Article

Assessing Feature Representations for Instance-Based Cross-Domain Anomaly Detection in Cloud Services Univariate Time Series Data

Firstname Lastname [1,†,‡], Firstname Lastname [1,‡] and Firstname Lastname [2,*]

[1] Affiliation 1; e-mail@e-mail.com
[2] Affiliation 2; e-mail@e-mail.com
* Correspondence: e-mail@e-mail.com; Tel.: (optional; include country code; if there are multiple corresponding authors, add author initials) +xx-xxxx-xxx-xxxx (F.L.)
† Current address: Affiliation 3
‡ These authors contributed equally to this work.

Citation: Lastname, F.; Lastname, F.; Lastname, F. Feature Representations for Cross-Domain Time-Series Anomaly Detection. *Journal Not Specified* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

Abstract: In this paper, we compare and assess the efficacy of a number of time-series instance feature representations for anomaly detection. To assess whether there are statistically significant differences between different feature representations for anomaly detection in time series we calculate and compare confidence intervals on the average performance of different feature sets across a number of different model types and cross-domain time-series datasets. Our results indicate that the Catch22 time-series feature set augmented with features based on rolling mean and variance performs best on average and that that the difference in performance between this feature set and the next best feature set is statistically significant. Furthermore, our analysis of the features used by the most successful model indicates that features related to mean and variance are the most informative for anomaly detection. In future work we will use the insights gained from these experiments to inform our research on cross-domain time-series anomaly detection and in particular we will explore the suitability of these feature representations for instance-based and feature-based transfer learning and also their use with deep recurrent neural network models.

Keywords: Time series analysis; Anomaly detection; Data Representation; Cloud Monitoring; AIOPS

1. Introduction

A time series is a set of data points that were collected and are indexed sequentially through time. Time series analysis can be applied to a wide range of data generating processes including data from sensors monitoring tides, stock prices, medical applications such as monitoring for cardiac pathologies, and so on. There are range methods that can be applied such as visualization, classification, anomaly detection. This paper focuses on anomaly detection in cloud systems. Anomaly detection is the identification of rare events, these events could be indicative of critical failures in cloud systems. One area of increasing importance for time-series analysis is the maintenance of cloud services. Cloud services are often made up of many microservices all of which can have different operating parameters. For these services different KPIs can be tracked and provide real-time insights into the performance of the different micro-services within the system, for example: the amount of network traffic might be useful for a load balancer, while levels of disk I/O might be informative for a database. Consequently, these KPIs provide a basis for early detection of problems in a system due to system failure, crashes, or surges in traffic. Early detection of problems in this way can allow engineers address issues or scale systems in a timely manner.

Nielsen [1] distinguishes between four different types of computational model for time series analysis, namely: (i) statistical models (e.g., auto-regressive models, and moving averages); (ii) state-space models (e.g. kalman-filters, hidden markov models); (iii) machine learning methods (e.g., SVMs, tree based ensembles) applied to hand-engineered feature sets; and (iv) end-to-end deep learning methods (e.g., recurrent architectures, and convolutional networks). Statistical and state-space methods augmented with thresholds and rules can work well for anomaly detection on relatively stable KPIs where the thresholds can be tailored to normal operation parameters of the KPI being monitored. However, it is difficult to set appropriate thresholds when there is significant variation within a KPI due to temporal or seasonal variation [2], or where there is significant variation across a set of KPIs that is being monitored as part of a cloud services infrastructure [3]. Focusing on machine learning based methods, both supervised [4,5] and unsupervised methods [6–8] for anomaly detection have been proposed. There is also a growing amount of research in applying deep learning methods to anomaly detection for time-series. Within the research on deep anomaly detection models the majority or research has focused on unsupervised methods; this is because of the difficulties that deep supervised models can have in modelling imbalanced data, and also the because of the lack of available training data (see [9] for a recent overview of this work). However, despite this current trend towards unsupervised deep anomaly detection research, and following [10] and [3], we believe supervised approaches to anomaly detection are ultimately necessary in order to achieve the high-levels of performance required to support critical cloud-based services. In particular, our long-term research goals are to develop highly-accurate supervised machine learning/deep learning approaches to anomaly detection that scale across multiple domains with minimal data labelling overhead. This points to the use of transfer learning and active learning methods in the final framework. However, as a first step in the development of a scalable framework for time-series anomaly detection in this paper we report results from a set of experiments that were designed to assess the relative performance of different feature representations of time-series data points for the task of cross-domain anomaly detection for univariate time-series data; and, we will focus on the interaction between these features sets and transfer learning and active learning in later work. The contributions of this paper are as follows:

- A comprehensive analysis of the multiple feature time-series feature sets for anomaly detection across a number of cross-domain time-series datasets
- Identifying a feature set that outperforms the next best representation by a statistically significant margin on the task of cross-domain anomaly detection
- Elaborating feature importance in the top performing representation

The rest of this paper is structured as follows: in Section 2 we review related work on feature representations for time-series analysis; in Section 3 we describe the set of features which we are considering under experiment, in Section 4 we describe our experimental design and in particular introduce the dataset and models types we use in the experiment and our use of confidence intervals as part of our experimental method for feature comparison; in Section 5 we present the experimental results; and in Section 6 we present our conclusions.

2. Related Work on Feature Engineering for Time-Series Analysis and Anomaly Detection

Feature engineering and selection can have significant impact on the performance of a data driven model on a given task. Using appropriate features can reveal relevant underlying information in a domain. For example, in time-series analysis features can reveal variance, seasonality, or deviations from observable trends. However, including irrelevant or redundant features in a representation can results in models overfitting the particularities of the training sample. Furthermore, features that may be useful in one time-series task may not be appropriate for another. For example, the use of a mean makes sense where a time-series is stationary (or at least ergodic) but is less meaningful when a time-series is non-stationary, and also features such as max and min values may become unstable as the length of a time-series increases [1, page. 242]. Consequently, selecting an appropriate feature set for a task and domain is important, although this become more challenging in a cross-domain setting.

In an era of end-to-end deep learning, many data driven projects often rely on the ability of deep networks to learn the appropriate representations for a task from the raw data [11]. However, as noted above relatively little work has been done on supervised deep learning methods for anomaly detection in time-series data because of the data required for deep learning approaches, and also because of the difficulties of training deep models using highly-imbalanced data. One example of recent work that has explored deep learning for anomaly detection in time series data is [12] which used a synthetic dataset to compare the performance of end-to-end LSTM approach trained on the raw time series with a number of machine learning models trained on feature vector representations of the same time series data points. One interesting point arising from that work was that both approaches, end-to-end LSTM and feature vector based machine learning models, achieve similar performance on the task. This does indicate that time-aware (recurrent) deep architectures such as LSTMs can learn useful representations from raw data, and this could save on feature engineering efforts. However, this research was carried out on a single domain data set of contiguous glucose monitoring for diabetes where anomalies were injected into the normal data. This dataset was specifically prepared in order to ensure that there was enough labelled examples available to train an end-to-end approach using supervised learning, and it is not clear how the performance of the models might degrade as the size of training date is reduced (although it is to be expected that the deep learning architecture's performance will be more sensitive to the data reduction than the less data hungry machine learning models). Furthermore, as we mentioned in the introduction our long-term focus is on cross-domain anomaly detection and solving this challenge will likely require the use of transfer learning and active learning as part of the solution. However, the integration between deep learning and active learning is still an open research demonstrated by [13] and [14] who have worked on deep active learning with image data, neural networks being overconfident in their prediction making selecting data for labeling challenging and due to the computational cost associated with the frequent retraining of a deep architecture. For these reasons in this work we focus on feature set design for machine learning based approaches to anomaly detection, with a particular focus on feature sets that exhibit good performance across multiple domains.

