Auto-ML and anomaly detection in time series

Abetayeva Anar

Université Cote d'Azur anar.abetayeva@etu.univ-cotedazur.fr

Abstract. This work aims to evaluate AutoML tools for anomaly detection in time series and to analyze the influence of temporal sequence preprocessing before applying AutoML. Moreover, the results, rules and constraints obtained from experiments performed during this research are used as a part of Knowledge Base for anomaly detection in time series, so that pipelines built during experiments are reusable in the future problems with similar context.

Keywords: time series · anomaly detection · AutoML · Feature Model · Deep Learning.

1 Introduction

Building a Machine Learning system is an iterative and time-consuming process. There are a huge number of scenarios during the process, which involves different input data, preprocessing, feature development, feature selection, model selection, and training. There is a combinatorial number of methods that can be used to achieve a goal. However, the number of possible compositions decreases as one chooses a path to follow as any method has its requirements and constraints. Each of the choices can significantly influence the quality of the model produced. This process requires a deep knowledge in data science but also a good understanding of the business problem itself, *i.e.*, the data and the objectives.

Machine learning algorithms work with different kinds of data, one of them is time series, which are sequences of events in time. Time series data are ubiquitous: user behavior on the website, stock prices, video or sound data. Any event that is time-related can be classified as a time series. Anomaly detection in this context is especially challenging as it is sensitive to the domain and context they occur in.

Faced with this complexity, AutoML tools are ready-to-use modeling solutions which, in clearly identified contexts, offer very effective solutions. We propose to provide elements on the identification of these contexts with the aim of detecting anomalies. Here are the 3 questions we sought to answer. *RQ1*: Do the AutoML algorithms perform correctly in the presence of a low number of anomalies? *RQ2*: Is it possible to improve the performance of AutoML, by preparing the data before the learning phase? *RQ3*: Could we derive requirements for a more efficient use of AutoML, by explaining its behavior? We are building a system that captures a non exhaustive set of properties and constraints on building a time series anomaly detection workflow using AutoML tools. Since

AutoML tools are out-of-the-box solutions for modeling, we focus on preparing the data before the modeling phase and formalizing the requirements underlying the use of AutoML approaches. In addition, this work analyzes the influence of the preprocessing to efficiency of AutoML.

The paper consists of six sections. First two sections, Time series (cf. 2) and AutoML(cf. 3) , present description of the object of study along with review of the state of the art in the matter. Next section, Experiment Design, dedicated to explain conditions of the experiments. In the next section experiments results and obtained insights are presented followed by conclusion, which includes proposed future extension of the project.

2 Time Series

Time Series is a sequence of records (events) in time. Analysis of time series is ubiquitous including areas such as marketing, finance, engineering, IoT data and meteorological readings [9]. Experts in these areas are able to identify problems, anticipate measures and make better decisions by investigating how certain phenomena behave over time and predicting their evolution.

There are two main areas in Time Series Analysis: forecasting - entails predicting future behaviour based on past events, anomaly detection - specializes in identifying irregular observations within a given dataset [9]. Moreover, we can use Statistical and Machine Learning approaches for both forecasting and anomaly detection. Machine Learning techniques highly depend on their configuration, which, in turn, is defined by data fitted to them. Thus, knowing the domain is key to choosing the most suitable approach given the data and tuning the model's hyperparameters.

In the data analysis domain, anomaly detection solves the problem of discriminating anomalous or abnormal data within a given dataset [1]. Anomalies are patterns/samples in data that are not in accordance with expected or normal patterns/samples. Detecting anomalies is critical in data analysis because often processes or behavior generated them are abnormal and can cause problems within an object of study, e.g credit card fraud, cyber-attacks and visual anomalies in an MRI image. Thus it is not only a task of detecting an anomaly but also being able to understand its nature.

There are three main categories of anomalies: point anomalies, contextual anomalies and collective anomalies [1]. *Point anomaly* occurs when a specific sample (point) in data is substantially different from others. Sometimes these kinds of anomalies are called outliers. *Contextual anomaly* is abnormal with respect to the context, it is conditional. *Collective anomaly* is observed when some group of samples (points) behave abnormally compared to the rest of the dataset.

Further, we will discuss the state of the art techniques in anomaly detection both statistical and machine learning with some brief descriptions of an approach.

2.1 Statistical methods for anomaly detection in time series

Statistical methods for anomaly detection are usually based on assuming some underlying data distribution, i.e. normal, and comparing each data sample to the empirical expectation. The distribution assumption often relies on the central limit theorem. One of the proposed techniques applies the chi-square technique for anomaly detection [15]. This technique measures a distance of a sample from the empirical expectation using the chi-square test:

$$X^2 = \sum_{i=1}^{n} \frac{X_i - E_i}{E_i} \tag{1}$$

where X_i is the observed value, E_i is the expected value and n is a number of variables. Here the high value of a chi-square indicates an anomaly. We define high value using $\mu \pm 3\sigma$, so that if X^2 of a sample is greater than $\overline{X^2} + 3S_X^2$, where $\overline{X}$ is a mean and S is a standard deviation, it is considered abnormal.

There are some more sophisticated statistical methods including Principal Component Analysis (PCA) and Mixture Models for anomaly detection. A Novel Anomaly Detection Scheme Based on Principal Component Classifier is presented in [12]. In this work, the authors developed a predictive model based on robust PCA. This model applies two functions of principal components obtained from normal samples, first one includes major principal components that explain more than 50% of the variance within data, second is minor components with eigenvalues less than 0.20.

In [10], Gaussian Mixture Model (GMM) based anomaly detection technique for Hyperspectral Imagery (HSI) is introduced. They developed an approach to retrieve anomalous pixels within an image using the GMM-based anomaly extraction method, further they mixed retrieved results by applying the GMM-based weighting method to design an anomaly detection map, this map is also filtered to maintain the final detection map.

2.2 Machine Learning for anomaly detection in time series

Machine learning methods are the main focus of this project. An anomaly detection task can be solved using both unsupervised and supervised machine learning algorithms. Here we will discuss some state-of-the-art techniques both unsupervised and supervised.

One Class SVM Support Vector Machine (SVM) is a widely used machine learning technique that maps input features into a higher dimension so that samples from different classes are separable by a hyperplane in this higher-dimensional space. One-Class SVM (OCSVM), an unsupervised variation of SVM, is frequently used in anomaly detection problems. While in a classical SVM task a hyperplane separating two classes is chosen to maximize a margin, OCSVM tries to separate the entire data set from the origin [13]. To find the hyperplane that separates the training set from the origin we need to solve a quadratic equation that penalizes samples not separated from the origin and maximizes the distance of the hyperplane from the origin. During inference (prediction), this hyperplane acts as a decision function as those points which are separated from the origin are considered normal, and those on the other side of the hyperplane are classified as anomalous. In [13], initial input data was mapped to new feature space using a radial basis kernel.

This method is similar to SVM for several reasons [13]. It exploits the kernel function to perform implicit mappings and dot products, a similar kind of hyperplane is used to define the decision function, and the algorithm only relies on support vectors to solve the problem. However, these support vectors are defined differently. Since data is initially unlabeled, the algorithm assigns the +1 class to data points that are in the densest region, while other points are labeled as -1 class.

Autoencoders Another unsupervised method used for anomaly detection is autoencoders, but unlike OCSVM, autoencoders use deep learning. Generally in machine learning and data mining techniques for anomaly detection, the goal is to obtain a model that represents normal behavior on training data. Once we have the model we can check if the test data is consistent with the model, if not it is considered to be anomalous [11].

