0.14116
optdigits	0.529378	0.604356	0.583556	0.074978	0.054178
pima	0.692889	0.730444	0.740133	0.037556	0.047244
steel-plates-fault	0.555625	0.499236	0.509028	-0.05639	-0.0466
wbc	0.995465	1	1	0.004535	0.004535

Table 6. Default value and the suggested by flaml values of hyperparameters for Variational Autoencoder

Hyperparamter	Default value	Suggested hyperparameter by flaml	Measure of central tendacy used
Encoder Neurons	[128, 64, 32]	[104, 24, 18]	Mean
Epochs	100	415	Mean
Dropout rate	0.2	0.72	Mean
L2 regularizer	0.1	0.15	Mean
Optimizer	Adam	Adagrad (41%)	Mode
Output activation	Sigmoid	Relu (48%)	Mode
Hidden activation	Relu	Sigmoid (100%)	Mode
Loss	MSE	MSE (100%)	Mode

Table 7. Settings of Default and Best hyperparameters in LUNAR

Hyperparameters	Default value	Best value
$model\_type$	WEIGHT	WEIGHT
$negative\_sampling$	MIXED	UNIFORM
$n\_neighbours$	5	3
$epsilon$	0.1	0.03
lr	0.001	0.003
n_epochs	200	758
$proportion$	1.0	6.0

Table 8. Settings of Default and Best hyperparameters in GMM

Hyperparameters	Default value	Best value
$n\_components$	1	5
$covariance\_type$	full	full
$max\_iterations$	100	38

Table 9. AUC scores with default parameters and best suggested parameters for PCA Algorithm on test datasets

Datasets	AUC scores (default params)	AUC scores (best suggested params)
Diabetes_present	0.6896	0.6277
har	0.8867	0.8895
cover	0.9432	0.9172
breastw	0.9922	0.9921
fashion0	0.9024	0.9020
pima	0.7194	0.6559
Liver_1	0.5614	0.6284
wbc	0.9932	0.9932
steel-plates-fault	0.7176	0.7097
optdigits	0.5320	0.5339

Table 10. Comparison between Autoencoder (under) and Autoencoder (over) on test datasets with best suggested parameters

Datasets	AUC scores of Autoencoder(under)	AUC scores of Autoencoder(over)
Diabetes_present	0.5474	0.7064
har	0.8593	0.8858
cover	0.9627	0.9429
breastw	0.9919	0.9919
fashion0	0.9040	0.9228
pima	0.5591	0.6078
Liver_1	0.5690	0.5722
wbc	0.8980	0.9846
steel-plates-fault	0.7297	0.7271
optdigits	0.4981	0.5245

B. Figures

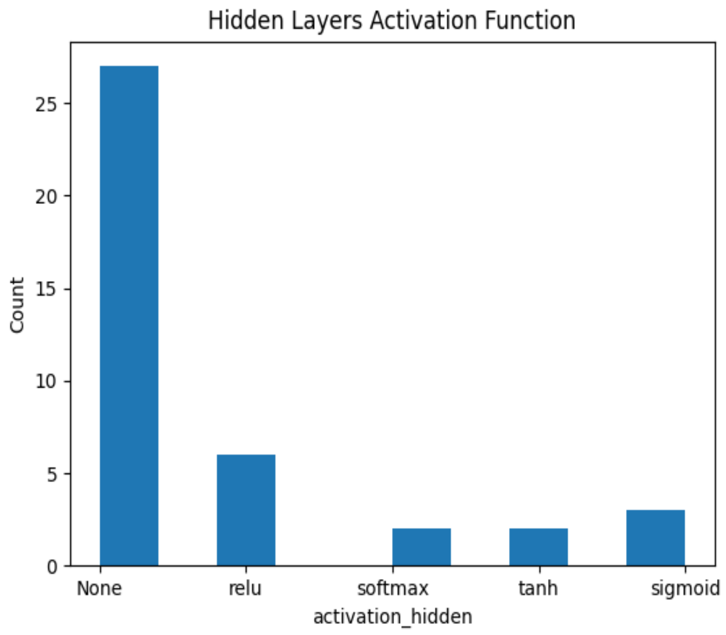


Figure 13. Frequently Hidden Layer Activation Function

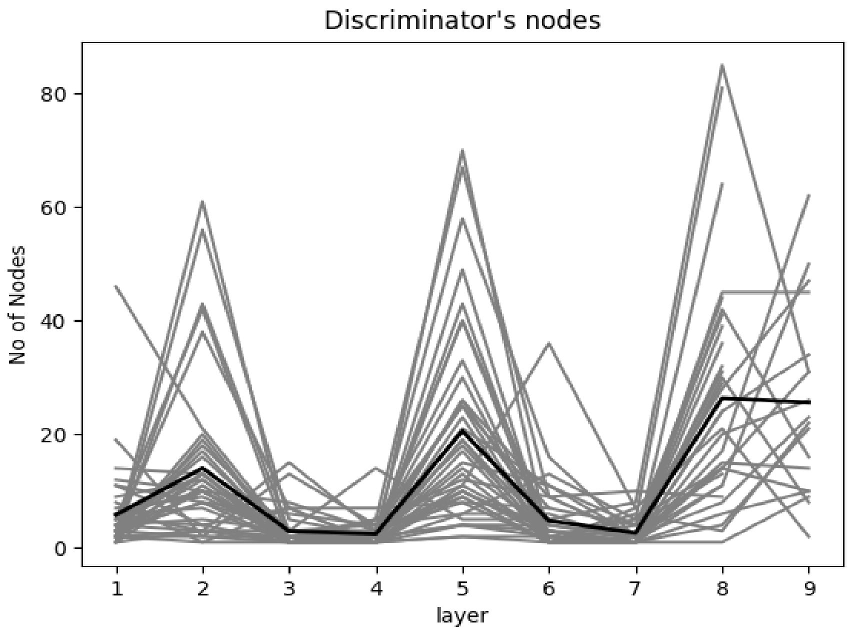


Figure 14. Number of Nodes in each Layer in Discriminator

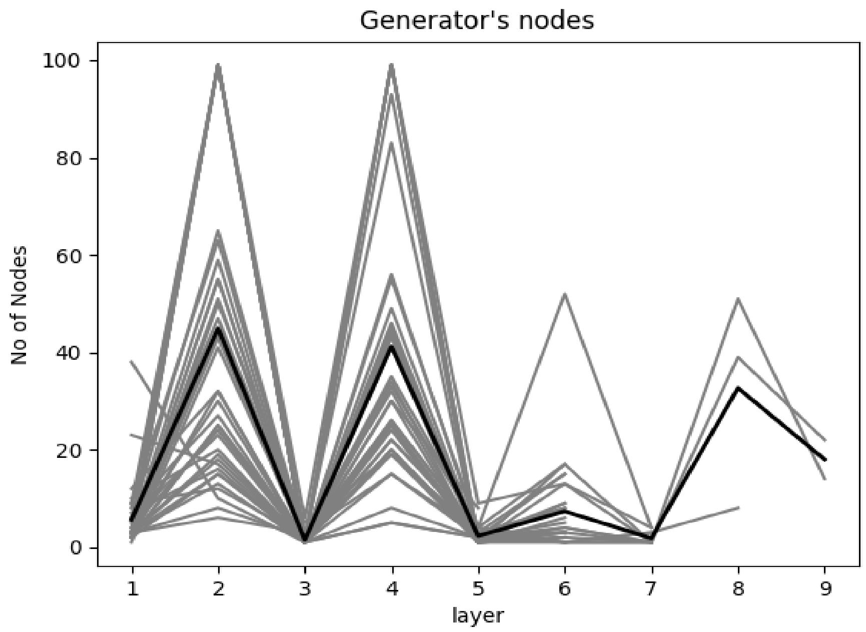


Figure 15. Number of Nodes in each Layer in Generator

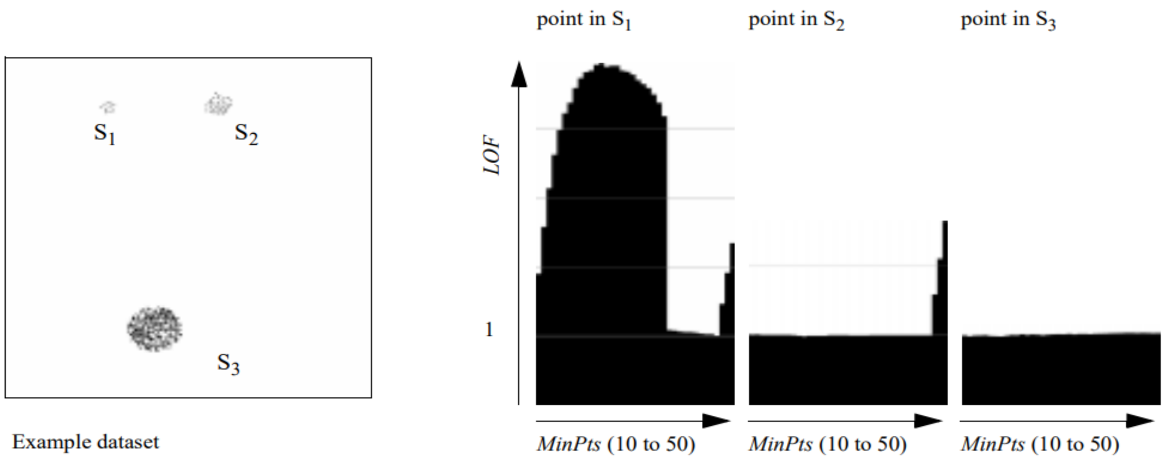


Figure 16. Effect of MinPts on LOF values (Breunig et al., 2000)

