A Review of AutoML Software Tools for Time Series Forecasting and Anomaly Detection

Christian O'Leary[1], Farshad Ghassemi Toosi[1] and Conor Lynch[2]
[1]*Department of Computer Science, Munster Technological University, Cork, Ireland*
[2]*Nimbus Research Centre, Munster Technological University, Cork, Ireland*

Keywords: Machine Learning, Deep Learning, AutoML, Software, Time Series, Forecasting, Anomaly Detection.

Abstract: Time series exist across a plethora of domains such as sensors, market prices, network traffic, and health monitoring. Modelling time series data allows researchers to perform trend analysis, forecasting, anomaly detection, predictive maintenance, and data exploration. Given the theoretical and technical knowledge required to implement mathematical and machine learning models, numerous software libraries have emerged to facilitate the programming of these algorithms via automated machine learning (AutoML). Comparatively few studies compare such technologies in the context of time series analysis and existing tools are often limited in functionality. This review paper presents an overview of AutoML software for time series data for both forecasting and anomaly detection. The analysis considers 28 metrics that indicate functionality coverage, code maturity, and community support across 22 AutoML libraries. These aspects of software development are crucial for the uptake and utilisation of AutoML tools. This study proposes a means of deriving a functionality score for correlation analysis between variables such as lines of code, package downloads from PyPi, and GitHub issue completion rate. This review paper also presents an overview of AutoML library features which can facilitate informed decisions on which tools are most appropriate in various instances.

1 INTRODUCTION

The growing ubiquity of time series data and effective modelling approaches has co-evolved with the proliferation of sensing technologies (Lu and Xu, 2019), digitisation (Qiu et al., 2016), the development of programming tools (Abadi et al., 2015), and ongoing academic research. Time series modelling has use cases in many industries including agriculture (Zhang et al., 2020), computing (Wu et al., 2018) and energy (Kavanagh et al., 2017). Applied instances include point forecasting (Lynch et al., 2021), density forecasting (Bergmann et al., 2018), anomaly detection (Hundman et al., 2018), time series classification (Ismail Fawaz et al., 2019) and exploratory analysis (Gürtler and Paulsen, 2018). Methodologies range from traditional statistical models (Bergmann et al., 2018) to more modern approaches such as machine learning (ML). ML is a subtopic within Artificial Intelligence (AI) and is a highly complex field where models applied to time series data include Random Forests (Lago et al., 2018), K-Nearest Neighbours (KNN) (O'Leary et al., 2021), Support Vector Machine (SVM) ensembles (Lynch et al., 2019), shallow neural networks (NNs) (Zhang et al., 2020) and deeper NNs which are referred to by the term deep learning (DL) (O'Leary et al., 2021).

Ordinarily, the process of implementing a time series model requires theoretical knowledge of the algorithm itself, as well as proficiency in programming and debugging. These prerequisites often present a barrier to the field of time series analysis, restricting the range of implementable experiments without an in-depth ability of the extensive range of tasks required. Empirical ML analysis often includes data gathering, data exploration, data cleansing, data augmentation, feature engineering, feature selection, scaling, model hyperparameter search, model training, post-processing, (nested) cross-validation, model selection, model testing, statistical significance testing, and reporting (He et al., 2021). To the author's knowledge, existing libraries reviewed in this paper either exclude or only partially facilitate the majority of these steps. Therefore, considerable manual effort is required to implement a solution that incorporates ML or statistical modelling. For instance, while scikit-learn (Pedregosa et al., 2011) contains various model implementations, end-users are required to re-

O'Leary, C., Toosi, F. and Lynch, C.
A Review of AutoML Software Tools for Time Series Forecasting and Anomaly Detection.
DOI: 10.5220/0011683000003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 3*, pages 421-433
ISBN: 978-989-758-623-1; ISSN: 2184-433X

view, implement and test these functions in their respective code.