Machine learning based models (such as tree-based ensemble, feed-forward neural nets, SVM) were primarily developed for classification and regression prediction on cross-sectional data. Consequently, these models are not naturally time-aware (as compared with, for example, recurrent neural networks). As a result, in order to apply these models to time-series data it is necessary to convert each data point in a time series into a fixed-width feature vector. This fixed-width feature vector representation may include both global features that summarise the characteristics of the entire time series and/or local features that capture the relevant information (including temporal trends) within the temporal locality of the data point within the time series. Frequently, converting the points within a univariate time-series into a sequence of feature vectors is achieved by passing a sliding window—these windows can either be overlapping, or tiled (non-overlapping)—along the time-series and calculating the features values for a point over the data within the window that contains the point, and feature values for the time-series by comparing characteristics calculated for different windows. *Stability* and *lumpiness* are examples of global features that are calculated based on non-overlapping (tiled) windows [15]. The calculation of stability and lumpiness assume that a set of tiled time windows have been defined across a time-series. The stability score for a time-series is then calculated by first calculating the mean for each of the tiled windows and then calculating the variance of these means. Lumpiness is calculated by first calculating the variance for each of the tiled windows and then calculating the variance of these variances. Furthermore, although time-series are naturally considered in the time-domain it is often useful to transform a time-series into a frequency domain, for example by applying a Fourier transform to project a time-series into a sine and cosine coordinate system. These two representations naturally foreground different characteristics of a time-series: the time-domain foregrounds *lagged* relationships (how does what happen in the past affect what will happen today/tomorrow), and the frequency-domain is more suitable to analyse the cyclic nature of a time-series [16].

Examples of the types of features that can be derived through operations applied to either the temporal and/or spectral representation of a time-series include: basic statistics (e.g., the mean, max or min values in the last n time steps), and features derived from different forms of variation that may be present in the data, be it measures of trend (long terms changes in the mean), cyclic/periodic variation (e.g., seasonality), non-cyclic variation, entropy, and so on. An assumption underpinning many approaches to time-series analysis is that the time-series is *stationary*, in general terms a time-series is stationary if the properties of one slice of the series are similar to those of any other [17, pg. 17]. For similarity to be maintained in this way across the time-series it is necessary that there is no underlying trend, the variance is constant, and there are no periodic variations. If any of these conditions do not hold than different operations can be applied to transform the raw time-series into a stationary one. For example, if a trend exists in the data a common method to remove the trend (i.e., to stabilize the mean), known as *differencing*, is to transform the time-series into the differences between consecutive observations [18]. The concept of differencing can be used to generate features describing the characteristics of the time series, such as the difference between the value at time t (x_t) and a previous value at some fixed time separation or *lag* k (x_{t-k}).

A particularly important form of analysis of a time-series is autocorrelation, which measures the linear relationship between the value of a time-series at time t, (x_t), and at lag k, (x_{t-k}). The autocorrelation of a time-series can be measured at many different lags, and so there are multiple autocorrelation coefficients for each time-series, one for each lag, r_k being the autocorrelation coefficient for lag k. A standard visualisation for these coefficients in the *correlogram* in which the autocorrelation coefficients are plotted against the lag. The correlogram is often referred to as the *sample autocorrelation function* [17], and a number of features can be derived from this function, for example the lag to the first time the function crosses zero, or the first minimum of the function. The concept of autocorrelation can be extended to non-linear dependencies by calculating mutual information at different lags. Similarly, the calculated mutual information coefficients can be plotted against lags to define the automutual information function, and again a range of features can be derived from this function. Another family of important features within time-series analysis is based on *autoregression*. An autoregressive model uses lagged values of a time-series to predict the value of the series at time t. One application of autoregression is to measure the volatility within a time-series; for example, the *autoregressive conditionally heteroscedastic* (ARCH) model [19] was developed to model changes in volatility in time-series, and were later extended to the generalized ARCH (GARCH) [20]. Autoregression models, are also frequently used to identify outlier anomalies [3,4], the intuition being that the autoregression model learns the normal behaviour of the time-series and a large discrepancy between the predicted and actual value at time t is taken as evidence for a potential anomaly [6].

The breadth of possibilities in feature design for time-series is most clearly demonstrated via the *highly comparative time-series analysis* (hctsa) toolkit which provides implementations of over $7,700$ features for time series analysis and an extensive library of time-series datasets [21–23]. Given such a large set of potential features it is not surprisingly that feature selection is a very important topic in time-series analysis. There are a growing number of time-series feature sets that are now commonly used across a large number of time-series projects, some of the best known include: *feasts* [24], *tsfeatures* [25], *Kats* [26], *tsfresh* [27], *TSFEL* [28], and *catch22* [29]. The majority of these general purpose time-series feature sets are manually curated. For example, The *tsfeatures* package implements a set of operations to extract a set of different types of frequently used features from a time-series, including: the autocorrelation function and related features; various measures of trend and seasonality extracted from an STL decomposition, developed by [30], of a time-series; *flatspot* features that are computed by dividing the sample space of a time series into 10 equal length bins and computing the maximum run length within a single interval; *crossing points* which returns the number of times a time-series crosses the median line; *lumpiness*, *stability*, among others.

An exception to these manually designed feature sets is the recently developed catch22 feature set. The catch22 – CAnonical Time-Series Characteristics – feature set was designed to be "*generically useful*" and developed using a data driven methodology that involved evaluating a set of 4791 features from hctsa on 93 time-series classification tasks, and selecting 22 features that perform well across a diverse set of 93 time-series data sets, covering multiple time-series tasks, and which were minimally redundant [29]. The selection of features from the huge pool of features was achieved using a three step process: (i) *statistical prefiltering* of the features by removing any feature whose performance across the 93 datasets was statistically equivalent to a random number generator; (ii) *performance filtering* of the remaining features by ranking the features by performance across the datasets and dropping all features whose overall performance was below a threshold; (iii) *redundancy minimization* which involved clustering the remaining features into groups based on similar performance on *tasks* (i.e., sets of similar datasets) and selecting a representative of each cluster for inclusion in the final set of features. The clustering process generated 22 clusters and hence 22 features were included in the final catch22 feature set. The catch22 feature set had comparable performance across the datasets as the initial pool of 4791 features from hctsa: hctsa feature set achieved an average accuracy of 77.2% whereas the catch22 feature set achieved an average accuracy of 71.7%. The catch22 creator notes that some of the datasets used in their feature evaluation experiments were normalised and this resulted in features that are derived from the mean and variance of a time-series begin excluded from consideration; as a result they recommend to consider including features related to the mean and variance if applicable when using the catch22 feature set [29]. The catch22 feature set includes linear and non-linear autocorrelation, successive differences, value distribution and outliers and fluctuation scaling properties. The complete list of catch22 features are listed in Table 1 below.

Table 1. The catch22 feature set (based on Table 1 from [29]).