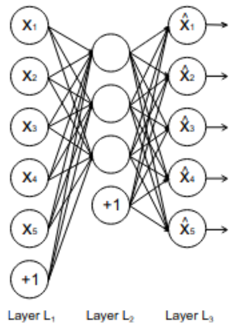


Fig. 1: Autoencoder [11]

Figure 1 depicts an autoencoder neural network architecture that is trained to reconstruct input vectors $\{x_1, x_2, ..., x_m\}$ as outputs $\{\hat{x}_1, \hat{x}_2, ..., \hat{x}_m\}$ in an unsupervised manner. In this figure, a hidden layer, L2, represents a compressed version of the inputs. In the work done by [11], authors used sigmoid activation for a hidden layer and linear function for an output layer because autoencoders do not require pre-scaling to a specific interval.

The typical objective function for autoencoders is Mean Squared Error loss, which was employed in the [11] as wee, but with regularization:

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}||\mathbf{x}(i) - \hat{\mathbf{x}}(i)||^2) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^{(l)})^2 \tag{2}$$

where n_l is a number of layers in the network and s_l denotes number of units in layer L_l.

RNN Recurrent Neural Networks (RNN) with Long Short Term Memory (LSTM) architecture are useful for learning sequential data by maintaining long term memory that allows important information to persist [8]. These networks also enable comprehending higher level temporal features for more efficient learning with sparse representations by stacking multiple recurrent hidden layers. In [8], this kind of network was applied for anomaly detection in time series, which is justified due to the sequential nature of time series.

Detecting changes in the underlying distribution of a sequential process traditionally involves calculating statistics such as cumulative sum (CUMSUM) and exponentially weighted moving average (EWMA) over a predefined time window [8]. As the length of the window needs to be set the results are highly influenced by this initial configuration. In contrast, LSTM can learn long term correlations in a sequence and model complex multivariate sequences without a predetermined time window. Long term memory is maintained in LSTM by the input (I_G), output (O_G) and forget (F_G) multiplicative gates that are responsible for sustaining relevant information and removing irrelevant one (Figure 2). In [8] it was shown that stacked LSTM networks can learn normal behavior of a time series, which is further can be used to detect alterations from this normal behavior without predefined time window and preprocessing. Moreover, by employing multiplicative gates LSTM has overcome a vanishing gradient problem present in classic RNNs.

As it was mentioned before, LSTM is natural for sequential data. Thus, during the experimental part of the project, a single layer LSTM network without any preprocessing was chosen as a baseline.

3 Auto ML

In recent years, automating machine learning tools is a popular research and development area motivated by multiple reasons. First, now we have enough computational capabilities to employ trial-and-error methods to select the best models along with the best hyperparameters given a dataset. Next, we have enough data to be able to train, validate and test multiple hypotheses. Moreover, we observe the boost in the industrial application of Data Science and Machine Learning, where stakeholders want to improve their users' experiences and reduce time to market. This is why there are several emerging Auto ML tools available today. One of the main goals of this paper is to apply these tools for anomaly detection in time series in order to compare them to conventional methods and define the necessary preparation workflow for time series data to be fitted to the

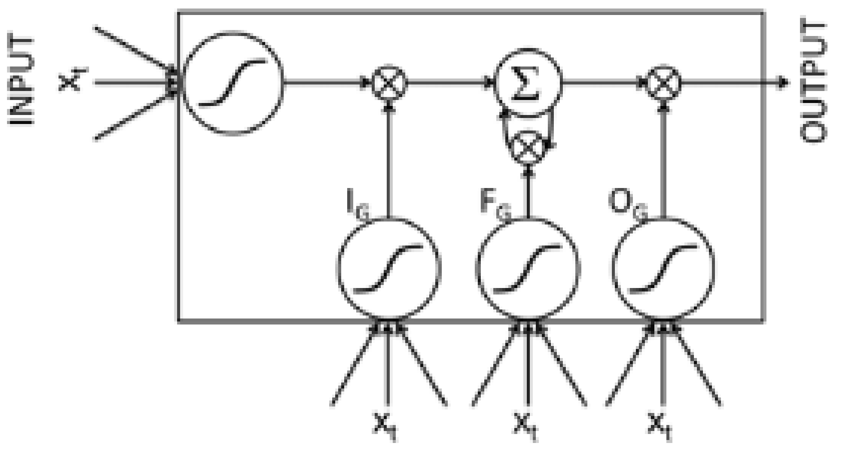


Fig. 2: Long short-term memory cell [8]

black-box Auto ML tool. In Section 5, we perform qualitative and quantitative analysis of both conventional and Auto ML techniques. We tested Auto-PyTorch and Auto-Keras, and in following paragraphs we present descriptions of those techniques.

3.1 Auto-PyTorch Tabular Classification

Deep learning (DL) is widely used due to its outstanding performance and functionality to retrieve useful data representations without human intervention, but initially most of the AutoML frameworks were focused on reproducing traditional machine learning [17]. The optimal choice of neural architecture and hyperparameters affects DL algorithm's performance and varies between tasks and datasets. Moreover since the best architecture and best hyperparameter configuration for it are interrelated, AutoDL systems have to optimize them in conjunction. To tackle this task researchers introduced neural architecture search (NAS) methods.

DL techniques achieve strong performance, but training them is more expensive compared to traditional ML pipelines [17]. Thus traditional AutoML approaches such as blackbox Bayesian Optimization, evolutionary strategies or reinforcement learning do not scale to DL models. Fortunately, the full optimization problem can be approximated by proxy tasks on cheaper fidelities, e.g. training only for a few epochs. This approach is referred to as multi-fidelity optimization and is exploited in Auto-Pytorch to build a robust AutoDL system with anytime performance.

Auto-PyTorch Tabular implements multi-fidelity optimization and jointly searches for architectural parameters and training hyperparameters of a neural network [17]. As the name suggests it is designated for tabular data as authors believed that DL hasn't shown its full potential on applications that feature tabular data. The framework is the successor of Auto-Net, and besides multi-fidelity optimization it exploits ensemble learning and meta-learning for a data-driven selection of initial configurations for warm-starting Bayesian optimization [5, 6, 16].

3.2 Auto Keras ImageClassification

NAS time complexity is proportional to number of neural architectures in a search space (n) and the average time consumption ($\hat{t}$) for evaluating those n neural networks, so that it is $O(n\hat{t})$ time [7]. To speed up the process of search we can make use of network morphism, which morphs the architecture but keeps its functionality. By the use of network morphism it is possible to modify an already trained network into a new architecture, then it requires only a few more epochs to further train a new architecture. For example, it is possible to obtain new architecture by inserting new layers or adding skip-connection to an existing model. This technique decreases the average training time $\hat{t}$ during a NAS. Further to make the search less time-consuming Bayesian Optimization can be used in guiding the network morphism to reduce the number of trained neural networks n.

In Auto-Keras, network morphism with Bayesian optimization to select the most promising optimization is implemented to obtain an efficient neural architecture search [7]. Besides, the framework maintains an edit-distance neural network to measure the number of operations to transform one neural network into another. In addition, Auto-Keras has a built-in acquisition function optimizer for trading-off between exploration and exploitation, which was designed to comply with the tree-structure search space for Bayesian optimization.The changes in the neural architectures introduced by layer-level network morphism addressed in a designated graph-level network morphism. Finally, the framework enables parallel CPU and GPU workloads; some memory adaptation strategies were designed to address the issue of different GPU memory.

For the experimentation we have chosen to apply Auto-Keras ImageClassifier API, which works with images. Considering that our data is time series, we need to apply transformations on it to be able to fit it to Auto-Keras. Further we describe what kind of transformations can be made to turn time series into images.

Transforming time series to images The idea of transforming time series into images is inspired by the success of deep learning in computer vision [14]. The main challenge that time series encoding algorithms solve is making time series "visually" recognized and classified by machines. There are two methods that we used in this project: Gramian Angular Field (GAF), where time series is represented in polar coordinates instead of Cartesian coordinates, Markov Transition Field (MTF), which is inspired by duality between time series and complex networks.