In response to the innate difficulty of programming for ML (Thessen, 2016), a wide range of automated ML (AutoML) libraries and frameworks have emerged (see Section 3). The research presented here evaluates the extent to which these tools fulfil the needs of ML researchers with a particular focus on time series forecasting and anomaly detection. The analysis compares 22 Python-based libraries regarding the functionalities these tools offer along with the level of engagement from their developers and the programming community at large. This comparison aims to provide a benchmark of quantitative and qualitative metrics for selecting appropriate libraries based on domain-agnostic information prior to experimentation.

Section 2 provides an overview of time series forecasting and anomaly detection using traditional, ML and DL approaches. Section 3 evaluates existing Python-based AutoML libraries comparing the suitability of each tool for forecasting and anomaly detection. The AutoML tools are discussed and contrasted in Section 4 before conclusions are outlined in Section 5.

2 BACKGROUND

A survey of ML for big data processing (Qiu et al., 2016) revealed that ML, as a field of research, has exploded in popularity for domains such as medicine, Internet of Things (IoT) systems, social media, cybersecurity, computer vision, and recommender systems. A popular application of ML modelling within these domains is time series analysis, which includes forecasting (Borovykh et al., 2018) and anomaly detection (Wu et al., 2018).

2.1 Time Series Data

Time series vary significantly in attributes depending on their source, even within the same organisation (Chatterjee et al., 2022). Time series datasets can be decomposed into their respective base components which often envelop *trend*, *seasonality*, *cyclicity*, and *irreducible noise*. Trend refers to a long-term increase or decrease in the mean series value. A time series with a stable, unchanging trend is referred to as stationary. Time series seasonality denotes the regular frequencies in the data. Time series may have multiple frequencies, e.g. temperatures follow both daily and annual patterns. Cyclicity refers to cycles that occur irregularly; the period between cycles may vary. Finally, irreducible noise or white noise refers to naturally occurring minor fluctuations in points of data. A time series may also contain anomalies. There is no universal consensus for the definition of an anomaly, possibly because the distribution of anomalies is typically difficult to determine given their sparsity (Goix, 2016). Anomalies are not guaranteed to be evenly distributed and may occur consecutively in bursts (Chatterjee et al., 2022).

2.2 Modelling Time Series Data

Exploratory analysis can be conducted on time series data using statistical methods and visual examination via plots such as autocorrelation function (ACF) and partial autocorrelation function (PACF). Inherently, visual scrutiny of these plots introduces subjective error (O'Leary, 2020). Using these plots, a feature set can be selected as a continuous window of values (Lynch et al., 2019) or a smaller cohort of highly correlated features (O'Leary, 2020). This process can improve model accuracy and assist in preventing overfitting (He et al., 2021). A feature set can be augmented by the inclusion of engineered features such as mapping features to Sine or Cosine waves (Ugurlu et al., 2018) or by using Fourier Transforms (Ghaderi and Kabiri, 2012). Feature sets can be optimised further using dimensionality reduction methods such as Principal Component Analysis (PCA) (Qiu et al., 2016). While time series data is commonly associated with numeric data sources such as sensors (Ahmad et al., 2017), prices (Ugurlu et al., 2018) and traffic speeds (Kim et al., 2018), it is also possible to transform other data formats (e.g. text) into time series data using various transformations (Zhang et al., 2019).

Numerous studies have compared implementations for ML, algebraic and statistical models with each family of algorithms proving to be advantageous under different conditions (Javeri et al., 2021). Curve fitting and statistical models are computationally inexpensive and can yield explainable results (Thessen, 2016). In contrast, ML models are difficult to interpret for structural reasons, resulting in explainable AI being an active field of research (Amarasinghe et al., 2018). They also suffer from increased computational complexity and slow fitting times (Santurkar et al., 2019), necessitating optimisation strategies to limit the number of model instances trained (He et al., 2021). As none of the modelling approaches achieved supremacy across all problems in forecasting and anomaly detection, both ML and non-ML models are examined in this review paper.