Feature	Description
Distribution	
DN HistogramMode 5	Mode of z-scored distribution (5-bin histogram)
DN HistogramMode 10	Mode of z-scored distribution (10-bin histogram)
Simple temporal statistics	
SB BinaryStats mean longstretch1	Longest period of consecutive values above the mean
DN OutlierInclude p 001 mdrmd	Time intervals between successive extreme event above the mean
DN OutlierInclude n 001 mdrmd	Time intervals between successive extreme events below the mean
Linear Autocorrelation	
CO f1ecac	First 1/e crossing of autocorrelation function
CO FirstMin ac	First minimum of autocorrelation function
SP Summaries welch rect area 5 1	Total power in lowest fifth of frequencies in the Fourier power spectrum
SP Summaries welch rect centroid	Centroid of the Fourier power spectrum
FC LocalSimple mean3 stderr	Mean error from a rolling 3-sample mean
Non Linear Autocorrelation	
IN AutoMutualInfoStats 40 gaussian fmmi	First minimum of the automutual information function
CO HistogramAMI even 2 5	Automutual information, $m = 2, \tau = 5$
CO trev 1 num	Time-reversibility statistic, $\left\langle (x_{t+1} - x_t)^3 \right\rangle_t$
Successive Difference	
MD hrv classic pnn40	Proportion of successive differences exceeding 0.04σ
SB BinaryStats diff longstretch0	Longest period of successive incremental decreases
SB MotifThree quantile hh	Shannon entropy of two successive letters in equiprobable 3 letter symbolization
FC LocalSimple mean1 tauresrat	Change in correlation length after iterative differencing
CO Embed2 Dist tau d expfit meandiff	Exponential fit to successive distances in 2-d embedding space
Fluctuation Analysis	
SC FluctAnal 2 dfa 50 1 2 logi prop r1	Proportion of slower time scale fluctuations that scale with DFA (50% sampling)
SC FluctAnal 2 rsrangefit 50 1 logi prop r1	Proportion of slower time scale fluctuations that scale with linearly rescaled range fit
Others	
5SB TransitionMatrix 3ac sumdiagcov	Trace of covariance of transition matrix between symbols in 3-letter alphabet
PD PeriodicityWang th0 01	Periodicity

A recent empirical evaluation of the coverage, computational efficiency, and within feature-set redundancy of a number of general purpose time-series feature sets across a set of over 800 real-world and synthetic time-series dataset found that: *hctsa* was the most comprehensive feature set; *TSFEL* and *catch22* were the fastest to be evaluated for time series in the range of 100-1000 samples; and that the *catch22* had the lowest within set feature redundancy (which was a deliberate design choice for this feature set) [31]. However, a conclusion of this benchmarking work was that it would be important in future work to evaluate the relative performance of feature sets on specific time-series analysis tasks [31, pg. 7].

The focus of our work anomaly detection, and more specifically anomaly detection for cloud infrastructure. A number of researchers have proposed manually curated feature sets tailored to anomaly detection, for example [3,4,15,32]. A recent example of this task specific approach is the feature set developed in [3] which was specifically designed for the task of anomaly detection for cloud systems. Similar to the general catch22 feature set from [29], the [3] feature set also includes 22 features. Consequently, these two feature sets are comparable representatives of the general versus task specific feature set design process. As with the catch22 feature set, the feature engineering process used for the feature set from [3] converts each data point in the time series into a set of 22 features which can capture both temporal and contextual information around the point. The conversion from a univariate time-series to a sequence of 22-dimensional feature vectors is achieved by passing a sliding window along the time-series using a step size of 1, and each time the window is moved one time-step forward a feature vector is generated to represent the right-most point in the time-series by calculating the 22 features using the segment of the time-series covered by the window. For a given time-series the width of the sliding window is calculated by using a Discrete Fourier Transform (DFT) to estimate the periodicity of the data and using this periodicity to determine the size of the sliding window. This feature set includes three categories of features: Statistical features, Forecasting error features, and Temporal features. We describe each of these feature sets below.

Statistical features are used to describe basic characteristics around each data point in time series. The relevance of these statistical features for anomaly detection in time-series is based on the expectation that anomalies may violate these basic characteristics [3]. The set of statistical features included in the feature set from [3] are listed in Table 2.

Table 2. The statistical feature set from Table 1 of [3]

Feature	**Description**
ACF1	First order of autocorrelation
ACF Remainder	Autocorrelation of remainder
Mean	Rolling Mean
Variance	Rolling Variance
Entropy	Spectral Entropy
Linearity	Strength of linearity
Trend	Strength Trend
Crossing Point	Number of crossing points
ARCHtest	P value of Lagrange Multiplier (LM) test for ARCH model
Curvature	Strength of Curvature computed on Trend of STL decomposition
GARCHtest.p	P value of Lagrange Multiplier

Error forecasting features are also considered relevant to anomaly detection based on the intuition that if an actual point deviates dramatically from the forecast for that point then it is most likely to be an anomaly. In [3] ensemble models are used to carry out forecasting and different models are used based on the seasonality of the time series. The metric RMSE (Root Mean Squared Error) is used to provide different forecasting methods with different weights at a fixed time. The weighted prediction result of the ensemble model from M models at time t is calculated by:

$$\widehat{Y}_t = \sum_{m=1}^{M} \frac{\widehat{Y}_{m,t}}{M-1} \cdot \left(1 - \frac{RMSE_{m,t}}{\sum_{n=1}^{M} RMSE_{n,t}}\right) \tag{1}$$

where $\widehat{Y}_{m,t}$ is prediction by model m at time t, $RMSE_{m,t}$ is the prediction error of model m at time t, and $\widehat{Y}_t$ is the ensemble prediction at time t. Table 3 below lists the forecast error features used in [3]. Note that the inclusion of a large number of forecasting error features in the feature set does tailor the feature set to the task of anomaly detection, however it likely does so at the cost of increasing the internal redundancy among the features.

Table 3. Forecasting error features from Table 2 of [3]

Feature	**Formula**	**Description**
RMSE	$\sqrt{\frac{\sum\left(Y_i-\widehat{Y}_i\right)^2}{N}}$	Root mean square error
ME	$\frac{\sum\left(Y_i-\widehat{Y}_i\right)}{N}$	Mean Error
MAE	$\frac{\sum\lvert Y_i-\widehat{Y}_i)\rvert}{N}$	Mean Absolute Error
MPE	$\frac{1}{N} \cdot \frac{\sum Y_i-\widehat{Y}_i}{Y}$	Mean Percentage Error
MAPE	$\frac{1}{N} \cdot \frac{\sum\lvert Y_i-\widehat{Y}_i\rvert}{Y}$	Mean Absolute Percentage Error

The final set of features proposed in [3] are temporal features. These features are calculated relative to previous windows, and they are designed to capture drastic changes between windows. For example, some of these features compare the data in two consecutive windows and also computes the difference between the current value and previous w values (e.g., the difference between x_{n-w} and x_n). Table 4 lists the temporal features proposed in [3].

Table 4. Temporal features set from Table 3 of [3]

Feature	**Description**
Max level shift	Max trimmed mean between two consecutive windows
Max var shift	Max variance shift between two consecutive windows
Max KL shift	Max shift in Kullback-Leibler divergence between two consecutive windows
Diff-w	The differences between the current value and the w-th previous value
Lumpiness	Changing variance in remainder
Flatspots	Discretize time series value into ten equal sized interval, find maximum run length within the same bucket

Another recent work which focused on anomaly detection for time series data is [32] which uses the spectral residual (SR) algorithm to convert raw time series into a feature representation that is then fed into a convolutional neural network (CNN) which is responsible for learning a decision rule for distinguishing between anomalies and non-anomalies. In [32] the CNN model is trained using artificially generated anomaly labels. However, for this paper the most relevant aspect of [32] is the SR algorithm that is used to transform the raw data before presentation to the CNN. The SR algorithm was originally developed to identify regions of interest in images for computer vision (based on concepts from bottom up saliency detection) [33]. In [32] the SR algorithm is repurposed to the time-series domain in an attempt to identify points of interest in time-series data (such as anomalies). The SR algorithm is a domain independent unsupervised method that consists of three major steps: (i) applying a Fourier Transform to extract the log amplitude spectrum of the input sequence; (ii) calculation of the spectral residual by subtracting the average log spectrum from the log spectrum; and (iii) then an Inverse Fourier Transform is applied to convert the spectral residual back into the spatial domain.