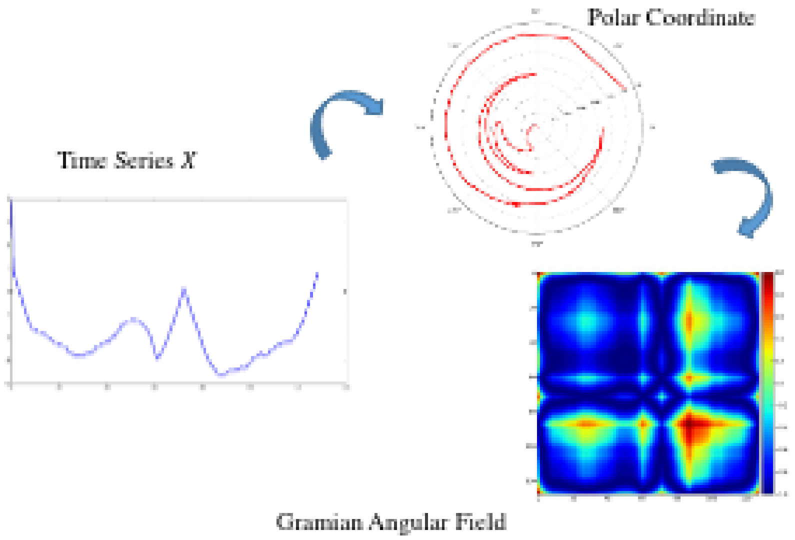


Fig. 3: Gramian Angular Field [14]

Gramian Angular Field Figure 3 illustrates an example of a GAF encoding map [14]. $X = x_1, x_2, \ldots, x_n$ is a time series of n real-valued variables. The first step involves re-scaling X to fall in the interval [-1, 1]:

$$\tilde{x}_i = \frac{(x_i - max(X)) + (x_i - min(X))}{max(X) - min(X)} \tag{3}$$

Next it is transformed into polar coordinate system:

$$\begin{cases} \phi = arccos(\tilde{x}_i), -1 \leq \tilde{x}_i \leq 1, \tilde{x}_i \in \tilde{X} \\ r = \frac{t_i}{N}, t_i \in N \end{cases} . \tag{4}$$

where t_i is a time and N is a constant factor to regularize the span of the polar coordinates. Now we can calculate GAF with:

$$G = \begin{bmatrix} cos(\phi_1 + \phi_1) & \ldots & cos(\phi_1 + \phi_n) \\ cos(\phi_2 + \phi_1) & \ldots & cos(\phi_2 + \phi_n) \\ \ldots & \ldots & \ldots \\ cos(\phi_n + \phi_1) & \ldots & cos(\phi_n + \phi_n) \end{bmatrix} = \tilde{X}'\tilde{X} - \sqrt{I - \tilde{X}^2}' \sqrt{I - \tilde{X}^2}$$

Advantages of GAF are perseverance of a temporal dependence and temporal correlation, so that using deep learning time series can be approximately reconstructed from GAF [14]. However, the GAF matrix is large $n \times n$, where n is the length of a time series. Thus it was proposed to use Piecewise Aggregation Approximation to reduce sequence length and smooth the time series while keeping trends.

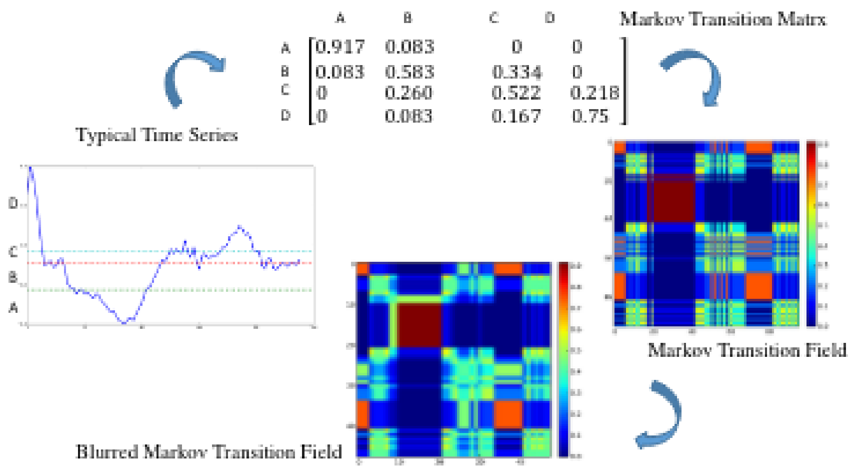


Fig. 4: Markov Transition Field [14]

Markov Transition Field Figure 4 illustrates an example of a MTF encoding map [14]. $X = \{x_1, x_2, \ldots, x_n\}$ is a time series of n real-valued variables. Initially X is discretized into Q quantile bins, based on which Markov Transition Matrix W is calculated. MTF is built using:

$$M = \begin{bmatrix} w_{ij|x_1 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_1 \in q_i, x_n \in q_j} \\ w_{ij|x_2 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_2 \in q_i, x_n \in q_j} \\ \cdots & \cdots & \cdots \\ w_{ij|x_n \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_n \in q_i, x_n \in q_j} \end{bmatrix}$$

MTF identifies relationships between two arbitrary points in time series by evaluating how frequent they appear together in the time series.

4 Experiment Design

An experiment that we held had multiple objectives covering multiple domains. From the Data Science perspective, we wanted to test how reducing the number of anomalies in a data set affects the performance of classifiers. Moreover, we aimed to benchmark AutoML classifiers against the performance of a classifier written from scratch (RQ1). In the AutoML domain, we tested how preprocessing might influence results (RQ2). Finally, all of the experience and knowledge gained during the experiment was formalized in the Feature Model [2] (RQ3).

4.1 Data sets

To perform the experiment we used UCR Time Series Classification Archive [3]. For this experiment, we selected data sets with binary target variables as we reproduced an anomaly detection problem, which usually consists of two classes. Table 1 shows data sets that were selected for the experiment along with their descriptions.

Table 1: Description of data sets [4]

Name	**Data creator**	**Number of classes**	**Size of training set**	**Size of testing set**	**Time series Length**
Wafer	Olszewski	2	1000	6174	152
Yoga	Xi	2	300	3000	426
ToeSegmentation1	Tony Bagnall	2	40	228	277
Wine	Tony Bagnall	2	57	54	234

Data sets are already normalized with mean=0 and variance=1. For the experiment, we selected 4 datasets and used training and testing splits suggested by UCR. Further we reduced the number of a minority class (anomaly) in a data set and splitted testing set into validation and testing sets.

Figure 5 illustrates how a target variable is distributed across datasets. We can see that the number of normal samples within sets doesn't change, while the number of anomaly samples is reduced gradually. Parameter "global_perc" denotes a fraction of initial anomalies used for training, validation and testing. We started with a target variable containing all the anomaly samples and then reduced the size; fractions of the anomalous class used are 1.00, 0.75, 0,50, 0.25 and 0.1. So for each data set 5 classifiers with different number of anomalies were trained, size of the normal class remained the same. More detailed information on class distribution is provided in Appendix A.

4.2 Classification

The experiment included three classifiers: LSTM, TabularClassifier from Auto-PyTorch and ImageClassifier from Auto-Keras. We can further divide them into two groups: classical classifier, which in our case is Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) architecture, and AutoML models, which are TabularClassifier and ImageClassifier. Table 2 demonstrates some initial configurations of aforementioned classifiers.