2.3 Mathematical Models

In a review of algebraic, semiparametric, statistical and conventional ML models applied to milk yield forecasts, Zhang et al. (2018) acknowledged the difficulty in comparing results across different use cases where datasets and metrics differ. Cubic splines have been found to be effective forecasting models, although they are highly sensitive to outliers (Zhang et al., 2018). For higher dimensional data, Multiple Linear Regression (MLR) models can be effective as forecasting models (Kaytez et al., 2015). Statistical models can be fitted to time series data using principles of ACF/PACF analysis, autoregression, moving averages, and smoothing or a combination thereof. Other mathematical models include Autoregressive (AR) (Lago et al., 2018), Dynamic Regression (DR) (Lago et al., 2018), Holt-Winters (Kavanagh et al., 2017) and hybrid models of the aforementioned.

Lago et al. (2018) compared 27 statistical and ML models for electricity price forecasting observing "a clear division" between the two families of algorithms. The ML models achieved statistically significantly better accuracies than their counterparts, but other studies (Kavanagh et al., 2017) have demonstrated cases where statistical models can outperform ML models. The time taken to fit ML models can be considerable (O'Leary and Lynch, 2022) and they are less interpretable than statistical approaches (Thessen, 2016).

Time series models can be applied to anomaly detection by measuring the residuals of predictions versus newly observed data (Smith, 2017). Where the residual exceeds a specified threshold, an anomaly is assumed. Statistical error thresholds for anomaly detection can be static (LinkedIn, 2022) or dynamically calculated (Choudhary et al., 2017). Anomaly detection scores can also be calculated via PCA (Zhang et al., 2019) or Robust PCA (RobustPCA or RPCA) (Choudhary et al., 2017). To benchmark model runtimes for anomaly detection, Choudary et al. (2017) implemented a series of statistical, ML, and time series-based mathematical models, finding that models using thresholds calculated from streaming percentiles, mean, median, median absolute deviation or autoregressive models may be preferable to ML under extreme time constraints.

2.4 Machine Learning Models

ML is a data-driven approach where models attempt to iteratively self-optimise from observed data. Time series forecasting is supervised regression as past observations are used to train the model. Anomaly detection is supervised classification if labels are available and unsupervised otherwise. In practice, anomaly detection modelling must often be unsupervised as anomaly labels are rarely available in production (Goix, 2016). Instead of relying on statistical tests, ML model hyperparameters are optimised using manual specification or algorithms such as grid search, randomised search, Genetic Algorithms (GAs), etc. (He et al., 2021). ML models are typically fitted on training data and evaluated on test data (which simulates out-of-sample data). For ML modelling, a third partition called a validation set is often introduced to rank trained model instances that are using different hyperparameters before the shortlisted model is finally evaluated on the test set.

Ridge, Lasso, Elastic Net, and Bayesian Ridge expand on traditional MLR models by adding penalty terms to tune coefficients (Nguyen et al., 2020; O'Leary et al., 2021). Such models adjust their parameters using a learning rate α. For ML models, the value of a learning rate needs to be chosen carefully (Ioffe and Szegedy, 2015). Overly large learning rates may cause model parameters may fail to converge, while small learning rates may slow training unnecessarily (Santurkar et al., 2019).

Support Vector Regression (SVR) models have been used for forecasting as standalone models (Grimes et al., 2014) and as part of ensembles such as K-SVM-SVR (Lynch et al., 2019). The K-Nearest Neighbour (KNN) model has been successfully used for forecasting electricity prices (O'Leary et al., 2021) and milk volumes (O'Leary and Lynch, 2022). Decision Trees are used for forecasting both as standalone models (O'Leary and Lynch, 2022), and in ensembles such as Random Forests (Lago et al., 2018) or Extremely Randomised Trees (Extra Trees) (O'Leary et al., 2021).

Dense neural networks (DNNs) are neuroscientifically inspired models that consist of layers of neurons (Singh and Mohanty, 2015). The inputs to each neuron are multiplied by weights that the model learns with a learning rate using algorithms such as backpropagation. NNs have grown increasingly 'deep' in terms of layers (He et al., 2021), although smaller networks remain effective for time series forecasting (Nguyen et al., 2020). The increased network depth of DNNs allows them to learn complex and non-linear relationships at the cost of longer training times (Ioffe and Szegedy, 2015) and less explainable outputs (Yu et al., 2019).