The comparison of feature representations for instance-based cross-domain anomaly detection across multiple cloud-services univariate time-series datasets presented in this paper is based on the above feature sets. One focus of this analysis is to asses whether in a cross-domain setting there is a statistical difference in performance between general time-series feature sets and anomaly detection specific feature sets. In this context it is worth noting that there is relatively little overlap between the features listed in the catch22 and [3] feature sets. For example, the catch22 feature set has features based on forecasting errors and these are not present in catch22. Another focus is to examine whether combining features from different sets or extending these feature sets produces better overall results, for example combining SR with catch22 or with the feature set of [3]. Finally, the analysis will also consider whether there is any similarity in terms of the most useful features found in these alternative feature sets for anomaly detection in a cross-domain setting.

3. Feature Sets Under Test

As a baseline representation for our experiments we use the raw time series without any form of preprocessing. In this raw representation each data point is represented by its raw value and the raw values of preceding $n-1$ data points. For all datasets we set $n = 15$. This window size was set as the mean periodicity for a set of time series sampled from across the different datasets used in the experiments (these datasets are introduced in the Experimental Design section below). In more detail, we used DFT to analyse the periodicity of each of the selected time-series and then averaged across the results of this analysis.

All of the other feature sets we assess also use a sliding window to define the context that the features are calculated over. For these other feature sets that size of this window was set separately for each time-series in a dataset. To set the window size for a time series we used the AUTOPERIOD method proposed in [34] which calculates the periodicity of a time series using Periodogram and Autocorrelation. This AUTOPERIOD method involves a two step process:

1. a list of candidate periods are extracted from the time series using the periodogram method, and
2. the candidate periods are then ranked using an autocorrelation method ACF.

The intuition behind the method is that if a candidate period from the periodogram lies on a hill of the ACF then we can consider it as a valid period otherwise the period is discarded as a false alarm.

Having defined a window size per time series in a dataset the next step in processing is to convert each data point into a feature vector. We have defined a number of different feature vectors based on feature sets reported in [3] and [29]. For the feature set from [3] we dropped a number of features. In particular, we dropped Lumpiness and Flatspot from the temporal based features (this was because in our experiments we considered features that are calculated over the running window whereas Lumpiness and Flatspot produce a constant number of values for any length of time series) and we also dropped GARCHtest from the statistical features (our reason for dropping GARCHtest was that we had an easy to use implementation of ARCHtest available to us and GARCH is just the extension of ARCH which considers an extra lag parameter). However, we also augmented this reduced feature set with three other features that we thought could be relevant, these were:

1. *Spectral Residual* is the concept borrowed from the visual computing world which is based on Fast Fourier Transform (FFT). Spectral Residual is unsupervised and has proven its efficacy in visual saliency detection applications. In anomaly detection tasks, the anomalies in the time series have similar characteristics as saliency detection i.e they both visually stand out from the normal distribution.
2. *Skewness* is the measurement of the symmetry in a frequency distribution. Hence we believe that the presence of any outlier may affect the symmetry of the frequency distribution and might be helpful in anomaly detection tasks.
3. *Kurtosis* is often described as the extent to which the peak of a probability distribution deviates from the shape of a normal distribution i.e. whether the shape is too peaked or too flat to be normally distributed. This feature can be used as a complementary feature with the skewness in case of distributions that are symmetrical but are too peaked or too flat to be normally distributed.

In what follows, to distinguish this modified feature set from the original feature set proposed in [3] and we refer to this modified feature set as the *Handcrafted Anomaly Detection* (HAD) feature representation. We can summarise this new feature set as:

$$\begin{aligned} HAD = &(Tables\ 2 + 3 + 4) \\ &- (Lumpiness, Flatspot, GARCHtest) \\ &+ (SR, Kurtosis, Skewness) \end{aligned} \tag{2}$$

We also assessed a number of variant of the Catch22 feature set, these were: (i) the standard Catch22 set, (ii) the Catch22 feature set augmented with the Spectral Residual (SR) feature, we refer to this feature set as Catch23, (iii) the Catch22 feature set augmented with window mean and variance features, which we refer to as Catch24, and (iv) the Catch22 feature set augmented with SR, window mean and window variance, which we refer to as Catch25. In total this resulted in us assessing the following 6 feature sets:

1. Raw (window size = 15),
2. HAD
3. Catch22,
4. Catch23 (Catch22+Spectral Residual),
5. Catch24 (Catch22+Mean+Variance),
6. Catch25 (Catch22+SR+Mean+Variance).

4. Experimental Design

This section provides an overview of the experimental process we used to analyse the relative effectiveness of the six different features sets for instance-based anomaly detection in cross-domain time-series. Within this study we wish to control for model type, and also for domain sample. Consequently, we assess the effectiveness of each feature representation across all combinations of a number of different models and different datasets.

Following [12] we used the following set of 6 model types in our experiments: naive bayes, tree-based ensembles (random forests and xgboost), feedforward neural networks, and SVM models (SVC and SVC linear). Note that the focus of this paper is on assessing the general effectiveness of different feature sets, rather than on fine-tuning the performance of models. Consequently, we did not expend a large amount of time on fitting model hyper-parameters. Instead, we selected a single set of hyper-parameters for each model type and kept these consistent across the datasets. In most cases these were the default hyper-parameters set in *sklearn* [35]. The following hyper-parameters were used for training each of these model types:

- *Naive bayes*: Model parameters are inferred from training data.
- *Random forest*: 200 estimators, $max\_features = \sqrt{n\_features}$, no max depth, two samples as minimum number of samples required to split an internal node, one as minimum number of samples in newly created leaves, with bootstrapping, using out-of-bag samples to estimate the generalization error.

- *Xgboost*: gbtree booster, Step size shrinkage of 0.3, 0 Minimum loss reduction required to make a further partition on a leaf node of the tree, 6 maximum depth of a tree,
- *Multi-layer perceptron*: 1 hidden layer with 100 ReLU units per layer with a single logistic unit in the output layer, adam as the solver, using L2 penalty of 0.0001, batch size of 200, constant learning rate, 0.001 step-size in updating the weights, max iterations set to 200, no early stopping, 0.9 as exponential decay rate for estimates of first moment vector, 0.999 as exponential decay rate for estimates of second moment vector, 1e-8 for numerical stability.
- *SVC*: rbf kernel, Regularization parameter of 1.0, scaled kernel coefficient on data set variance and vector length.
- *SVC linear*: linear kernel, Regularization parameter of 0.25.
  - Other SVC parameters: shrinking heuristic, no probability estimates, 1e-3 Tolerance for stopping criterion, 200MB kernel cache, no class weights, no limit on number of iterations, one-vs-rest decision function.

We use 6 different datasets for analysing the feature representations: NAB (AWS and Twitter) [36], Yahoo (Real and Artificial) [4], IOPS KPI (sometimes referred to as AIOPS in literature)[1], and Huawei. These datasets have time series of different time intervals and fan across different spectrum of patterns. In more detail:

- NAB (AWS and Twitter) is a benchmark for evaluating anomaly detection algorithms in streaming, real time applications. It is composed of over 50 labeled real-world and artificial time series. We focus on two of the real world datasets: (i) web monitoring statistics from AWS, and (ii) Twitter tweet volumes. The AWS time series are taken from different server metrics such as CPU utilization, network traffic, disk write bytes etc. The Twitter dataset is the number of Twitter mentions of publicly-traded companies such as Google and IBM. The value represents the number of mentions for a given ticker symbol in every 5 windows.
- Yahoo Anomaly Detection Dataset is a publicly available collection of datasets released by Yahoo for benchmarking anomaly detection algorithms. The dataset has four sets of time series one is collected from production traffic to yahoo services which is labelled by editors manually whereas the others are synthesized with anomalies that are embedded artificially. We use the real data, and syntheised data but not the “A3” and “A4” datasets.
- IOPS KPI dataset is a collection of time series datasets provided by Alibaba, Tencent, Baidu, eBay, and Sogou. The dataset is from real traffic to web services and was published as part of a series of anomaly detection competitions.
- Huawei dataset is the dataset obtained from the Anomaly detection Hackathon organised by Huawei[2]. The selected time series contains different KPI values and the datapoint are labelled against the anomaly

Table 5 below provides summary statistics of the size of each dataset and the % of anomalies present in each dataset.