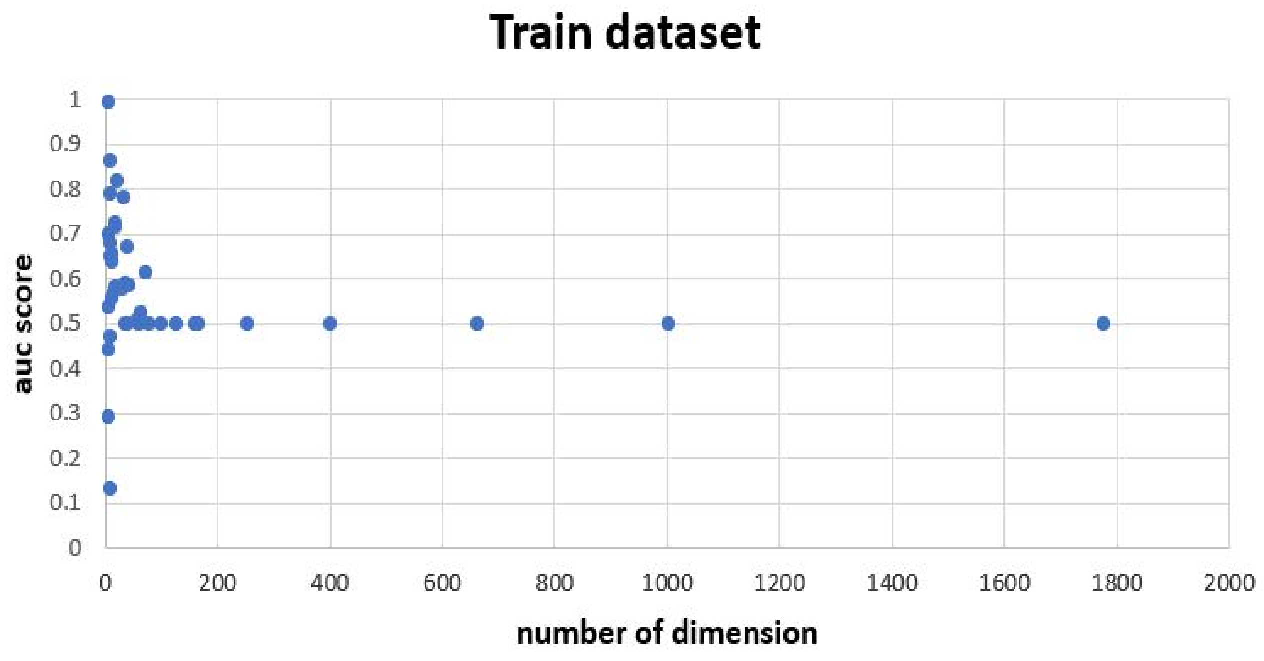


Figure 17. Interesting behavior of normalizing flows algorithm in high dimensions for the train dataset

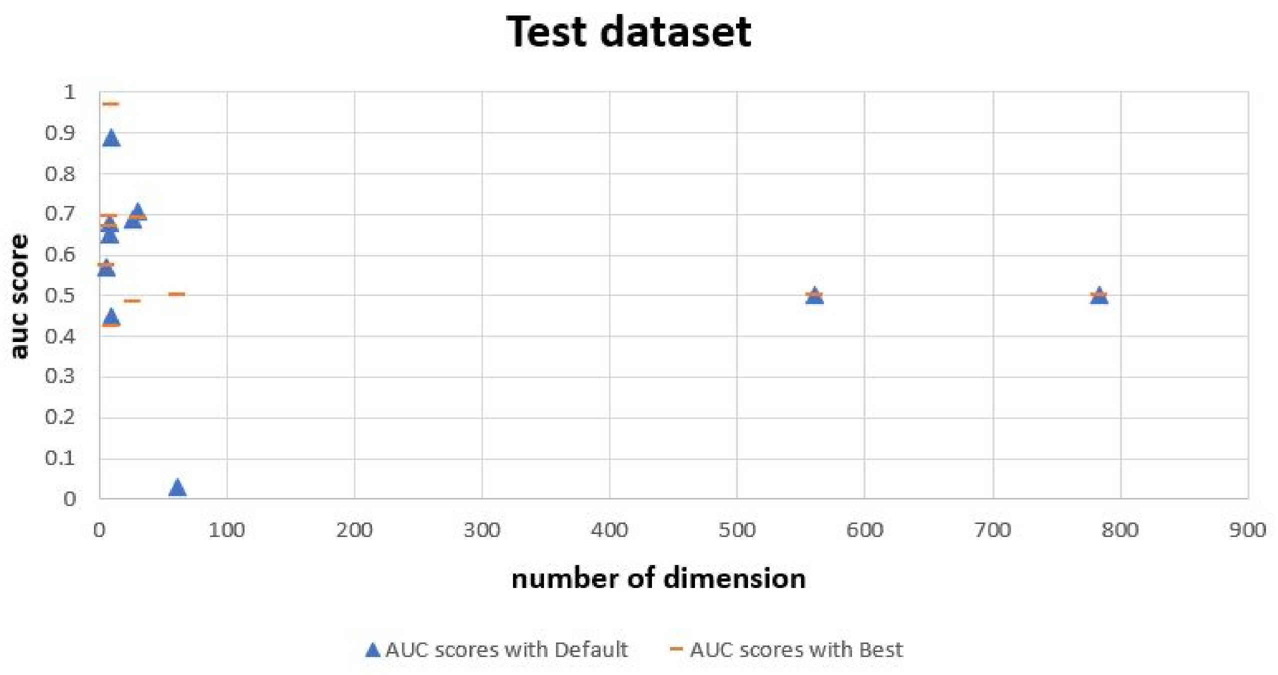


Figure 18. Interesting behavior of normalizing flows algorithm in high dimensions for the test dataset (default and best hyperparameters)

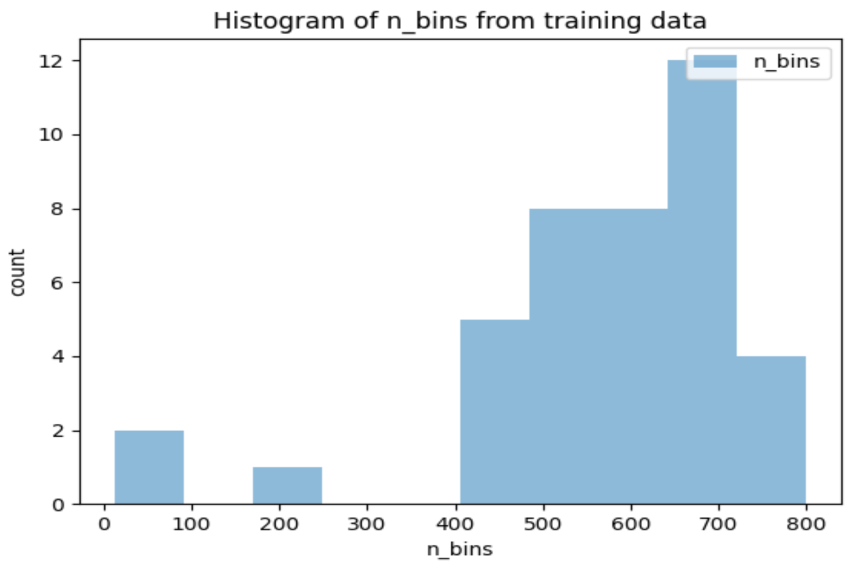


Figure 19. Frequency distribution of tuned parameter number of bins in Loda algorithm from 40 training datasets.