A limitation of DNNs is that they do not retain any internal state based on memories of recently observed data beyond their learned parameters. Recurrent Neural Networks (RNNs) solve this problem using a neu-

ron's output as an additional input connection. This forms a feedback loop that makes RNNs and their relatives particularly adept in dealing with sequential data, e.g. time series (Ugurlu et al., 2018) and natural languages (Lynch et al., 2020). Other forms of RNN include Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014).

Conventional Neural Networks (CNNs) consist of convolutional layers, pooling layers and finally one or more densely connected layers. CNNs are most commonly used for higher dimensional data which is often required for image analysis (Lynch et al., 2020), but they can be adapted to time series forecasting. Firstly, the time series data can be formatted into a 2D matrix and passed as an image (Kim et al., 2018). Alternatively, CNNs can use 1-dimensional filters for feature identification (O'Leary, 2020).

Autoencoders (AE) are a type of neural network that can be used in both supervised and unsupervised modelling and are commonly used in hybrid model architectures (Wang et al., 2017). AE models include a bottleneck that forces the network to learn a compact representation of data. AE models can be used for unsupervised anomaly detection by leveraging a reconstruction loss metric as a proxy for anomaly score (Zhao et al., 2022).

In the field of anomaly detection, it is widely acknowledged that labelled data are generally unavailable, necessitating the use of unsupervised or semi-supervised approaches (Ahmad et al., 2017). Alternatively, some researchers have used generative models to create synthetic labels for training models, although these models are less effective at detecting real anomalies (Chatterjee et al., 2022). Some proxy metrics for estimating the ranking of anomaly detection models without labels via Mass-Volume and Excess-Mass curves have been proposed, but follow-up research has been sparse and inconclusive (Warzynski et al., 2021).

Isolation Forests are unsupervised tree-based models that work by partitioning the dataset under the assumption that anomalies are uncommon and are measurably different from normal data. Time series bitmaps are effective for unsupervised anomaly detection and visualisation. Time series bitmaps process sliding windows of data of a fixed length which is set via hyperparameter. One-Class SVM (OCSVM) is an unsupervised variant of the traditional SVM model that assumes one normal and one anomalous class.

It should be noted that standard regression models such as NNs, KNR, SVR, etc. can also be adapted for unsupervised anomaly detection if combined with a threshold algorithm. Where a time series forecasting model's predictions experience an error beyond a threshold, an anomaly can be assumed. Such thresholds can be static, based on the modelling error (Hundman et al., 2018), or learned as hyperparameters (Chatterjee et al., 2022).

Choudhary et al. (2017) found ML models to be superior in terms of accuracy to statistical models, albeit at the cost of greater runtimes for both anomaly detection and forecasting. The disparity in accuracy is attributed to a lack of robustness in statistical models to concept drift.

2.5 Modelling in Practice

The Python programming language (Python, 2022) has a rich ecosystem of open-source libraries and frameworks available on public Git repository sites such as GitHub (Github, 2022) and the Python Package Index (PyPI, 2022). A 2021 Stack Overflow (SO) survey (StackOverflow, 2021) of over 80,000 programmers indicated Python as the prevailing ML programming language. The survey reported that 48.24% of programmers use Python while other programming languages used for scientific computing such as R and MATLAB were used by 5.07% and 4.66% of users respectively. Python libraries used for ML can be high-performance as they are built on C libraries and can run models on Graphics Processing Units (GPU) using Compute Unified Device Architecture (CUDA) (NVIDIA, 2017).

Scikit-learn is a popular ML library containing many model implementations. The scikit-learn maintainers do not recommend scikit-learn for training NNs at scale as the library is not CUDA compatible. TensorFlow is a framework for constructing NNs via lower-level tensor manipulation (Abadi et al., 2015). TensorFlow is compatible with CUDA enabling NNs to be trained using NVIDIA GPUs which significantly reduces training times. TensorFlow also incorporates the Keras library (Francois, 2022). Other NN frameworks include PyTorch (Paszke et al., 2017) and MXNet (Chen et al., 2015). Anomaly detection libraries include PyOD (Zhao et al., 2022), Luminol (LinkedIn, 2022) and Prophet (Facebook, 2022). The statsmodels library contains functionality for statistical modelling including ARIMA, SARIMA, etc. (Seabold and Perktold, 2010).