Table 5. Summary statistics for the datasets used in the experiment

Dataset	Number of Points	% Anomalies	Number of time series	Mean Length
Yahoo Real	91k	1.76%	64	1415
Yahoo Art	140k	1.76%	100	1415
IOPS	3M	2.26%	29	105985
AWS	67k	4.57%	17	67740
Twitter	142k	0.15%	10	142765
Huawei	54k	4.19%	6	9056

The algorithm below provides a pseudo-code description of our experimental approach.

```
1. for each feature set:
   (a) for each dataset:
       i. for each model type:
          A. use k-fold cross-validation to train an evaluate k models
          B. record the performance of each of these k models
   (b) calculate an aggregate performance metrics (e.g. mean F1) across all model instances and dataset folds
```

[1] Available from: https://github.com/NetManAIOps/KPI-Anomaly-Detection
[2] Details of the Hackathon competition are available at: https://huawei-euchallenge.bemyapp.com/ireland

(c) calculate confidence intervals for each aggregate performance metric using the variance across the sample used to calculate the aggregate metric (i.e., the variance across the results on each model type instance and folds across the datasets)

2. rank feature sets by aggregate performance metrics
3. check for statistical significance in differences by comparing confidence intervals

Below we describe the performance metrics we record and how we calculate the confidence interval for each metric, and the set of models and datasets we use.

We use the F1, precision and recall as our metrics for evaluating performance:

$$R = \frac{TP}{TP + FN} \tag{3}$$

$$P = \frac{TP}{TP + FP} \tag{4}$$

$$F1 = \frac{2 \times P \times R}{P + R} \tag{5}$$

where P and R are Precision and Recall respectively. TP, FP and FN refers to True Positive, False Positive and False Negative instances. F1 is the main evaluation metric for our models as if we pay more attention towards precision we might miss out on some of the crucial anomalies and reduce the recall whereas focusing on recall only may increase the number of false positives hence we take F1 to maintain the balance between precision and recall.

As described above for each feature set we calculate aggregate performance metrics for F1, Precision, and Recall by calculating the mean value for each metric across all model types and dataset folds. In other words, given that we use 5-fold cross validation, and given that we have 6 model types, and 6 datasets the mean F1 score for a feature set is calculated across a sample of (folds$\times$model-type$\times$dataset) $5 \times 6 \times 6 = 180$ observations. We can calculate a confidence interval around the mean value in order to quantify the uncertainty in this mean value with respect to the unknown true mean for this feature set across the models and datasets considered using the following equation:

$$CI = \bar{x} \pm T \times \frac{\sigma}{\sqrt{n}} \tag{6}$$

where, $\bar{x}$ is the mean value of observations, T is the critical value of Z-score for the respective confidence level from T-distribution, σ is the standard deviation of the sample and n is the number of observations in the sample. By comparing whether there is an overlap between the confidence intervals of different feature sets we can determine whether a difference in mean performance is statistically significant at the chosen confidence level.

Figure 1 below illustrates the combinations of datasets, feature sets and models that we used in our experiments. In the next section we will report the results from these experiments.

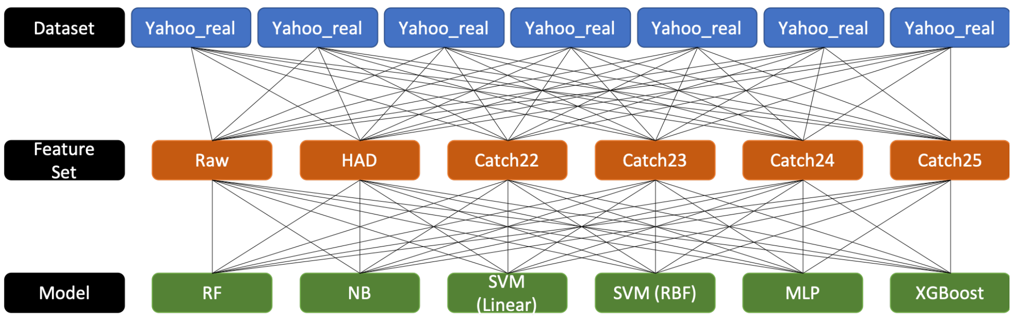

Figure 1. The combinations of datasets, feature sets and models used in the experiments. Non-standard terminology and acronyms: catch23 is catch22+sr, catch24 is catch22+sr+mean, catch25 is catch22+sr+mean+variance, random forest (RF), naive Bayes (NB), multi-layer perceptron (MLP), radial basis function (RBF), multilayer perceptron (MLP)

5. Results

Table 6 presents for each feature set the mean F1, Recall and Precision along with the 95% confidence interval and ranges. Our initial experimental plan was that for each feature set the calculation of the mean values and confidence intervals is calculated across the set of results generated by training one instance of each of the six types of models on each of 5 folds per 6 dataset. This would have meant that each mean value and confidence interval would have been calculated based on a population of 180 samples (6 model types by 6 datasets and 5 folds per dataset). However, in some model/fold combinations the model did not converge, in particular the SVC and SVC Linear models failed to fit on some folds. Consequently, the observation population size per feature set was as follows: Raw:157; HAD:179; Catch22:172; Catch23 (Catch22+SR):172; Catch24 (Catch22+mean+var):151; and Catch25 (Catch22+mean+var+SR):151.

Table 6. For each feature set under test the mean F1 Recall and Precision along with confidence intervals calculated to 95% and range calculated across model types and dataset fold combinations (populations sizes: 157,179,172,172,151,151)

	F1			Recall			Precision		
	Avg	Conf	Range	Avg	Conf	Range	Avg	Conf	Range
Raw (ws=15)	0.2558	±0.0390	0.2168 0.2948	0.2263	±0.0389	0.1873 0.2652	0.5604	±0.0565	0.5039 0.6169
HAD	**0.6481**	±0.0464	0.6016 0.6945	**0.6297**	±0.0465	0.5833 0.6762	0.7818	±0.0460	0.7358 0.8278
Catch22	0.5537	±0.0528	0.5009 0.6066	0.5132	±0.0516	0.4617 0.5648	0.6993	±0.0541	0.6452 0.7534
Catch23	0.5506	±0.0525	0.4980 0.6031	0.509	±0.0511	0.4579 0.5601	0.699	±0.0542	0.6447 0.7532
Catch24	0.5889	±0.0614	0.5275 0.6503	0.5407	±0.0608	0.4799 0.6015	**0.7826**	±0.0507	0.7319 0.8333
Catch25	0.5825	±0.0612	0.5213 0.6436	0.532	±0.0608	0.4712 0.5928	0.7812	±0.0504	0.7308 0.8316

In Table 6 we can see that the HAD feature set performs the best, followed by different variants of Catch22 and the Raw feature set does not perform well at all. Examining how the inclusion of extra features into the Catch22 feature set affects performance it is interesting to note that the addition of the SR feature actually resulted in a drop in performance (both in the case of Catch22 versus Catch22+SR, and Catch22+mean+var versus Catch22+mean+var+SR). Why the inclusion of the SR feature does not help is something we will explore in future work. However, the inclusion of mean and variance features into the Catch22 feature set does result in an improved performance (in terms of F1, Recall and Precision), and in fact the Catch24 (Catch22+mean+var) has the second best performance in terms of F1 and Recall and has the highest Precision overall. Returning to the HAD feature set, although this feature set was the best performing in terms of F1, at the 95% confidence level there is overlap between the ranges for the mean F1 confidence interval of the HAD feature set and all the Catch22 based features sets, and so this good performance of the HAD feature set is not statistically different from the Catch22 based feature sets at the 95% confidence interval. We also examined what would happen at a 90% confidence interval and found that there is a statistically significant difference between the HAD feature set and Catch22 and Catch23, however with the addition of rolling mean and variance to the features (i.e., Catch24) that margin is closed and the difference in performance is no longer statistically significant.