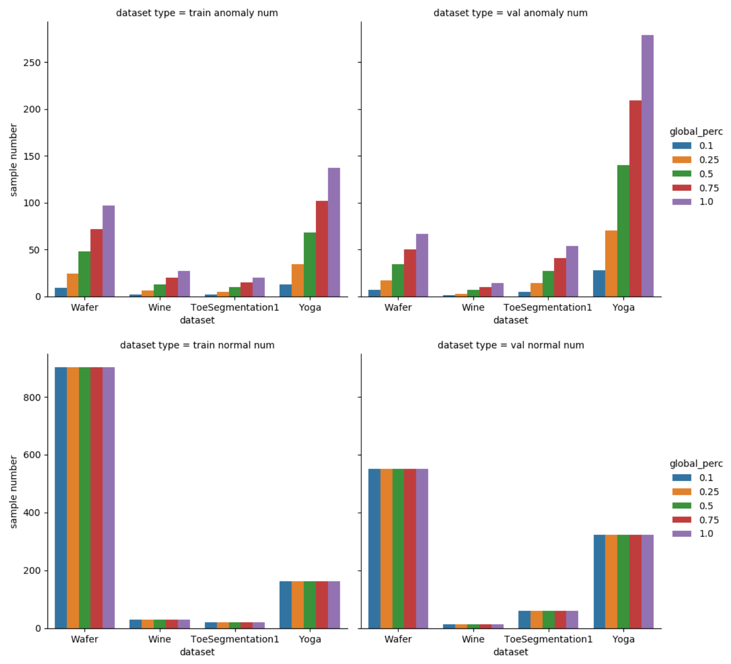


Fig. 5: Distribution of target variable in training, validation and test sets.

Table 2: Characteristics of the training phase

	LSTM	**TabularClassifier**	**ImageClassifier**
Data preparation	none	none	transform to 2D
Input size	n x seq_len x 1	n x seq_len	n x height x width
Model Selection	none	optimize ensemble size versus f1 score	none

RNN in general is used to model data that possesses contextual and temporal dependence. In order to handle this kind of data RNN maintains loops within them to persist the important information. LSTM, on other hand, is able to detect long-term dependencies. This property is important in our case as sequence length in our data sets can reach around 500. Thus the choice of LSTM is motivated by the sequential nature of data sets and possible long-term dependence. To train the model no preprocessing was performed, besides reshaping the dataset by adding extra dimension, which is a requirement of deep learning framework, keras, used to train the model. We built a one layer LSTM network with binary cross entropy loss, adam optimizer and we tuned it against AUC metric. Neural Network was trained for 10 epochs.

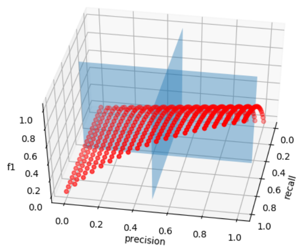
TabularClassifier is an API provided by Auto-PyTorch to train structural (tabular) data. Auto-PyTorch is intended both to optimize traditional ML pipelines and their hyperparameters and to perform neural architecture search [17]. This tool performs multi-fidelity optimization with portfolio construction for warm-starting, and ensembles deep neural networks (DNNs) with classic tabular data classifiers. The size of the ensemble is a parameter that we tuned with respect to performance. We tried ensembles of size 1, 5, 10, 20, 25 and 50 (maximum size). It is important to note that by using this classifier we assume that data has no sequential and temporal dependence, which is simplification in our case. In the results section we will see how compromising sequential dependence affects classifier performance.

Last type of classifier we trained is ImageClassifier from Auto-Keras. Auto-Keras is a framework that performs an efficient neural architecture search by employing Bayesian optimization to guide the network morphism [7]. The Auto-Keras API assumes the input is an image, so we need to transform initial data to be able to use it. We used multiple transformation techniques. First, we just reshaped the data set to be two-dimensional. Next, we used techniques to convert time series to image: Gramian Angular Fields and Markov Transition Fields. The search space of AutoKeras consists of 100 neural network architectures, which entails significant computational efforts. During the experiment we were not able to run it on a local machine, it required a server with GPU to perform the search and classification. This knowledge is useful for future experiments, thus we need to formalize it in a Feature Model.

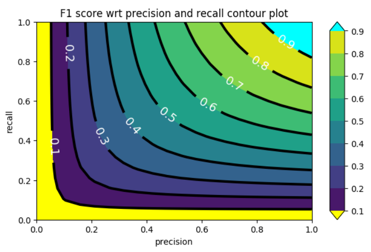
To sum up, experimental settings depend on three main factors: Classification method: LSTM, TabularClassifier, ImageClassifier Data transformation: reshape, Gramian, Markov Distribution of target variable: fraction of anomaly 1.0, 0.75, 0.5, 0.25 and 0.1. The flow chart of the experiment can be found in Appendix B.

5 Experiment Results

To evaluate performance of a pipeline we use f1 score on a test set. F1 score is a statistical measure to assess the accuracy of the binary classifier: $F_1 = 2 \times \frac{precision \times recall}{precision + recall}$



(a) F1 3d.



(b) F1 contour.

Fig. 6: F1 score values against precision and recall.

Precision is the proportion of true positives among all samples classified as positive; recall is the proportion of true positives among samples that are actually positive. In the experiment we label anomalous class as positive, and measure f1 score with respect to this class. Let's define our minimum requirements for a f1 score metric to consider training successful. We assume that the experiment is successful if the f1 score is greater than 0.5, which means that both precision and recall are greater than 0.5. When precision is greater than 0.5 it indicates that more than half of the predictions that classifier predicted as anomaly (class 1) are actually anomalies. When recall is greater than 0.5 it indicates that the classifier predicted as anomaly more than half of actual anomalies that were in the testing data set. Figure 6 illustrates how f1 score changes with respect to precision and recall. In the Figure 6a we can see that if we restrict precision and recall to be greater than 0.5 (shown by hyperplanes), f1 score is always greater than 0.5. More clear representation of the claim can be observed in the Figure 6b.

5.1 Wafer

Distribution of a target Figure 7 shows that for each classifier F1 score increases as the number of anomalies increase in the dataset. This result complies with our initial hypothesis, the more data of each class a classifier receives the better is its ability to discriminate between classes. The variance in model performances with respect to anomaly class size is highest with the LSTM model, which improves significantly with the increase of anomaly samples. In contrast, ImageClassifier with MTF transformation shows very similar performance regardless of the number of anomalies it encounters during training.

AutoML vs Baseline In the case of the Wafer dataset, AutoML classifiers perform significantly better compared to the baseline. Notably, even with only 9 (1%) samples of anomalies seen during the training, ImageClassifier with MTF transformation shows f1-score of 0.97.

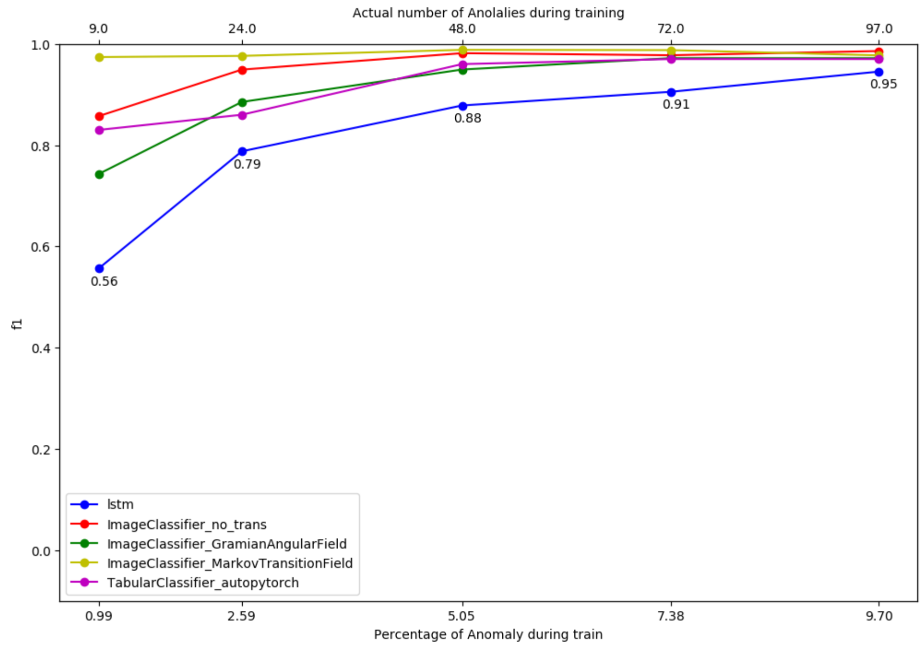


Fig. 7: Wafer f1 score per percentage of anomaly for each classifier.