3 AutoML LIBRARIES AND FRAMEWORKS

AutoML can incorporate many phases of a typical ML pipeline including data cleaning, data augmentation,

feature selection, model training, HO and AO (Bahri et al., 2022). AutoML systems tend to be highly configurable, although increased configuration can be overwhelming for some practitioners (Thessen, 2016).

AutoML may use data augmentation which involves generating synthetic data to enlarge a dataset. Data augmentation is typically used for image data (He et al., 2021) but has been successfully applied to time series data by Javeri et al. (2021). Javeri et al. effectively doubled the size of a COVID dataset in their AutoML pipeline by transitioning from daily to half-day forecasts; half-day values were imputed using a Seasonal Auto-Regressive Integrated Moving Average (SARIMA) model. This improved the overall accuracy of their model and has a regularisation effect.

AutoML pipelines use a variety of model optimisation methods that can be categorised into architecture optimisation (AO) and (HO). AO is specific to NN models and is synonymous with Neural Architecture Search (NAS). It refers to the automatic optimisation of a NN's architecture which can consist of many low-level operations, e.g., convolution, pooling and concatenation (He et al., 2021). A wide variety of NAS methods have emerged including reinforcement learning, gradient descent, and evolutionary algorithm approaches. It should be noted that NAS is still an open problem as there are cases where sophisticated AO algorithms may not outperform randomised searches (He et al., 2021). There are many advanced HO methods including contracting grid searches (Hesterman et al., 2010), randomised searches (Bergstra and Bengio, 2018) and Bayesian approaches. Hyperparameter searches are inherently limited by their specified search spaces (Bahri et al., 2022). HO and AO can occur jointly or separately via a two-stage process (He et al., 2021). In the two-stage approach, NAS is conducted using a fixed set of hyperparameters. Having achieved a local optimum, hyperparameter tuning is then applied to the model (He et al., 2021). The majority of research on NAS has, however, traditionally focused on non-time series-related problems such as image classification (He et al., 2021). As a result, some researchers noted that conventional NAS algorithms are not appropriate for anomaly detection AutoML workflows (Li et al., 2020a).

Given one dataset in isolation, it is computationally costly to train and test many models, but the availability of multiple datasets allows for the use of meta-learning (Feurer et al., 2021). Meta-learning involves recording the performances of models across various datasets to learn to predict (via a meta-learner model) what models are suitable for an unseen dataset (Bahri et al., 2022). By predicting what model and hyperparameters should be employed, costly search space exploration can be partially avoided (Bahri et al., 2022). For time series data, meta-features may include statistics such as the number of instances, autocorrelation, seasonality, or trend (Chatterjee et al., 2022). The meta-learner used for model selection used can vary; Feurer et al. (Feurer et al., 2021) used a policy selector, while Chatterjee et al. (Chatterjee et al., 2022) used a Random Forest classifier and one multi-task NN per algorithm. Multiple studies have espoused AutoML via meta-learning as a means of rapidly training models more effectively with respect to time than manually specified algorithms (Chatterjee et al., 2022). However, creating a training dataset for meta-learners is expensive (Feurer et al., 2021) and the effectiveness of meta-learning is tied to the level of similarity between encountered datasets (Bahri et al., 2022). For anomaly detection, it is possible to employ unsupervised models, but existing meta-learner models are still based on supervised algorithms (Bahri et al., 2022).