We also examined the mean performance by dataset and by model type. Table 7 presents the mean F1 score and confidence interval for all feature set * model type combinations across the five folds of each dataset. Ideally, each of these mean F1 scores and confidence intervals would have been calculated across a population of 180 observations: 6 feature sets * 6 model types * 5 folds. However, as noted above not all models converged on all folds and consequently the population size for each dataset was as follows: AWS:165; Huawei: 180; IOPS:137; Twitter:165; Yahoo Artificial:170; and Yahoo Real:165.

Table 7. Mean F1 by dataset, confidence interval calculated to 95% with population sizes 165, 180, 137, 165, 170, 165 respectively.

Data	Average F1	Confidence (Error Margin)	Range
AWS	0.3963	±0.0549	0.3414 0.4512
Huawei	0.5830	±0.0553	0.5278 0.6383
IOPS	0.5693	±0.0513	0.5180 0.6206
Twitter	0.3729	±0.0476	0.3253 0.4205
Yahoo Artificial	**0.7724**	±0.0392	0.7333 0.8116
Yahoo Real	0.4958	±0.0593	0.4365 0.5551

Table 8 presents the mean F1 score and confidence interval for all feature set * dataset combinations across the six model types. Ideally, each of these mean F1 scores and confidence intervals would have been calculated across a population of 180 observations: 6 feature sets * 6 dataset * 5 folds. However, as noted above not all models converged on old folds and consequently the population size for each dataset was as follows: MLP:180; Naïve Bayes:180; RF:180; SVC:163; SVC Linear:99; and Xgboost:180.

Table 8. Mean F1 by model type, confidence interval calculated to 95% with population sizes 180, 180, 180, 163, 99, 180 respectively.

Data	Average F1	Confidence (Error Margin)	Range
Multi-Layer Perceptron	0.6129	±0.0451	0.5677 0.6580
Naive Bayes	0.2386	±0.0242	0.2144 0.2627
Random Forest	**0.8279**	±0.03110	0.7967 0.8590
SVC	0.3259	±0.04680	0.2791 0.3727
SVC Linear	0.2233	±0.0710	0.1524 0.2943
Xgboost	0.8082	±0.0322	0.7760 0.8404

Looking at the dataset results in Table 7 the results for Yahoo Artificial are significantly better than those of the other datasets. This indicates the results on the Yahoo Artificial may be outliers that are skewing the feature set results we observed in Table 6. This consideration together with the fact that Yahoo Artificial is a synthetic dataset prompted us to consider removing the Yahoo Artificial results from our analysis and to examine how this would affect the relative performance of the different feature sets. Similarly if we examine the model results in Table 8 we can observe that the best performing models are Random Forest and XGBoost and, indeed, these tree based algorithms perform significantly better than the other models. Consequently we also decided to remove the three weakest model types (Naive Bayes, the SVM based models, and Multi-Layer Perceptron) and reexamine the results for the feature sets when these weak model results are removed. Table 9 presents the per feature set mean performance results and confidence intervals after the removal of the results for the weak models and the Yahoo Artificial feature set. The removal of these outlier values resulted in a reranking in the performance of the different feature sets: Catch24 (Catch22+mean+var) now performs better than HAD in terms of F1, recall, and precision. In fact the improvement in F1 performance between the Catch24 feature set and the HAD feature set is statistically significant at the 95% confidence interval, and the addition of mean and variance in Catch24 provides a significant improvement in F1 score when compared to Catch22 and Catch23.

Table 9. For each feature set under test the mean F1 Recall and Precision along with confidence intervals calculated the 95% interval (in all cases using a population of 50 samples) and range calculated across the results for the Random Forest and Xgboost model types and dataset fold (excluding Yahoo Artificial) combinations.

	F1			Recall			Precision		
	Avg	Conf	Range	Avg	Conf	Range	Avg	Conf	Range
Raw (ws=15)	0.4253	±0.0700	0.3553 0.4954	0.3236	±0.0693	0.2543 0.3930	0.7800	±0.0445	0.7355 0.8245
HAD	0.8675	±0.0269	0.8406 0.8944	0.8030	±0.0378	0.7652 0.8408	0.9556	±0.0144	0.9412 0.9700
Catch22	0.8434	±0.0315	0.8119 0.8749	0.7610	±0.0419	0.7191 0.8029	0.9598	±0.0120	0.9478 0.9718
Catch23	0.8360	±0.0313	0.8047 0.8673	0.7495	±0.0413	0.7082 0.7909	0.9587	±0.0123	0.9464 0.9710
Catch24	**0.9167**	±0.0196	0.8971 0.9364	**0.8678**	±0.0283	0.8395 0.8960	**0.9766**	±0.0070	0.9696 0.9835
Catch25	0.9161	±0.0198	0.8964 0.9359	0.8671	±0.0284	0.8387 0.8954	0.9760	±0.0073	0.9687 0.9832

5.1. Feature Importance by Random Forest

The Random Forest model has the best overall model performance of any model type (see Table'8). The Random Forest model also has a built-in feature importance which is computed by Gini importance (mean decrease in impurity). A Random Forest is the set of Decision Trees (DT) and each DT is the combination of internal nodes and leaves. The selected features in the internal nodes are used to make the decision how to divide the data into two separate sets with similar responses. The selection criterion for the internal nodes for classification tasks can be gini impurity or information gain. Random forest measures how each feature decreases the impurity of the split. Hence for each feature we can collect the average decreases the impurity by splitting by that feature in each tree and use this as a measure of feature importance across the ensemble of trees.

The figures below show the feature importance ranking of the features in the Catch25 (we chose Catch25 over Catch24 because it included the largest set of features based on Catch22) and the HAD feature sets for the Random Forest model when training on the Huawei and IOPS dataset (we selected these dataset as we believe they are the most representative of cross-domain anomaly detection in cloud architectures). Figures 2 and 3 plot for the random forest model trained on the Huawei dataset the feature rankings from the Catch25 and HAD datasets respectively; and Figures 4, and 5 shows corresponding plots for the IOPS dataset. Examining the ranking of the features it is noticeable that features related to mean and variance are in the top three positions across all combinations of the Huawei and IOPS datasets and Catch25 and HAD feature sets (Note: the features termed as rolling_avg or the Mean and rolling_var or Var are processed in the same way, they were computed over a moving window). More specifically, the mean feature is ranked in the top 2 position in both of the datasets and feature representations whereas the other features such as Crossing Point, Trend, Variance and pNN40 which are in top three positions are indirectly extracted using mean or the variance. For example the Crossing point is the number of points crossing the mean level at the given window and the Trend is the strength of Trend which is the variance on the remainder of time series which we get from the time series decomposition into Trend, Seasonality and Remainder. These ranking of the mean and variance features aligns with the results of the experiments that showed that adding mean and variance improved the performance of the Catch22 feature set performance.