AutoML and Preprocessing If we analyze results among only AutoML tools, we see that classification without preprocessing is not inferior to those with preprocessing. Specifically, we define the pipeline to be without preprocessing when data was fed into AutoML without any preparation with respect to the nature of underlying data, i.e sequential time dependent data. Auto-PyTorch TabularClassifier and Auto-Keras ImageClassifier with reshaping into 2D grid is considered to be without preprocessing and Auto-Keras ImageClassifier with both GAF and MTF involve preprocessing. This distinction is motivated purely by the fact that with transformation algorithms we take time dependency of the data into account and in other cases we just ignore this factor. Thus in the case of a given data set preprocessing doesn't significantly improve performance as the accuracy achieved without preprocessing is high. The more detailed results are displayed in Appendix C, Table 4.

Rules extracted When we formulate rules here and in upcoming similar sections it should be taken into account that we presume a similar problem and dataset definition. This rules are layman formulation of what will be formalized in a Feature Model. To understand a context we are working in, Figure 8 demonstrates empirical expectation of each observation in a sequence for normal and abnormal instances. We can conclude that anomalous class is very different from normal one, which explains high performance of classifiers even with small amount of anomalies. We can conclude from the set of experiments on the Wafer dataset that one layer LSTM requires more than 9, which correspond to 1%, anomalous samples to perform well. Other pipelines can be added as a valid for a given task with the condition that the number of anomalies is not less than 9 (1%).

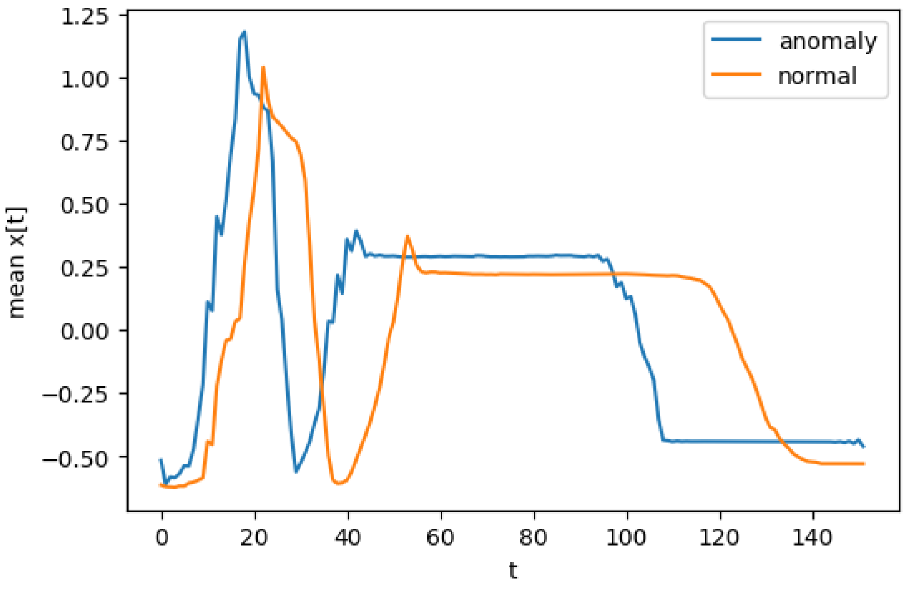


Fig. 8: Wafer mean of abnormal and normal sequences.

5.2 Wine

Distribution of a target Figure 9 demonstrates performance on Wine dataset. Here we see that the results are noticeably inferior to the classifiers trained on the Wafer dataset. It is evident that small size of an anomaly class leads to poor classification results, and most of the pipelines are not able to learn adequate models during training. In the case of TabularClassifier, increasing the number of anomaly classes improves performance.

AutoML vs Baseline Only two classifiers are successful, which are Tabular-Classifier with 30.23% and 47.37% of anomalies in the training set with 0.62 and 0.87 f1 scores respectively. Baseline model is not able to perform classification.

AutoML and Preprocessing Classifier without transformation, TabularClassifier, shows better results than ImageClassifiers with transformations. However,

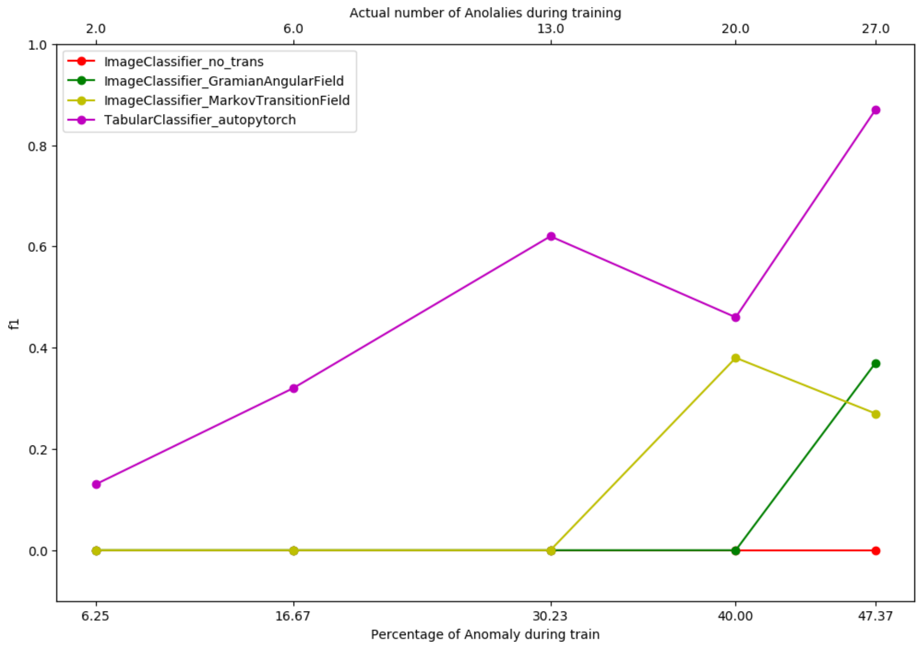


Fig. 9: Wine F1 score per percentage of anomaly for each classifier.

the ImageClassifier with no transformations is not able to classify instances of the Wine dataset even when the anomalous class is not reduced.

Rules extracted In the given problem and input data, TabularClassifier can learn to classify anomalous patterns if there are at least 13 samples (30%). If we look at Figure 10, we can understand the reason for such poor results. Mean vectors of anomalous and normal sequences almost coincide.

5.3 ToeSegmentation1

Distribution of a target In Figure 11 we see that general trend complies with hypothesis and previous experiments, more anomaly samples we have the better a classifier is able to distinguish classes, except for ImageClassifier with Gramian Angular Fields transformation, which shows better performance with 33% of anomalies rather than with 43%. None of the algorithms passess the success criteria for datasets with 9% and 20% of anomalies. The possible reason for this is in those datasets we have 2 and 5 anomalous samples during training, which is very small amount of data for classifier to be able to recognize it. For dataset with 33% anomalies only ImageClassifier with reshaping considered successful with 0.71 f1 score. For 43% anomaly ImageClassifier with no transformations shows superior results with 0.79 f1 score and TabularClassifier shows acceptable