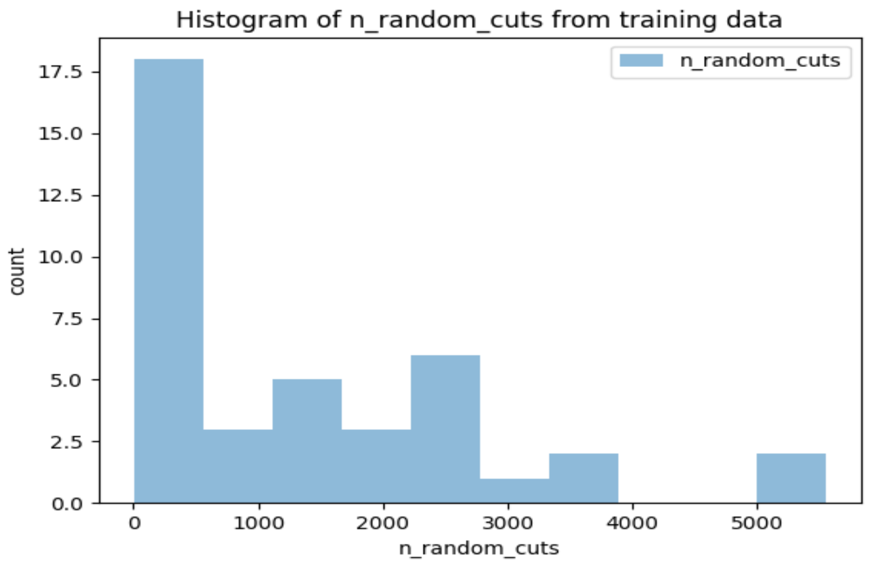


Figure 20. Frequency distribution of tuned parameter number of random cuts in Loda algorithm from 40 training datasets.

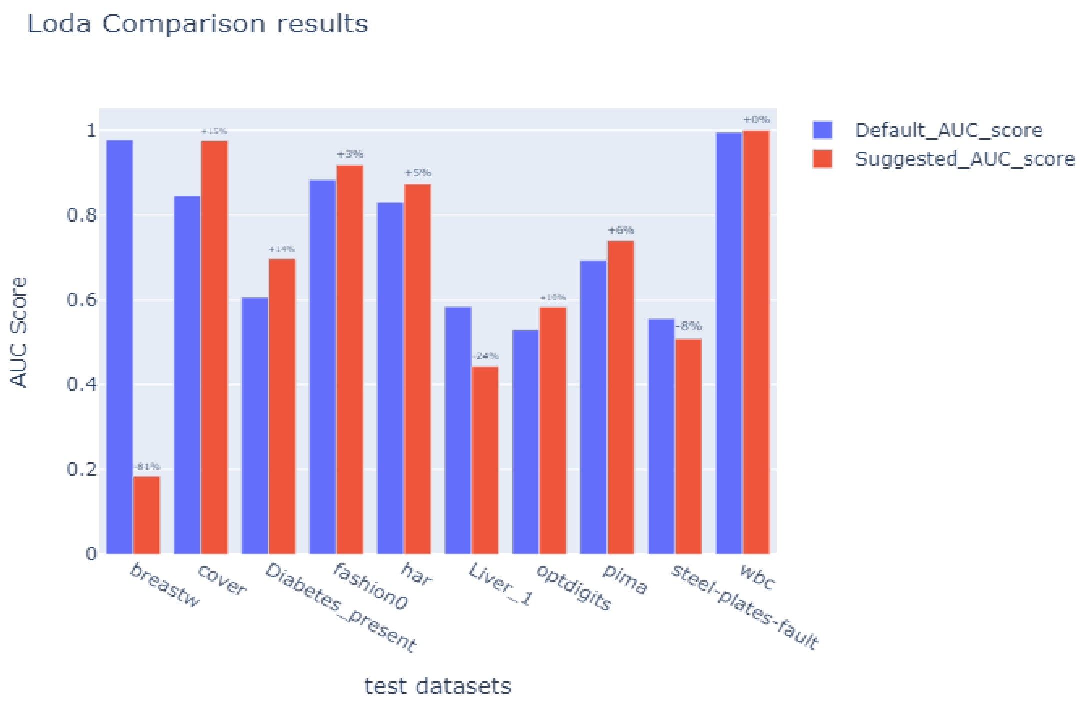


Figure 21. Performance graph of Loda algorithm between decision tree suggested parameters and default parameters

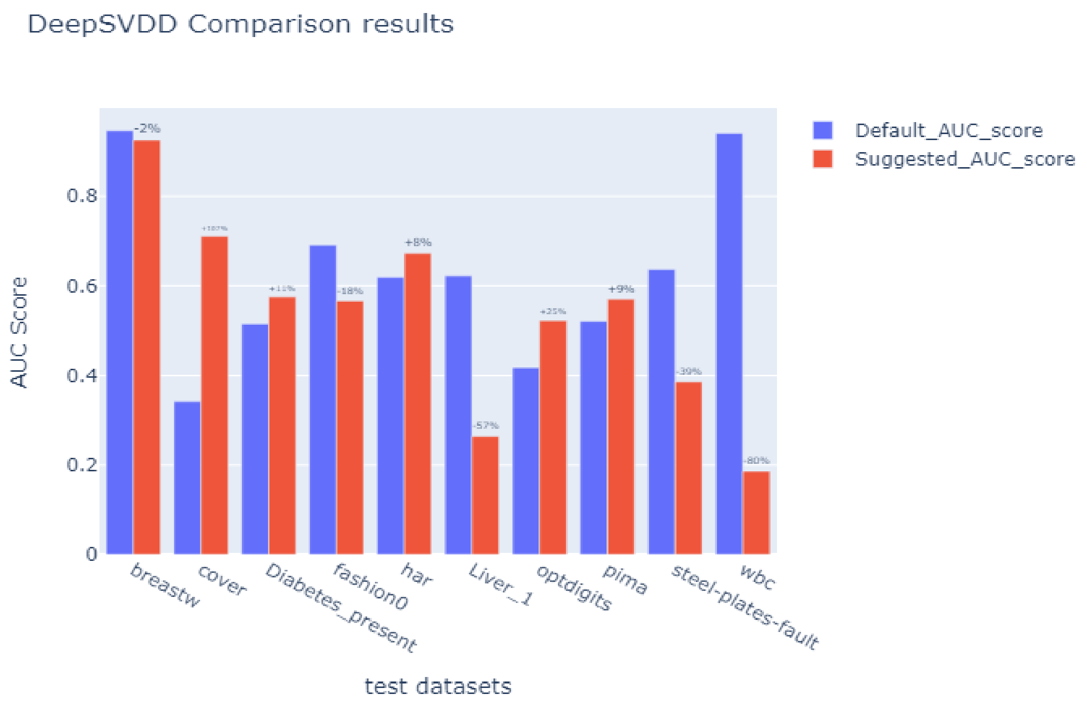


Figure 22. Performance graph of DeepSVDD algorithm between decision tree suggested parameters and default parameters

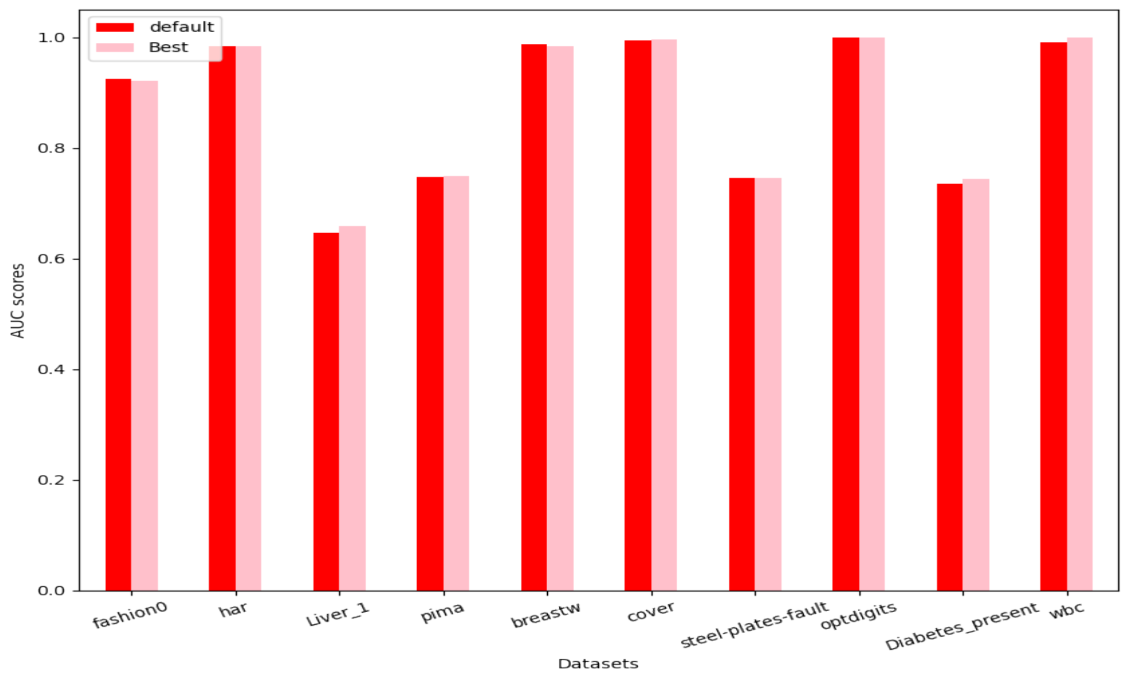


Figure 23. Comparison of best hyperparameters with the default parameters of LUNAR algorithm