In practice, AutoML libraries vary in functionality, scope, and community engagement as presented in Tables 1 and 2. The 22 AutoML libraries selected include some of the most popular open-source Python-based AutoML libraries. **AdaNet** is a TensorFlow-based AutoML framework for learning NN-based ensembles (Cortes et al., 2017). **AutoGluon** is a library for deep learning and ensembles for text, image, and tabular data (Erickson et al., 2020). **AutoKeras** (or Auto-Keras) is a library that automates the optimisation of a selection of Keras models (Jin et al., 2019). **Auto-PyTorch** jointly performs both hyperparameter and architecture optimisation of DL models (Zimmer et al., 2021). The **auto-sklearn** library is an AutoML library built on scikit-learn using meta-learning (Feurer et al., 2021). **AutoTS** is a library for developing statistical, ML and DL forecasting models at scale (Catlin, 2022). **ETNA** is a time series AutoML framework with a wide variety of forecasting functions and some for anomaly detection (Tinkoff.AI, 2022). **EvalML** is a general AutoML framework with statistical and ML models with some time series functionality (Alteryx, 2022). **FEDOT** automates ML pipeline development using evolutionary algorithms (Nikitin et al., 2022). The Fast and Lightweight AutoML Library (**FLAML**) is a lightweight AutoML library with HO (Wang et al., 2021). H2O is an open-source AutoML framework for distributed learning (H2O, 2022). **Hyperopt-sklearn** is a Python library that combines the functionality of Hyperopt and scikit-learn for automatic

model optimisation (Komer et al., 2019). **Kats** (Kits to analyse time series) is a library specifically for time series analysis including forecasting (Facebook, 2022), anomaly detection, (Meta, 2022) and meta-learning (Chatterjee et al., 2022). **LightAutoML** is a general AutoML library for supervised classification and regression (Vakhrushev et al., 2022). **Ludwig** is a general-purpose toolbox for ML modelling on tabular data for problems such as classification, forecasting, natural language processing, etc. (Molino et al., 2019). Active Anomaly Detection with Meta-Policy (**Meta-AAD**) uses a meta-policy with reinforcement learning to optimise anomaly detection models (Zha et al., 2020). Outlier Detection via Meta-Learning (**MetaOD**) is a Python tool using meta-learning for outlier detection (Zhao et al., 2021). **MLBox** is an AutoML Python library with both optimisation and preprocessing features (De Romblay, 2022). The **mljar** library is an AutoML Python package for tabular data with options for (paid) remote learning (Plonska and Płoński, 2021). **PyCaret** is a low-code ML library for generic ML modelling including some functionality for time series modelling and anomaly detection (Ali, 2020). Python system for Outlier Detection with Database Support (**PyODDS**) is a supervised outlier detection library with database support (Li et al., 2020b). Tree-based Pipeline optimisation Tool (**TPOT**) is a Python AutoML tool that uses Genetic Algorithms for model optimisation (Olson et al., 2016).

Statistics from the 19th of August 2022 are presented for each software tool in Table 1. These fluctuate given the dynamic nature of software development. The metrics in Table 1 present an approximate view of the maturity, popularity, and community engagement behind each project. The lines of code metric counts lines of Python code. The Stars, Forks, and Commits correspond to their respective values on GitHub. The last update time corresponds to the time since the last commit on the main/master branch. GitHub Issues correspond to user-submitted bug reports, feature requests, questions, and other issues. The issue completion rate is calculated as the number of closed issues divided by the total. Most of the GitHub-related figures are subject to regular change but provide some quantifiable estimates of both developer and user engagement. As PyPi is the main repository for publicly available Python packages, the number of package downloads from PyPi is also included as both a total and from the previous month from PyPi Stats (Flynn, 2022). SO is a Q&A website with over 21 million recorded questions and millions of monthly visits, making it one of the most prominent programming resources available (StackOverflow, 2021). The SO tag count represents the number of user-submitted questions relating to a particular software package. It should be assumed that this count will not include questions where the library was not tagged.