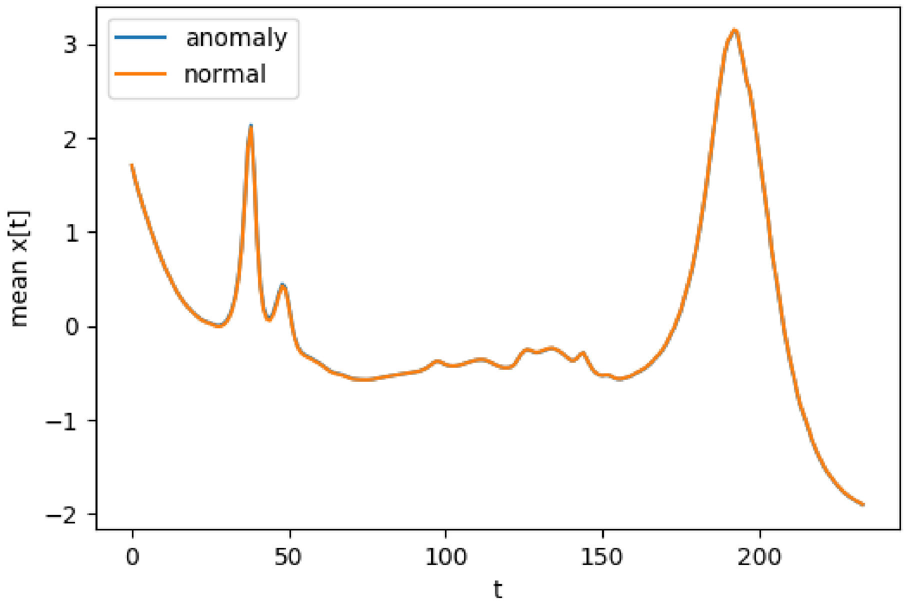


Fig. 10: Wine mean of abnormal and normal sequences.

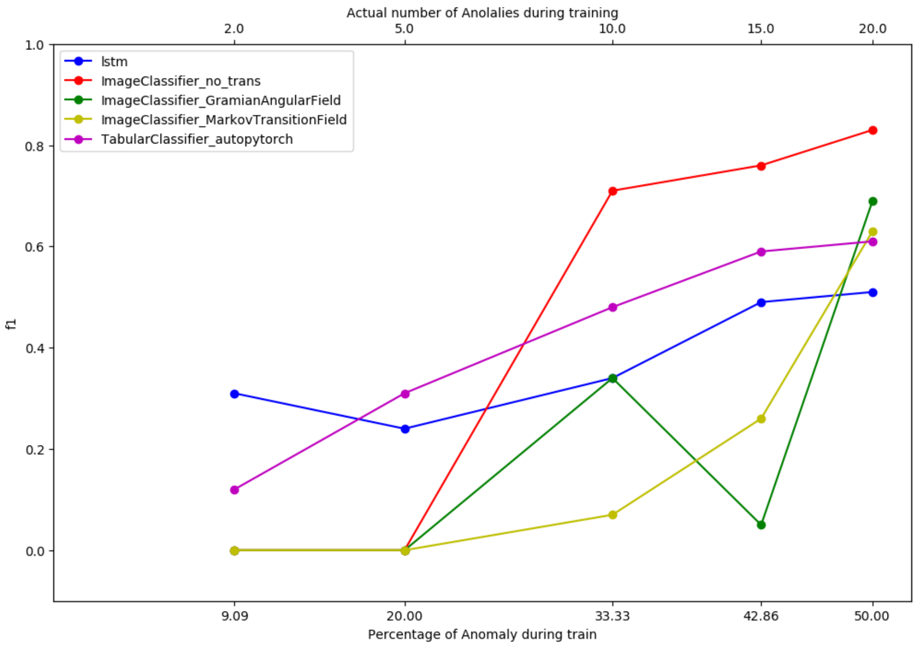


Fig. 11: ToeSegmentation1 F1 score per percentage of anomaly for each classifier.

results with 0.59 f1 score. For 50% of anomaly all algorithms are acceptable, and ImageClassifier with no transformations is better than others with 0.83 f1 score.

AutoML vs Baseline. For ToeSegmentation1 dataset AutoML tools with no preprocessing are in general better than baseline, while AutoML with preprocessing (GAF, MTF) are worse, except when we have a maximum number of anomalies.

AutoML and Preprocessing From the results we can tell that applying ImageClassification with no preprocessing works better in this context given that we have at least 10 anomalous samples (33%) in an overall dataset, and ImageClassifier with preprocessing doesn't tackle the problem well.

Rules extracted For ToeSegmentation1 data set with Auto-PyTorch TabularClassifier, there was a problem when we run algorithm with parameter n_jobs=1, we needed to increase it to n_jobs=4 to perform the classification. We need to use parallelization to allow the AutoML tool to fit the model, we could also increase memory or time limits to provide more resources for fitting. As it was mentioned above in this kind of problem and input data we can use TabularClassifier with no preprocessing only if we have at least 10 samples (33%).

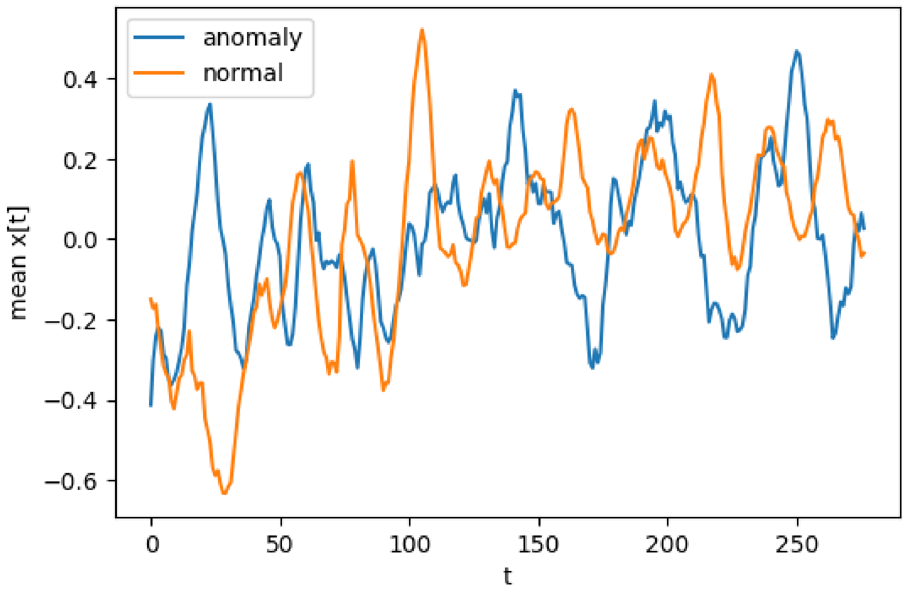


Fig. 12: ToeSegmentation1 means of abnormal and normal sequences.

Figure 12 shows descriptive statistics for anomalous and normal data, from which we can grasp that classes are dissimilar. Thus we can conclude that poor quality of classifiers are mostly because deep learning algorithms require more samples and not due to the nature of underlying data. We can deduce a rule for a given context to try methods, for example statistical, which work good with less data. But this claim needs to be empirically justified.

5.4 Yoga

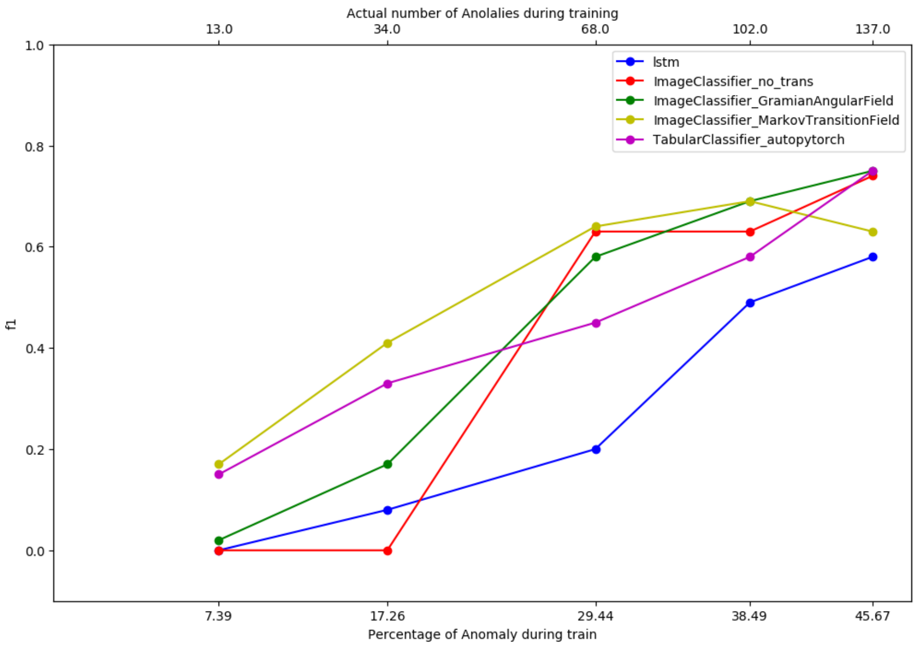


Fig. 13: Yoga F1 score per percentage of anomaly for each classifier.