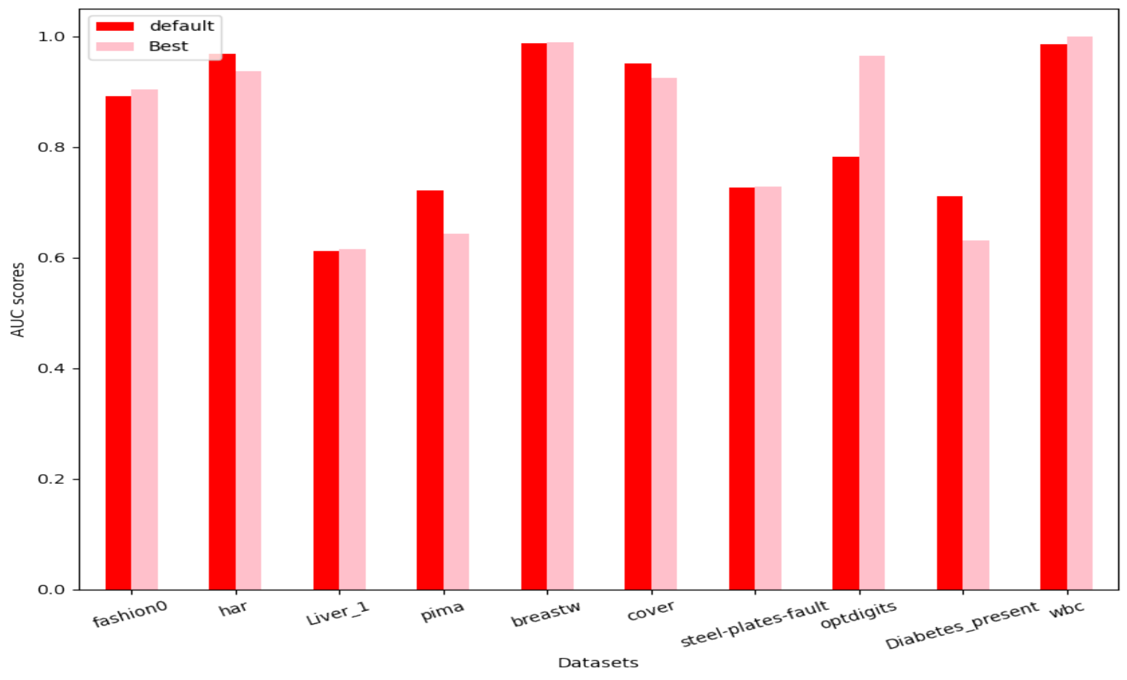


Figure 24. Comparison of best hyperparameters with the default parameters of GMM algorithm

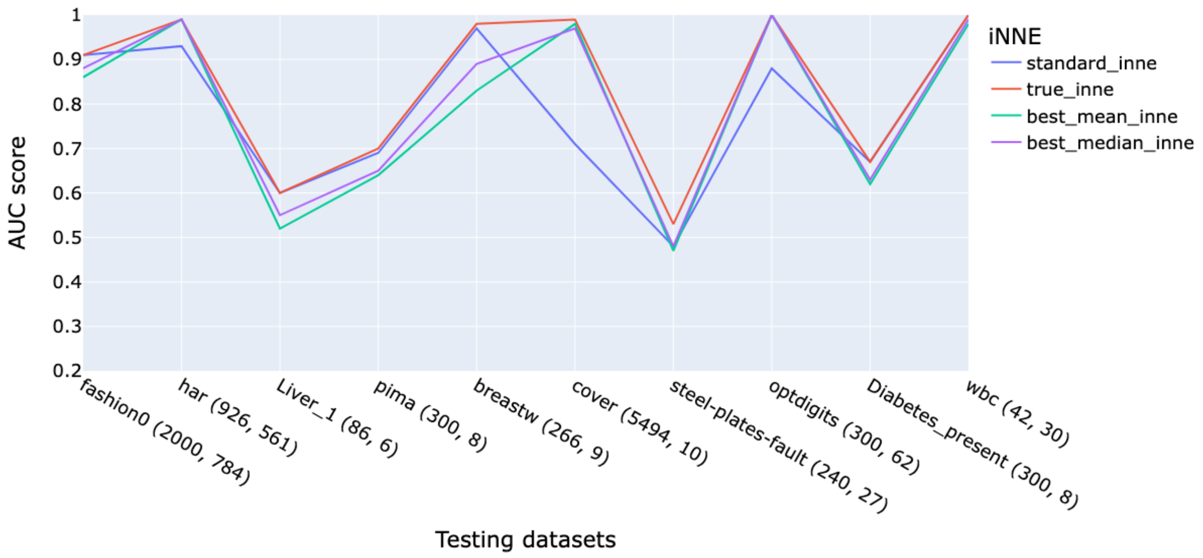


Figure 25. Graph shows AUC scores of iNNE models obtained with 4 different hyperparameter settings on the 10 test data sets. Note that true inne AUC score is obtained with a different sample size for each dataset: $\psi_{true_D} = [10, 174, 10, 10, 10, 143, 10, 75, 10, 23]$. The others are obtained for fixed sample sizes across the data sets: standard inne: $\psi_{standard} = 8$, best mean inne: $\psi_{mean} = 73$, and best median inne $\psi_{median} = 49$

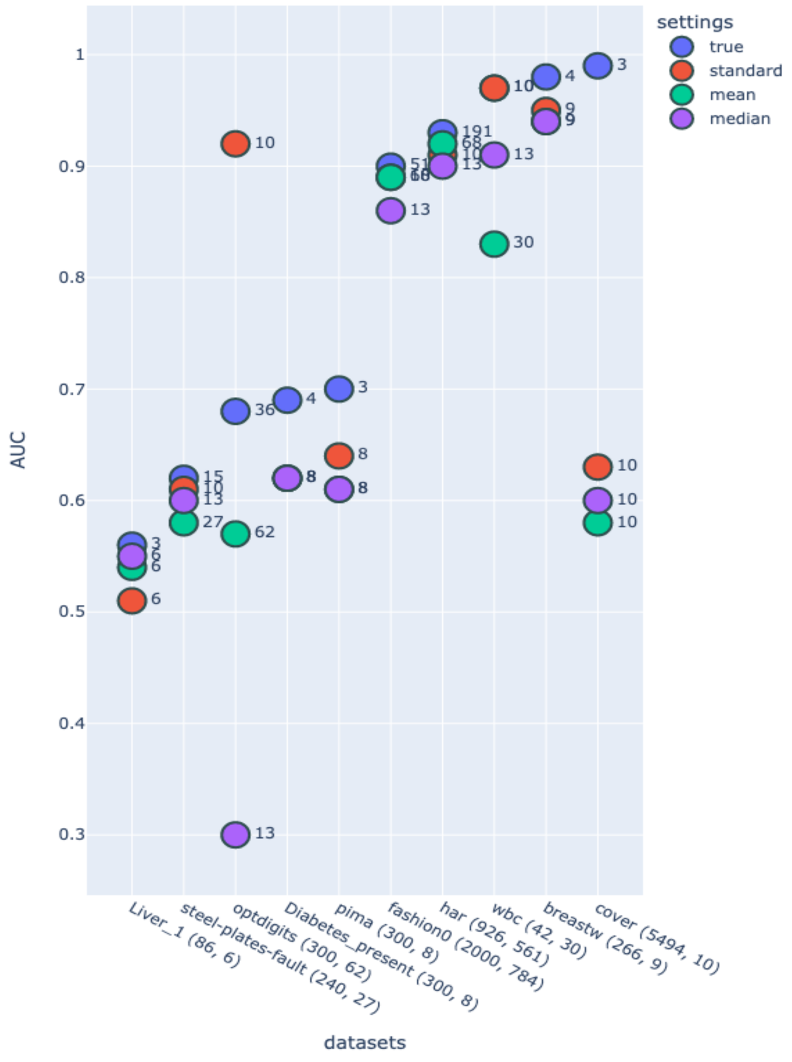


Figure 26. Graph shows AUC scores of DEAN models obtained with 4 different hyperparameter settings. The values on the right of the markers represent the bag parameter values. Note that for high dimensional data sets, both the bag size and the true AUC scores increase. For low dimensional data sets, this correlation becomes negative.