Finally, we also noted earlier that the HAD dataset included a large number of features based on forecast errors (see Table 3, and listed here for convenience: root mean squared error (RMSE), mean error (ME), mean absolute error (MAE), mean percentage error (MPE), mean absolute percentage error (MAPE)). The inclusion of these features reflects the fact that the HAD feature set was hand-tailored towards the task of anomaly detection and these features may be particularly useful for this task because a large forecast error may signal the presense of an anomaly. However, we also noted that the inclusion of a number of features that are likely to be correlated contrasted with the design principals of the catch22 feature set which was specifically designed to be a feature set with minimal redundancy. Reviewing Figure 3 we can see that on the Huawei dataset these features have low importance for the model ranked at 10th (MPE), 11th (MAPE), 20th (MAE), 21st (ME), and 25th or least important (RMSE) . By contrast, in Figure 4 the random forest ranks all of these features in the top 10 most important features: at 4th (MPE), 5th (MAPE), 6th (RMSE), 8th (MAE) and 9th (ME). The fact that the ranks of these set of features changes consistently as the dataset changes indicates that these features are redundant, and likely it would be a good idea to remove some of them. Furthermore, the impact of this set of anomaly detection specific but redundant features is evident in the performance of the Random Forest model on these two datasets.

Table 10 lists the mean F1 Recall and Precision of the Random Forest model on the Huawei and IOPS datasets using the Catch25 and HAD feature sets respectively. It can be observed that on the Huawei dataset for which the model ranks these features with low importance the Catch25 feature set outperforms the HAD feature set in Precision, Recall and F1. However, for the IOPS dataset the Random Forest model returns much higher importance rankings for these forecast features and the performance of the HAD features set on this dataset is much more competitive, resulting in higher Recall and a slightly better F1 score. Overall we take from this analysis that for some (but not all) anomaly detection datasets including forecast error features can be useful, however minimizing redundancy between features is still likely a useful principle.

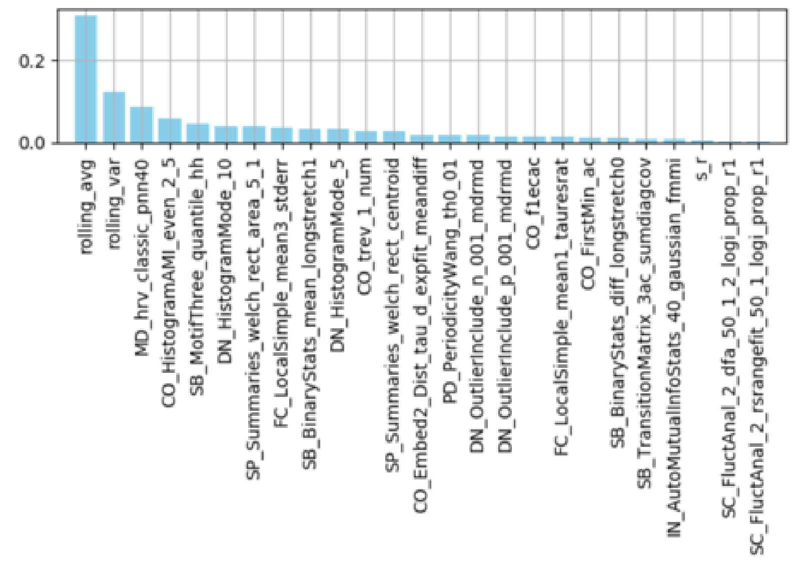

Figure 2. Catch25 feature importance for Random Forest on Huawei dataset

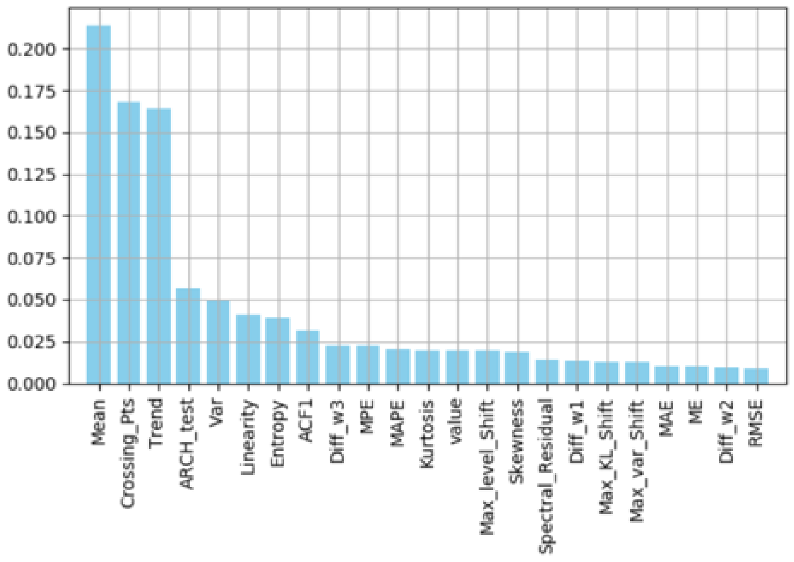

Figure 3. HAD feature importance for Random Forest on Huawei dataset

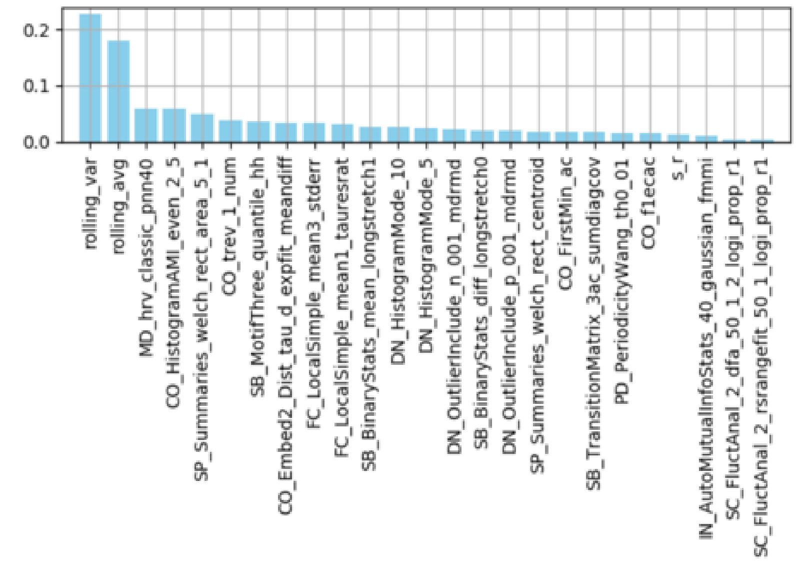

Figure 4. Catch25 feature importance for Random Forest on IOPS dataset

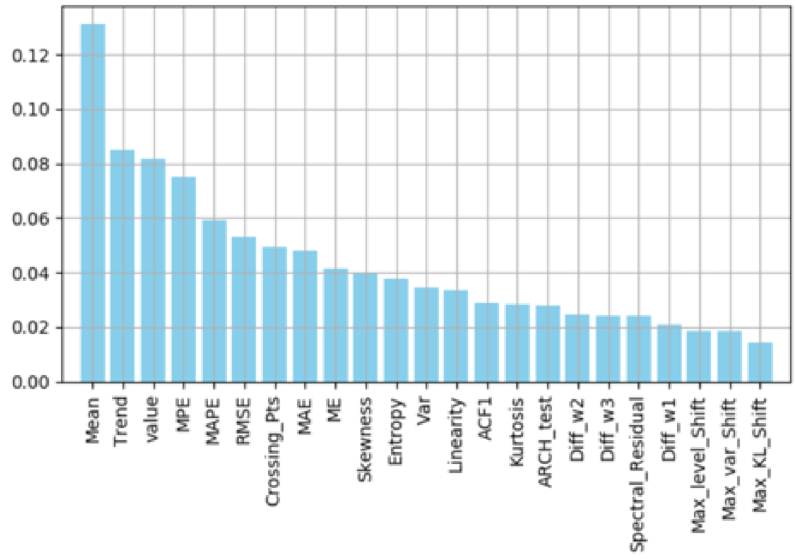

Figure 5. HAD feature importance for Random Forest on IOPS dataset

Table 10. The mean F1 Recall and Precision of the Random Forest model on the Huawei and IOPS datsets using the Catch25 and HAD feature sets respectively.