Distribution of a target Same to previous experiments, the number of anomalies in data affects classifiers performance (Figure 13).

AutoML vs Baseline In this problem, baseline performs worse than AutoML, and demonstrates acceptable results only on a balanced dataset.

AutoML and Preprocessing Both pipelines with and without preprocessing start showing admissible results only when the number of anomalies is sufficient (30%, 68 samples). In general, pipelines with no preprocessing and pipelines with preprocessing perform similarly.

Rules extracted In a given condition, we can use ImageClassifier with no transformation or with transformation (MTF) if number of anomalies is at least around 68 (30%). From the general trend of anomalous versus normal sequence, depicted in Figure 14, we can tell that they are similar, but there are areas where they vary.

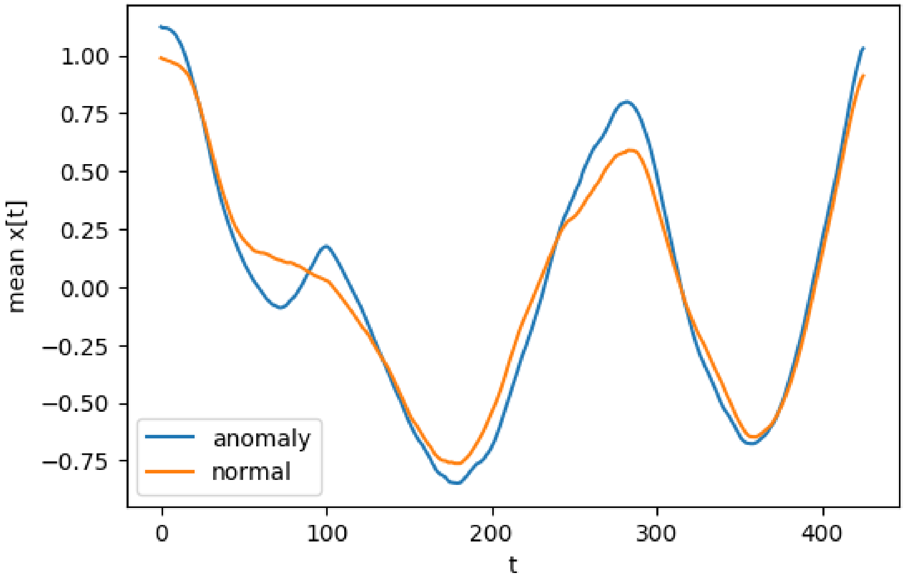


Fig. 14: Yoga means of abnormal and normal sequences.

5.5 Towards knowledge based on observations

To sum up, we can retrieve useful rules from each experiment individually, but we can also derive some common empirical observations. First, the more representatives of an anomalous class a classifier sees during the training the better its performance. Then, AutoML tools perform mostly better than one layer LSTM. Next, AutoML tools without any preprocessing of data are able to show comparable results to pipelines involving preprocessing. Finally, if anomalous sequences on average different from normal AutoML pipelines are able to detect anomalies even with a low number of anomalous samples participated in training. On the other hand, if there are no significant differences between anomalous and normal sequences[1], AutoML tools struggle to detect anomalies unless there are enough representatives of anomalies in the training set.

[1] the mean vectors are the same.

6 Conclusion

This paper has presented theoretical background on anomaly detection in time series along with state of the are techniques in the domain. Anomaly detection in time series is ever developing area of study, and researchers tried to tackle it from many statistical and machine learning perspectives. The goal of this work was to expand the research towards AutoML tools for anomaly detection in time series. The main take away message of the work is that AutoML tools succeed to identify anomalies in time series if 1) anomalies and normal sequences are sufficiently different or 2) there are enough anomalous samples in a data set to train the model irrespective of target variable distribution. To further clarify second point, anomalous samples can be a very small fraction of the total data set, but if there is enough samples in general AutoML tools perform effectively (RQ1). Further, there is no significant advantage in using preprocessing techniques given that previous conditions hold (RQ2). Finally, we can extract useful rules and constraints based on experiments (RQ3). Also, we need take into account that in the real-world anomaly detection tasks it is more difficult to achieve high quality, and labeling data is expensive. Thus, the research contributes to gaining some empirical knowledge on the lower bounds of number of anomalies along with their fraction in a data set needed in order to train a classifier. This insights can be used in the real world problems with careful attention to the nature of an underlying data set and context. In the future work, we suggest to use Dynamic Time Warping (DTW) technique in order to compare data sets and the efficiency of proposed pipelines on the similar data sets. DTW is an algorithm used to compare time series of different sequence lengths. This algorithm can be used to check the reproducibility of the proposed pipelines in the same context but with different data sets. In addition, resources and time consumption of each pipeline needs to be analysed and corresponding rules needs to be defined in the future work.

References

1. Ahmed, M., Naser Mahmood, A., Hu, J.: A survey of network anomaly detection techniques. Journal of Network and Computer Applications **60**, 19–31 (2016). https://doi.org/10.1016/j.jnca.2015.11.016, `http://dx.doi.org/10.1016/j.jnca.2015.11.016`
2. Arcaini, P., Gargantini, A., Vavassori, P.: Generating tests for detecting faults in feature models. 2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings (May) (2015). https://doi.org/10.1109/ICST.2015.7102591
3. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The ucr time series classification archive (July 2015), `www.cs.ucr.edu/~eamonn/time_series_data/`
4. Dau, H.A., Bagnall, A., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Keogh, E.: The UCR time series archive. IEEE/CAA Journal of Automatica Sinica **6**(6), 1293–1305 (2019). https://doi.org/10.1109/JAS.2019.1911747

5. Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F.: Practical Automated Machine Learning for the AutoML Challenge 2018. International Workshop on Automatic Machine Learning at ICML (Section 2), 1189–1232 (2018), https://ml.informatik.uni-freiburg.de/papers/18-AUTOML-AutoChallenge.pdf
6. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing Bayesian hyperparameter optimization via meta-learning. Proceedings of the National Conference on Artificial Intelligence **2**, 1128–1135 (2015)
7. Jin, H., Song, Q., Hu, X.: Auto-keras: An efficient neural architecture search system. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining pp. 1946–1956 (2019). https://doi.org/10.1145/3292500.3330648
8. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2012. ESANN 2012 proceedings, 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (April), 1–650 (2012)
9. Moreira Costa, P.: Time Series Analysis Framework (July) (2019)
10. Qu, J., Du, Q., Li, Y., Tian, L., Xia, H.: Anomaly Detection in Hyperspectral Imagery Based on Gaussian Mixture Model. IEEE Transactions on Geoscience and Remote Sensing **59**(11), 9504–9517 (2021). https://doi.org/10.1109/TGRS.2020.3038722
11. Sakurada, M., Yairi, T.: Anomaly detection using autoencoders with nonlinear dimensionality reduction. ACM International Conference Proceeding Series **02-Decembe**, 4–11 (2014). https://doi.org/10.1145/2689746.2689747
12. Shyu, M.L., Chen, S.C., Sarinnapakorn, K., Chang, L.: A Novel Anomaly Detection Scheme Based on Principal Component Classifier. 3rd IEEE International Conference on Data Mining pp. 353–365 (2003)
13. Stolfo, S.J., Portnoy Leonid, Eskin, E., Andrew, A., Michael, P.: A Geometric framework for unsupervised intrusion detection, vol. 3
14. Wang, Z., Oates, T.: Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. AAAI Workshop - Technical Report **WS-15-14**(May), 40–46 (2015)
15. Ye, N., Chen, Q.: An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. Quality and Reliability Engineering International **17**(2), 105–112 (2001). https://doi.org/10.1002/qre.392
16. Zela, A., Klein, A., Falkner, S., Hutter, F.: Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search pp. 58–65 (2018), http://arxiv.org/abs/1807.06906
17. Zimmer, L., Lindauer, M., Hutter, F.: Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(9), 3079–3090 (2021). https://doi.org/10.1109/TPAMI.2021.3067763