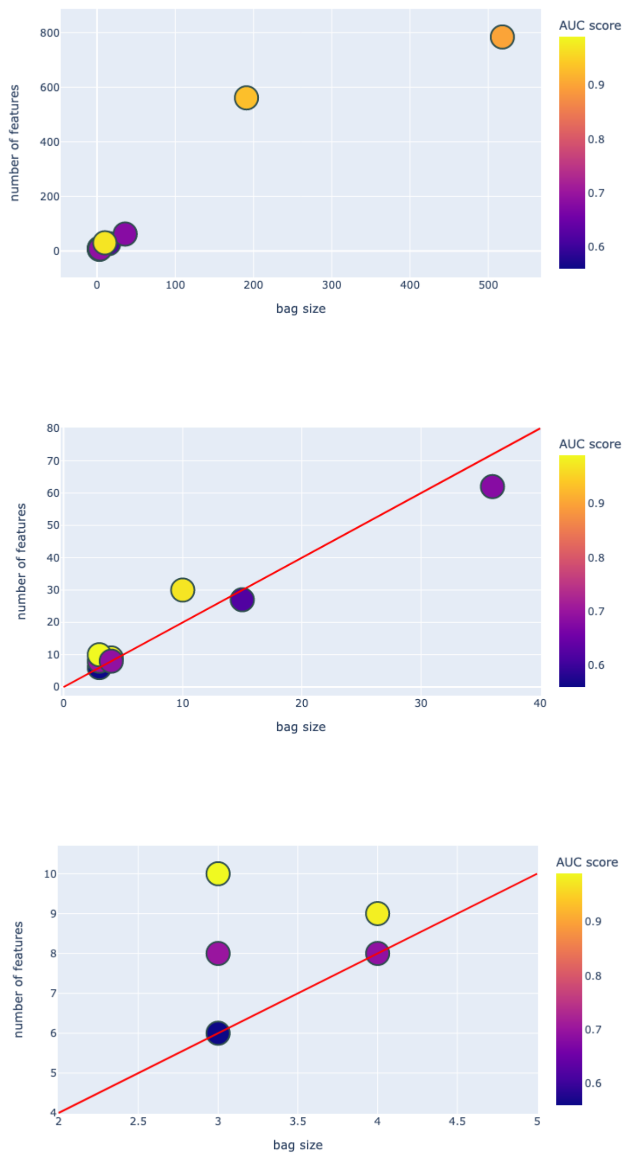


Figure 27. Graph shows the effect of the *bag* over features ratio on the achieved AUC scores of the model. The slope of the red line is 0.5. Note that for the 9 low dimensional data sets, DEAN achieves better AUC scores when the *bag* size is lower than half the number of features in the data set.

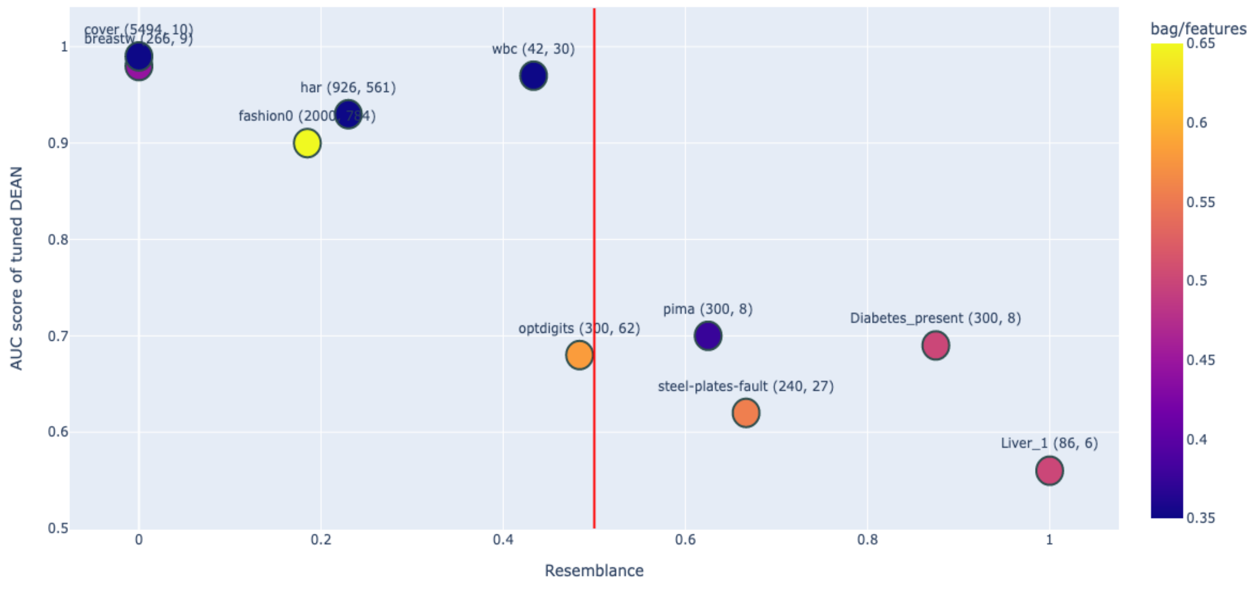


Figure 28. Graph shows the negative correlation between the AUC scores achieved and the computed resemblance between the training and testing sets of our data sets. Note that data sets with the same resemblance and decreasing bag over features largely reduces the AUC scores. Similarly, increasing the resemblance while using the same bag over features ratio also results in lower AUC scores.

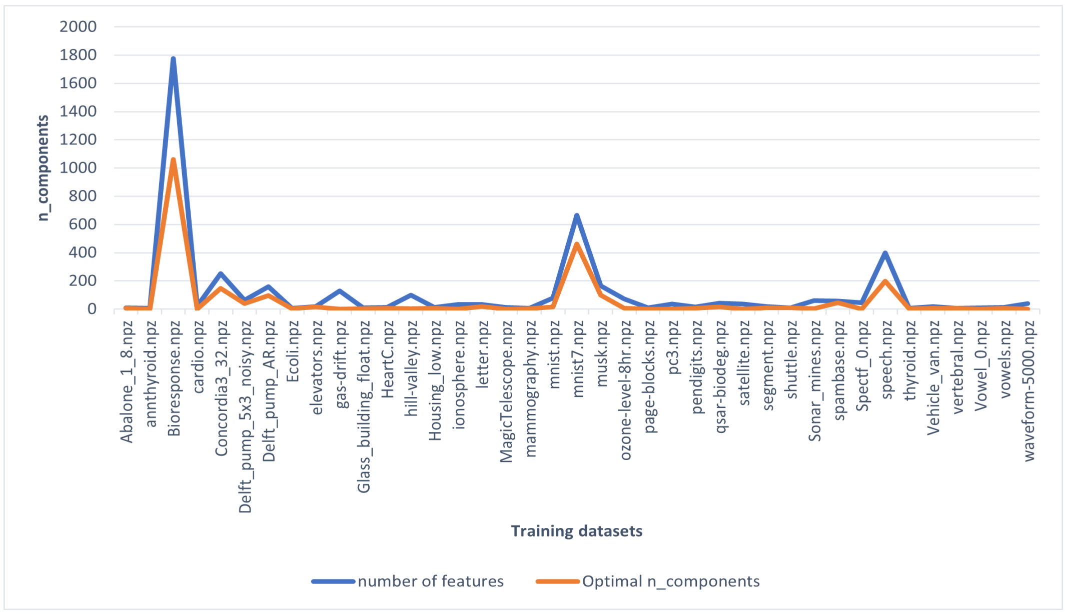


Figure 29. This plot depicts the impact of dimensionality on tuned hyperparameter for corresponding datasets. The blue colored line indicates the number of features present in the data and the yellow line represents the optimal number of components found after tuning.