Table 2 compares the libraries in terms of selected functionality. Resource limits refer to the capability of the software to impose constraints via limits on overall runtime, CPU, RAM, or GPU usage. Most of the libraries use a parameter specifying the number of iterations or models, but these do not guarantee that the simulations will end within a specific timeframe. Preprocessing includes features such as aggregation, feature scaling, imputation, etc. Model/output visualisation refers to the presence of incorporated classes or functions to visualise or tabulate results automatically. Data augmentation may include resampling methods, generative NNs or any method for extending a dataset with new data. Feature engineering involves the creation of feature sets through extrapolation or collecting historical values to use as a sliding 'window'. Feature selection involves the shortlisting of features from the prospective feature set based on metrics such as feature importance scores. The availability of statistical, ML, and DL models are also listed with an additional 'anomaly detection' column presented to clarify which of these libraries offer integrated anomaly detection. HO and AO indicate the model self-tuning capabilities of the software. The following time series-specific functionality are also examined: time-based cross-validation (TCV), time series formatting features, and multi-output regression predictions which are required for forecasting beyond one period. Finally, the presence of built-in pipeline Serialisation functionality is recorded. Features are ignored where virtually all of the libraries have such capabilities, thus making the comparison redundant. Feature implementation was determined by scanning existing library documentation for the relevant functionalities. Where a feature was not outlined in the documentation, the Python source code was also manually examined for any undocumented functionality. Where a library has dedicated AutoML modules, only those modules are considered for examination. For example, the H2O library has some feature selection functionalities that may not be supported by the H2OAutoML class.

4 DISCUSSION

Section 3 describes 22 AutoML libraries using 28 technical and non-technical characteristics. The feasibility of examining these variables via statistical

Table 1: Statistics on AutoML Tools.

Software Name	Lines of Code	Repository Activity				GitHub Issues				PyPi Downloads		SO Tag Count
		Stars	Forks	Commits	Last Update	Open	Closed	Total	Close Rate	Last Month	Total	
AdaNet	28,106	3,417	531	440	2 years	63	49	112	0.44	659	158,503	4
AutoGluon	86,748	4,729	621	1,069	6 hours	152	582	734	0.79	118,763	2,335,813	0
AutoKeras	18,513	8,586	1,373	1,322	2 days	94	743	837	0.89	41,271	837,509	60
Auto-PyTorch	63,141	1,740	216	249	29 days	37	193	230	0.84	920	15,578	0
auto-sklearn	62,864	6,440	1,175	2,742	1 hour	108	806	914	0.88	35,344	838,285	15
AutoTS	28,639	560	51	233	2 months	10	40	50	0.80	39,163	923,229	0
ETNA	38,636	562	53	467	2 hours	54	295	349	0.85	5,507	37,426	0
EvalML	92,561	525	71	1,815	7 minutes	253	1,414	1,667	0.85	7,948	169,491	1
FEDOT	50,474	427	48	635	2 hours	90	261	351	0.74	1,160	26,882	0
FLAML	26,202	2,023	296	369	4 days	44	191	235	0.81	29,579	632,880	0
H2O	185,282	5,939	1,924	31,136	23 hours	65	6,161	6,226	0.99	428,906	17,173,047	1,857
Hyperopt-sklearn	11,730	1,362	253	399	3 months	70	53	123	0.43	8,422[a]	1,167,652	0[c]
Kats	83,507	3,901	392	553	1 day	34	100	134	0.75	44,990	607,668	4
LightAutoML	35,595	231	12	28	11 days	0	3	3	1	3,372	115,869	0
Ludwig	91,121	8,475	1,002	2,806	15 hours	188	624	812	0.77	2,429	124,023	19
Meta-AAD	1,194	34	10	6	15 months	0	1	1	1	0	0	0
MetaOD	1,685	106	17	38	6 months	3	3	6	0.50	252	7,775	0
MLBox	22,043	1,343	270	1,121	2 years	18	74	92	0.80	6,469	177,362	0
mljar	22,043	2,023	285	1,010	3 days	95	398	493	0.81	8,288	797,714	3
PyCaret	74,142	6,132	1,411	3,997	6 days	271	1,429	1,700	0.84	902,608	6,650,713	124
PyODDS	5,215	199	34	115	2 years	6	0	6	0	155[b]	14,240	0
TPOT	12,233	8,705	1,498	2,390	21 days	252	605	857	0.71	47,993	1,650,310	9