	Huawei			IOPS		
	Prec.	Recall	F1	Prec.	Recall	F1
Catch25	0.9940	0.9465	**0.9697**	0.9908	0.8957	0.9409
HAD	0.9881	0.8759	0.9287	0.9897	0.9069	**0.9465**

6. Conclusions

To conclude, across the representations that we assessed we find the Catch24, Catch22 with the addition of variance and mean, to be the best performing feature representation. When looking at the best performing models and removing outlier datasets we see there is a statistically significant improvement in F1 score between Catch24 and both HAD and Catch22, at 95% confidence level. Looking at the importance of features reinforces that features based on mean and variance are useful for anomaly detection. For our instance based anomaly detection spectral residual does not seem like a particularly useful feature. Spectral residual may be more useful to consider for sequential analysis such as recurrent architectures, because in the literature the use of this feature for anomaly detection is associated with improving the latency of the detection which points to its use being most beneficial in sequential/recurrent time-aware models. We also found that these representations work best with tree based ensemble models; although it is possible that hyperparameter tuning or additional preprocessing could improve any of the models tested. These results will feed into our future work in terms of providing a basis for selecting both the feature representations we explore and the model types we use in our experiments.

References

1. Nielsen, A. *Practical time series analysis: Prediction with statistics and machine learning*; O'Reilly Media, 2019.
2. Qiu, J.; Du, Q.; Qian, C. Kpi-tsad: A time-series anomaly detector for kpi monitoring in cloud applications. *Symmetry* **2019**, *11*, 1350.
3. Zhang, X.; Lin, Q.; Xu, Y.; Qin, S.; Zhang, H.; Qiao, B.; Dang, Y.; Yang, X.; Cheng, Q.; Chintalapati, M.; Wu, Y.; Hsieh, K.; Sui, K.; Meng, X.; Xu, Y.; Zhang, W.; Shen, F.; Zhang, D. Cross-dataset Time Series Anomaly Detection for Cloud Systems. 2019 USENIX Annual Technical Conference (USENIX ATC 19); USENIX Association: Renton, WA, 2019; pp. 1063–1076.
4. Laptev, N.; Amizadeh, S.; Flint, I. Generic and scalable framework for automated time-series anomaly detection. Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, 2015, pp. 1939–1947.
5. Liu, D.; Zhao, Y.; Xu, H.; Sun, Y.; Pei, D.; Luo, J.; Jing, X.; Feng, M. Opprentice: Towards practical and automatic anomaly detection through machine learning. Proceedings of the 2015 Internet Measurement Conference, 2015, pp. 211–224.
6. Ahmad, S.; Lavin, A.; Purdy, S.; Agha, Z. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* **2017**, *262*, 134–147.
7. Terzi, D.S.; Terzi, R.; Sagiroglu, S. Big data analytics for network anomaly detection from netflow data. 2017 International Conference on Computer Science and Engineering (UBMK). IEEE, 2017, pp. 592–597.
8. Xu, H.; Chen, W.; Zhao, N.; Li, Z.; Bu, J.; Li, Z.; Liu, Y.; Zhao, Y.; Pei, D.; Feng, Y.; others. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. Proceedings of the 2018 World Wide Web Conference, 2018, pp. 187–196.
9. Chalapathy, R.; Chawla, S. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* **2019**.
10. Görnitz, N.; Kloft, M.; Rieck, K.; Brefeld, U. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research* **2013**, *46*, 235–262.
11. Kelleher, J.D. *Deep learning*; MIT Press, 2019.
12. Verner, A. LSTM Networks for Detection and Classification of Anomalies in Raw Sensor Data. PhD thesis, College of Engineering and Computing, Nova Southeastern University, 2019.
13. Gal, Y.; Islam, R.; Ghahramani, Z. Deep bayesian active learning with image data. International Conference on Machine Learning. PMLR, 2017, pp. 1183–1192.
14. Kirsch, A.; Van Amersfoort, J.; Gal, Y. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems* **2019**, *32*, 7026–7037.
15. Hyndman, R.J.; Wang, E.; Laptev, N. Large-scale unusual time series detection. 2015 IEEE international conference on data mining workshop (ICDMW). IEEE, 2015, pp. 1616–1619.
16. Shumway, R.H.; Stoffer, D.S. *Time series analysis and its applications with R Examples*, 4 ed.; Springer Texts in Statistics, Springer, 2017.
17. Chatfield, C.; Xing, H. *The Analysis of Time Series: An Introduction with R*, 7 ed.; Texts in Statistical Science, CRC Press, 2019.
18. Hyndman, R.J.; Athanasopoulos, G. *Forecasting: principles and practice*; OTexts, 2018.
19. Engle, R.F. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica: Journal of the econometric society* **1982**, pp. 987–1007.
20. Bollerslev, T. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics* **1986**, *31*, 307–327.
21. Fulcher, B.D.; Little, M.A.; Jones, N.S. Highly comparative time-series analysis: the empirical structure of time series and their methods. *Journal of the Royal Society Interface* **2013**, *10*, 20130048.
22. Fulcher, B.D.; Jones, N.S. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering* **2014**, *26*, 3026–3037.
23. Fulcher, B.D.; Jones, N.S. hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell systems* **2017**, *5*, 527–531.
24. O'Hara-Wild, M.; Hyndman, R.; Wang, E.; Cook, D.; Talagala, T.; Chhay, L. feasts: Feature Extraction and Statistics for Time Series (0.2.1), 2021. https://CRAN.R-project.org/package=feasts.
25. Hyndman, R.; Kang, Y.; Montero-Manso, P.; Talagala, T.; Wang, E.; Yang, Y.; O'Hara-Wild, M. tsfeatures: Time Series Feature Extraction (1.0.2), 2020. https://CRAN.R-project.org/package=tsfeatures.
26. Facebook. Kats, 2021. https://facebookresearch.github.io/Kats/.
27. Christ, M.; Braun, N.; Neuffer, J.; Kempa-Liehr, A.W. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing* **2018**, *307*, 72–77. doi:https://doi.org/10.1016/j.neucom.2018.03.067.
28. Barandas, M.; Folgado, D.; Fernandes, L.; Santos, S.; Abreu, M.; Bota, P.; Liu, H.; Schultz, T.; Gamboa, H. TSFEL: Time Series Feature Extraction Library. *SoftwareX* **2020**, *11*, 100456. doi:https://doi.org/10.1016/j.softx.2020.100456.

29. Lubba, C.H.; Sethi, S.S.; Knaute, P.; Schultz, S.R.; Fulcher, B.D.; Jones, N.S. catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery* **2019**, *33*, 1821–1852.
30. Robert, C.; William, C.; Irma, T. STL: A seasonal-trend decomposition procedure based on loess. *Journal of official statistics* **1990**, *6*, 3–73.
31. Henderson, T.; Fulcher, B.D. An Empirical Evaluation of Time-Series Feature Sets. *arXiv preprint arXiv:2110.10914* **2021**.
32. Ren, H.; Xu, B.; Wang, Y.; Yi, C.; Huang, C.; Kou, X.; Xing, T.; Yang, M.; Tong, J.; Zhang, Q. Time-series anomaly detection service at microsoft. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 3009–3017.
33. Hou, X.; Zhang, L. Saliency detection: A spectral residual approach. 2007 IEEE Conference on computer vision and pattern recognition. Ieee, 2007, pp. 1–8.
34. Vlachos, M.; Yu, P.; Castelli, V. On periodicity detection and structural periodic similarity. Proceedings of the 2005 SIAM international conference on data mining. SIAM, 2005, pp. 449–460.
35. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
36. Lavin, A.; Ahmad, S. Evaluating real-time anomaly detection algorithms–the Numenta anomaly benchmark. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). IEEE, 2015, pp. 38–44.