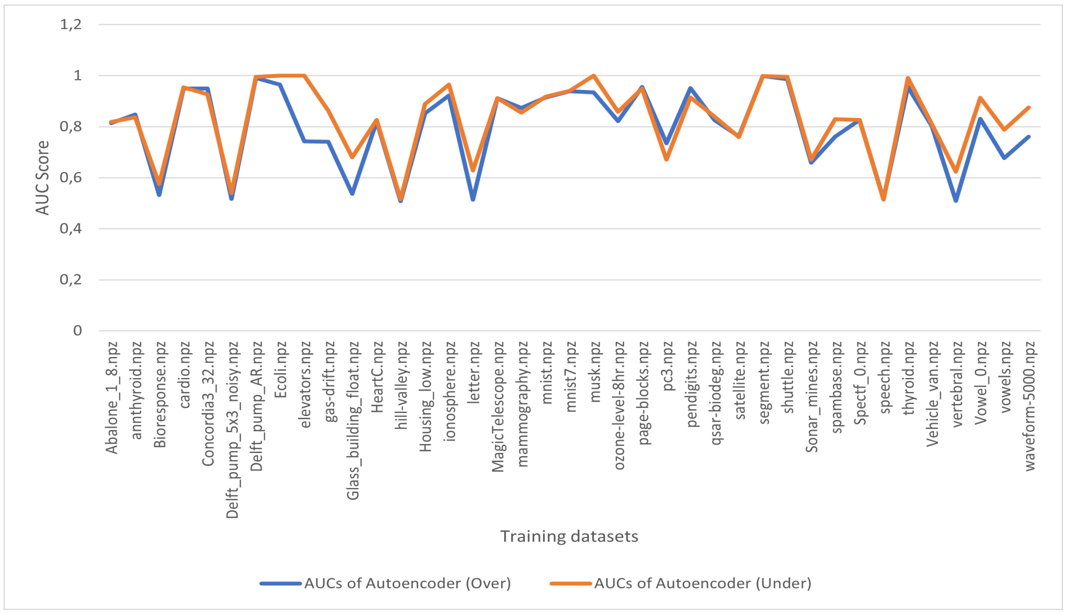


Figure 30. This plot depicts the comparison of AUC scores between autoencoder(under) and autoencoder(over) for 40 training datasets. For most of the datasets, autoencoder (under) performs better than autoencoder (over) with higher AUC scores.

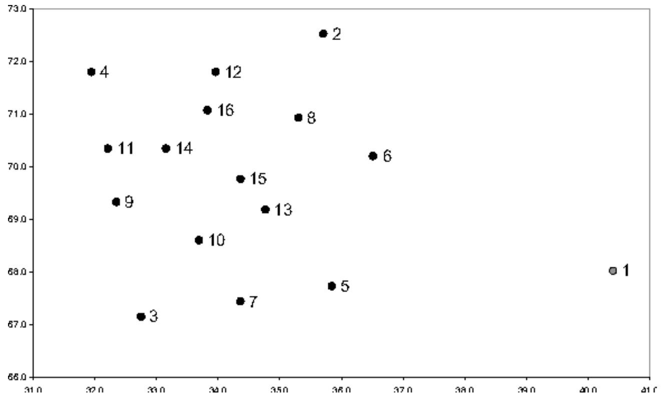


Figure 31. The points ranked according to their angle-based outlier factor (ABOF) (Kriegel et al., 2008)

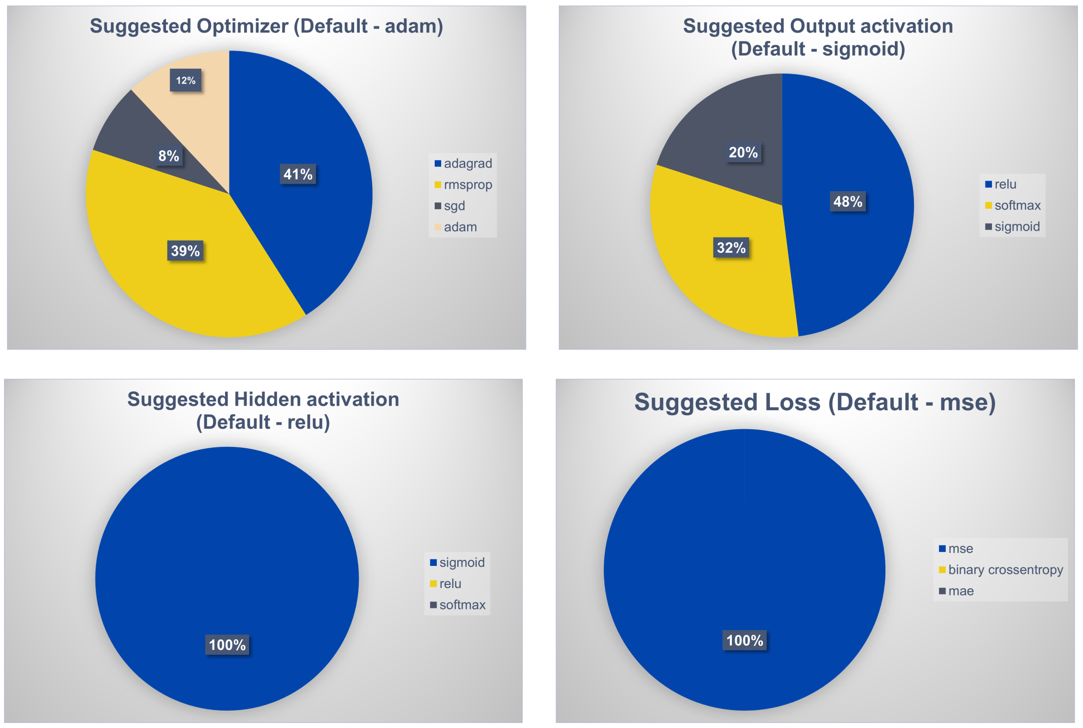
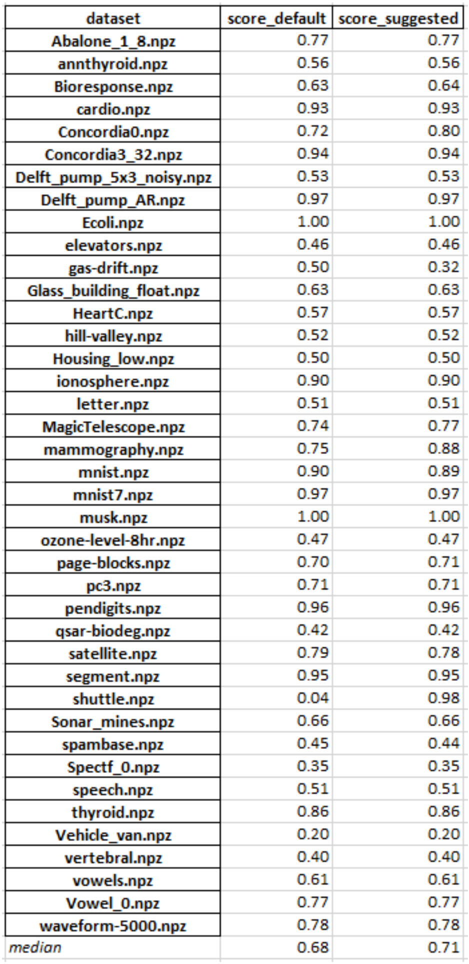
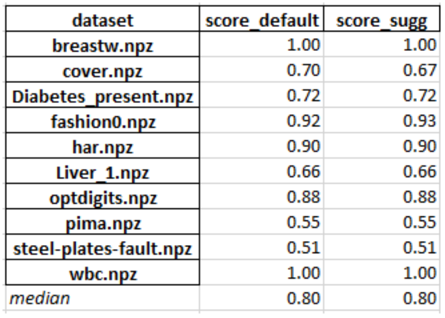


Figure 32. Pie chart showing the provided function to flaml and their proportions in 40 training datasets after flaml tuning