Appendices

Appendix A

Distribution of target variables

Table 3: Number of anomaly and normal classes by data set type for each data set.

dataset	global_perc	train normal num	train anomaly num	val normal num	val anomaly num	test normal num	test anomaly num
Wafer	1.00	903.0	97.0	550.0	67.0	4949.0	598.0
Wafer	0.75	903.0	72.0	550.0	50.0	4949.0	448.0
Wafer	0.50	903.0	48.0	550.0	34.0	4949.0	298.0
Wafer	0.25	903.0	24.0	550.0	17.0	4949.0	149.0
Wafer	0.10	903.0	9.0	550.0	7.0	4949.0	59.0
Wine	1.00	30.0	27.0	14.0	14.0	13.0	13.0
Wine	0.75	30.0	20.0	14.0	10.0	13.0	10.0
Wine	0.50	30.0	13.0	14.0	7.0	13.0	6.0
Wine	0.25	30.0	6.0	14.0	3.0	13.0	3.0
Wine	0.10	30.0	2.0	14.0	1.0	13.0	1.0
ToeSegmentation1	1.00	20.0	20.0	60.0	54.0	60.0	54.0
ToeSegmentation1	0.75	20.0	15.0	60.0	41.0	60.0	40.0
ToeSegmentation1	0.50	20.0	10.0	60.0	27.0	60.0	27.0
ToeSegmentation1	0.25	20.0	5.0	60.0	14.0	60.0	13.0
ToeSegmentation1	0.10	20.0	2.0	60.0	5.0	60.0	5.0
Yoga	1.00	163.0	137.0	322.0	279.0	1285.0	1114.0
Yoga	0.75	163.0	102.0	322.0	209.0	1285.0	835.0
Yoga	0.50	163.0	68.0	322.0	140.0	1285.0	556.0
Yoga	0.25	163.0	34.0	322.0	70.0	1285.0	278.0
Yoga	0.10	163.0	13.0	322.0	28.0	1285.0	111.0

Appendix B

High-level representation of the experiment algorithm.

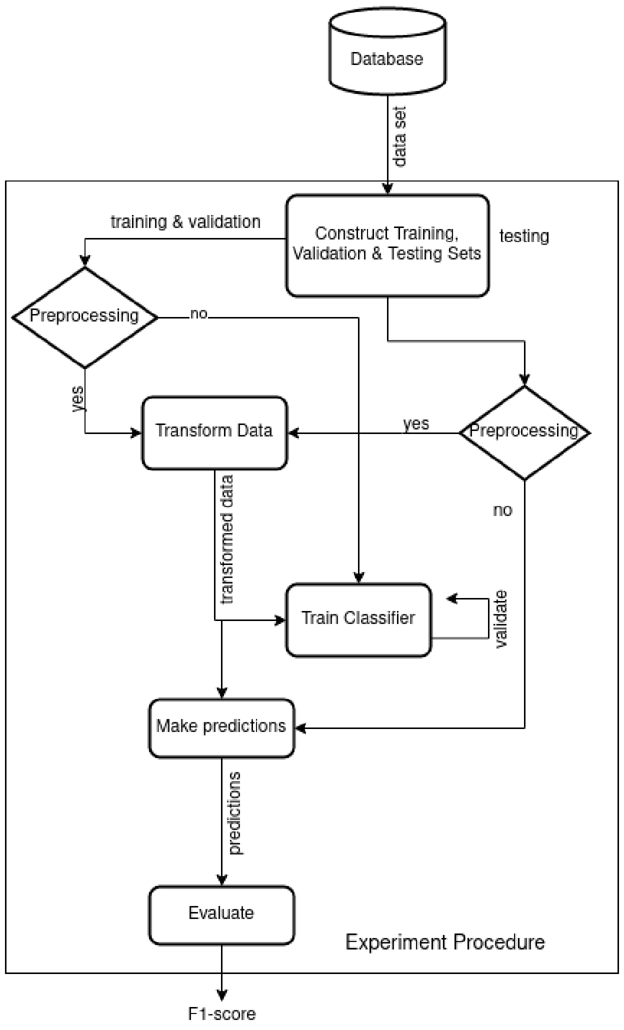


Fig. 15: Experiment flow chart.

Appendix C

F1 scores of the classifiers.

Table 4: Wafer f1 score for each classifier.

	algorithm	**LSTM_keras**	**ImageClassifier_autokeras**			**TabularClassifier_autopytorch**
	preprocessing	**None**	**2D**	**GAF**	**MTF**	**None**
percentage anomaly train	**number anomaly train**					
0.99	**9.0**	0.56	0.86	0.74	0.97	0.83
2.59	**24.0**	0.79	0.95	0.89	0.98	0.86
5.05	**48.0**	0.88	0.98	0.95	0.99	0.96
7.38	**72.0**	0.91	0.98	0.97	0.99	0.97
9.70	**97.0**	0.95	0.99	0.97	0.98	0.97

Table 5: Wine F1 scores for each classifier.

	algorithm	**LSTM_keras**	ImageClassifier_autokeras			**TabularClassifier_autopytorch**
	preprocessing	**None**	**2D**	**GAF**	**MTF**	**None**
percentage anomaly train	**number anomaly train**					
6.25	**2.0**	0.00	0.0	0.00	0.00	0.13
16.67	**6.0**	0.00	0.0	0.00	0.00	0.32
30.23	**13.0**	0.00	0.0	0.00	0.00	0.62
40.00	**20.0**	0.00	0.0	0.00	0.38	0.46
47.37	**27.0**	0.27	0.0	0.37	0.27	0.87

Table 6: ToeSegmentation1 F1 scores for each classifier.

	algorithm	**LSTM_ keras**	ImageClassifier_autokeras			**TabularClassifier_ autopytorch**
	preprocessing	**None**	**2D**	**GAF**	**MTF**	**None**
percentage anomaly train	**number anomaly train**					
9.09	**2.0**	0.31	0.00	0.00	0.00	0.12
20.00	**5.0**	0.24	0.00	0.00	0.00	0.31
33.33	**10.0**	0.34	0.71	0.34	0.07	0.48
42.86	**15.0**	0.49	0.76	0.05	0.26	0.59
50.00	**20.0**	0.51	0.83	0.69	0.63	0.61

Table 7: Yoga F1 scores for each classifier.

	algorithm	**LSTM_ keras**	**ImageClassifier_autokeras**			**TabularClassifier_ autopytorch**
	preprocessing	**None**	**2D**	**GAF**	**MTF**	**None**
percentage anomaly train	**number anomaly train**					
7.39	**13.0**	0.00	0.00	0.02	0.17	0.15
17.26	**34.0**	0.08	0.00	0.17	0.41	0.33
29.44	**68.0**	0.20	0.63	0.58	0.64	0.45
38.49	**102.0**	0.49	0.63	0.69	0.69	0.58
45.67	**137.0**	0.58	0.74	0.75	0.63	0.75