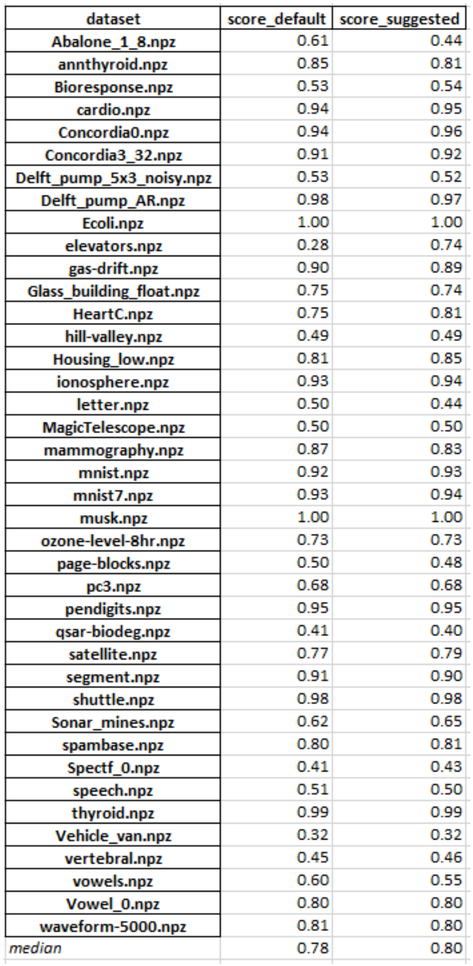
dataset	score_default	score_suggested
Abalone_1_8.npz	0.77	0.77
annthyroid.npz	0.56	0.56
Bioresponse.npz	0.63	0.64
cardio.npz	0.93	0.93
Concordia0.npz	0.72	0.80
Concordia3_32.npz	0.94	0.94
Delft_pump_5x3_noisy.npz	0.53	0.53
Delft_pump_AR.npz	0.97	0.97
Ecoli.npz	1.00	1.00
elevators.npz	0.46	0.46
gas-drift.npz	0.50	0.32
Glass_building_float.npz	0.63	0.63
HeartC.npz	0.57	0.57
hill-valley.npz	0.52	0.52
Housing_low.npz	0.50	0.50
ionosphere.npz	0.90	0.90
letter.npz	0.51	0.51
MagicTelescope.npz	0.74	0.77
mammography.npz	0.75	0.88
mnist.npz	0.90	0.89
mnist7.npz	0.97	0.97
musk.npz	1.00	1.00
ozone-level-8hr.npz	0.47	0.47
page-blocks.npz	0.70	0.71
pc3.npz	0.71	0.71
pendigits.npz	0.96	0.96
qsar-biodeg.npz	0.42	0.42
satellite.npz	0.79	0.78
segment.npz	0.95	0.95
shuttle.npz	0.04	0.98
Sonar_mines.npz	0.66	0.66
spambase.npz	0.45	0.44
Spectf_0.npz	0.35	0.35
speech.npz	0.51	0.51
thyroid.npz	0.86	0.86
Vehicle_van.npz	0.20	0.20
vertebral.npz	0.40	0.40
vowels.npz	0.61	0.61
Vowel_0.npz	0.77	0.77
waveform-5000.npz	0.78	0.78
median	0.68	0.71

Figure 33. KDE AUC scores (default and suggested) for train data



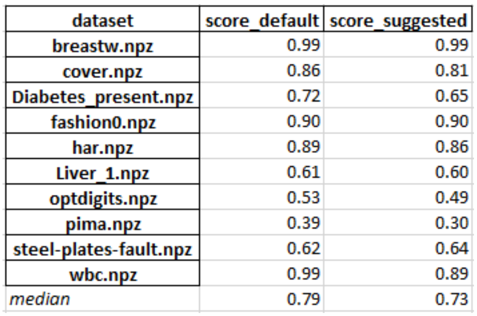
dataset	score_default	score_sugg
breastw.npz	1.00	1.00
cover.npz	0.70	0.67
Diabetes_present.npz	0.72	0.72
fashion0.npz	0.92	0.93
har.npz	0.90	0.90
Liver_1.npz	0.66	0.66
optdigits.npz	0.88	0.88
pima.npz	0.55	0.55
steel-plates-fault.npz	0.51	0.51
wbc.npz	1.00	1.00
median	0.80	0.80

Figure 34. KDE AUC scores (default and suggested) for test data



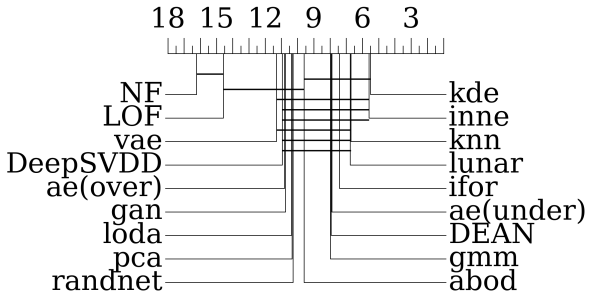
dataset	score_default	score_suggested
Abalone_1_8.npz	0.61	0.44
annthyroid.npz	0.85	0.81
Bioresponse.npz	0.53	0.54
cardio.npz	0.94	0.95
Concordia0.npz	0.94	0.96
Concordia3_32.npz	0.91	0.92
Delft_pump_5x3_noisy.npz	0.53	0.52
Delft_pump_AR.npz	0.98	0.97
Ecoli.npz	1.00	1.00
elevators.npz	0.28	0.74
gas-drift.npz	0.90	0.89
Glass_building_float.npz	0.75	0.74
HeartC.npz	0.75	0.81
hill-valley.npz	0.49	0.49
Housing_low.npz	0.81	0.85
ionosphere.npz	0.93	0.94
letter.npz	0.50	0.44
MagicTelescope.npz	0.50	0.50
mammography.npz	0.87	0.83
mnist.npz	0.92	0.93
mnist7.npz	0.93	0.94
musk.npz	1.00	1.00
ozone-level-8hr.npz	0.73	0.73
page-blocks.npz	0.50	0.48
pc3.npz	0.68	0.68
pendigits.npz	0.95	0.95
qsar-biodeg.npz	0.41	0.40
satellite.npz	0.77	0.79
segment.npz	0.91	0.90
shuttle.npz	0.98	0.98
Sonar_mines.npz	0.62	0.65
spambase.npz	0.80	0.81
Spectf_0.npz	0.41	0.43
speech.npz	0.51	0.50
thyroid.npz	0.99	0.99
Vehicle_van.npz	0.32	0.32
vertebral.npz	0.45	0.46
vowels.npz	0.60	0.55
Vowel_0.npz	0.80	0.80
waveform-5000.npz	0.81	0.80
median	0.78	0.80

Figure 35. AE AUC scores (default and suggested) for test data

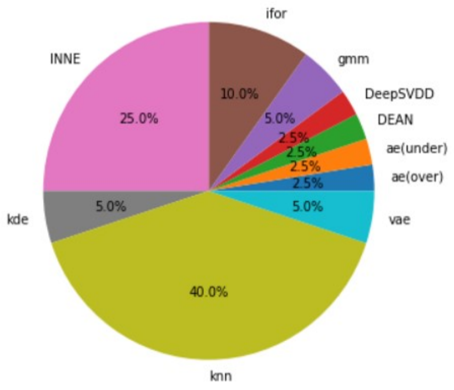


dataset	score_default	score_suggested
breastw.npz	0.99	0.99
cover.npz	0.86	0.81
Diabetes_present.npz	0.72	0.65
fashion0.npz	0.90	0.90
har.npz	0.89	0.86
Liver_1.npz	0.61	0.60
optdigits.npz	0.53	0.49
pima.npz	0.39	0.30
steel-plates-fault.npz	0.62	0.64
wbc.npz	0.99	0.89
median	0.79	0.73

Figure 36. AE AUC scores (default and suggested) for test data

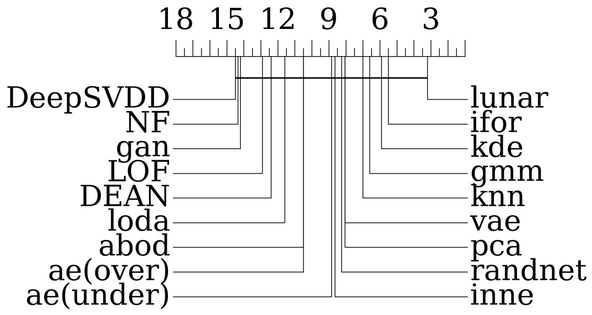


(a) Critical Difference Diagram

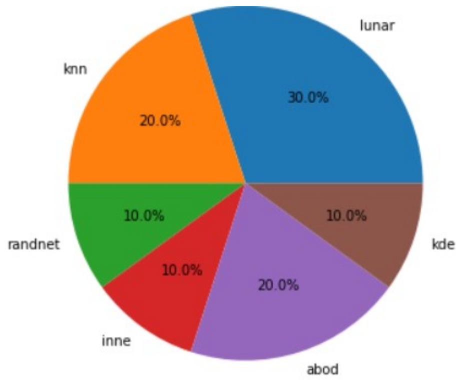


(b) Top Rank Algorithm

Figure 37. Ranking of Algorithms – Tuned Hyperparameters

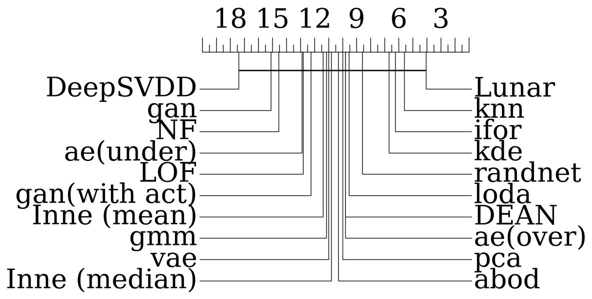


(a) Critical Difference Diagram

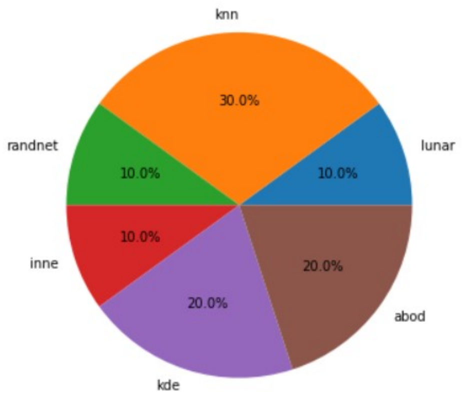


(b) Top Rank Algorithm

Figure 38. Ranking of Algorithms – Default Hyperparameters



(a) Critical Difference Diagram



(b) Top Rank Algorithm

Figure 39. Ranking of Algorithms – Best Hyperparameters

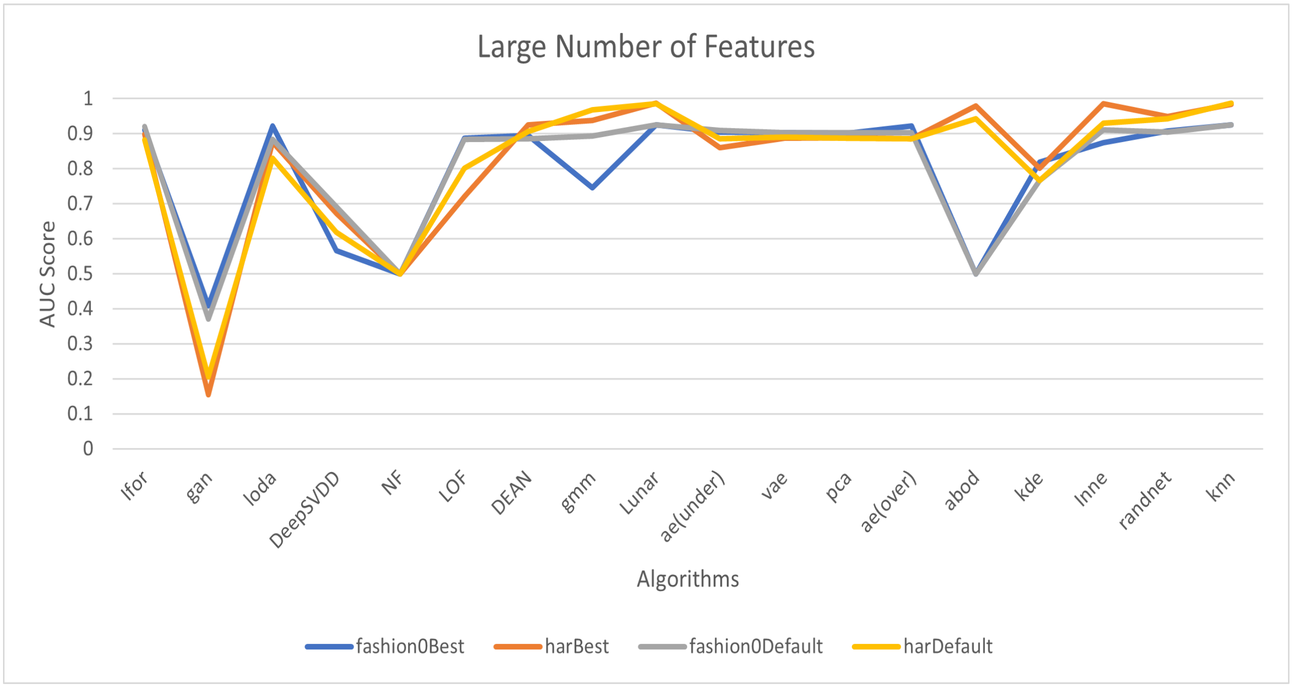


Figure 40. Performance of 18 algorithms in high dimensions

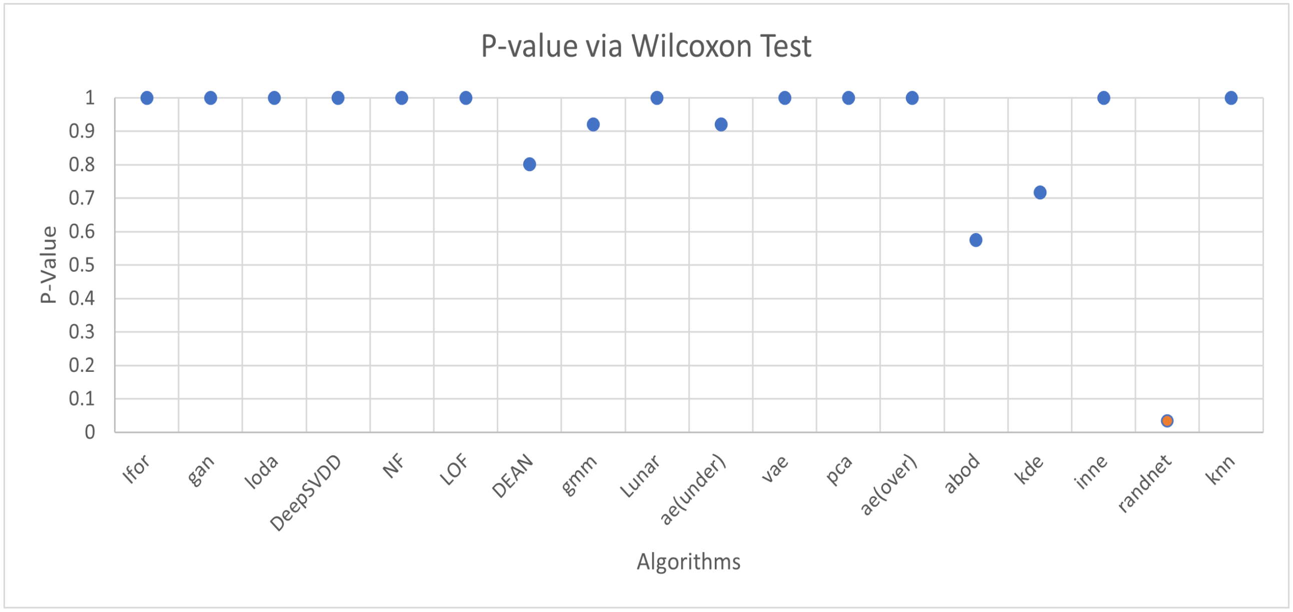


Figure 41. P-value via Wilcoxon test for AUC scores (default and best hyperparameters) for 18 algorithms

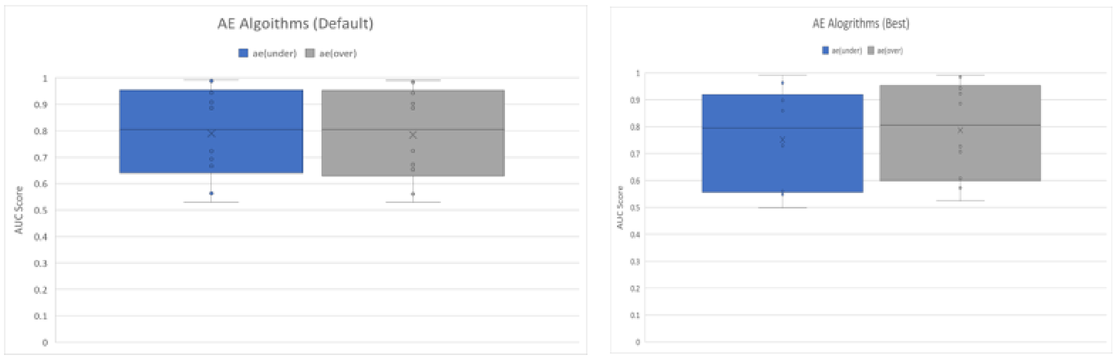


Figure 42. Autoencoders comparison for default and best hyperparameters (Blue: ae under, and Gray: ae over)