SO = Stack Overflow. *Data accurate as of the 19th of August 2022.* [a] Might not be maintained, [b] Not updated since first upload, [c] There are, however, 118 matches for "hyperopt" some of which have clear references to the package.

analysis was evaluated by examining the variables with regard to the linearity of relationships, outliers, and monotonicity. The relationships between each of the 12 variables in Table 1 were determined to be non-linear and some variables were not normally distributed or had outliers. Thus, Spearman rank correlation was selected to measure the monotonicity of the relationship between variables. To facilitate the incorporation of the 16 variables in Table 2, feature categorisations were converted into an ordinal encoding: 0 indicates a feature is not implemented, 0.5 indicates some implementation with limitations and 1 indicates a full implementation for general usage in an AutoML pipeline. These functionality scores are subjective and are used for comparative purposes only. For example, the auto-sklearn library receives a score of 0.5 for DL as it only supports MLPs via scikit-learn. The scores for each library or framework were summed and included in the Spearman rank correlation analysis which is presented in Table 3 (a p value of 0.05 is assumed for all statistical testing). The Stack Overflow (SO) Tag Count was excluded because the data was sparse as many of the packages do not seem to have an associated tag on SO.

The analysis presented in Table 3 offers insights into the relationships between the aforementioned variables. Lines of code strongly correlate with the functionality score (0.77, p=$3.03e-05$). This suggests that repository size could be useful as a proxy metric for total functionality coverage, although this does not necessarily indicate the universal applicability of functionality to all users. For example, the H20 library is by far the largest tool with 185,000 lines of code, but some of the functionality is unrelated to time series analysis, e.g. image preprocessing. As this study focuses on time series forecasting and anomaly detection, H2O's functionality score is lower than smaller but more specialised libraries such as ETNA. The lines of code metric has a moderate correlation with the number of GitHub commits (0.58, p=0.005) which implies a large degree of variability in commit size. There is a moderate correlation between functionality score and the number of closed GitHub issues (0.54, p=0.0091) and the total number of GitHub issues (0.52, p=0.0122). While many closed issues may be an indicator of functionality, the correlation is weaker than with lines of code. A possible reason is that not all closed issues result in code fixes or new features. Many GitHub issues relate to user questions, documentation changes or are simply closed by maintainers. The 0.99 correlation (p=$6.55e-17$) between total issues and closed issues may indicate redundancy between the two variables. Similar redundancy may exist between the PyPi download metrics;

Table 2: AutoML Library/Framework Functionality.

Software Name	Resource Limits	Pre-processing	Serialisation	MOV	Data Augmentation	Feature Engineering	Feature Selection	Statistical Models
AdaNet	No	No	Yes	Yes[2]	No	No	No	No
AutoGluon	Time, presets	Yes	Yes	Yes	Yes	Yes	Yes	No
AutoKeras	No[1]	Yes	SO	No	Images Only	Yes	No	No
Auto-PyTorch	Time, memory	Yes	Yes	Yes	Resampling	Yes	Not for TS	Not for forecasting
auto-sklearn	Time, memory, CPU	Yes	SO	No	Resampling	Yes	Yes	No
AutoTS	No	Yes	Yes	Yes	No	Yes	No	Yes
ETNA	No	Yes	No	Yes	Resampling	Yes	Yes	Yes
EvalML	Time	Yes	Yes	Yes	No	Yes	Yes	Yes
FEDOT	Time	Yes	Yes	Yes	No	Yes	No	Yes
FLAML	Time	Yes	SO	No	Not for TS	No	No	Yes
H2O	Time, memory	Yes	Yes	Yes	Not via API[3]	Yes	No	Limited[6]
Hyperopt-sklearn	Time	Yes	No	No	No	Yes	No	No
Kats	No	Yes	Yes	Yes	Resampling[4]	Yes	Yes	Yes
LightAutoML	Time	Yes	SO	Yes	No	Yes	Yes	No
Ludwig	Time	Yes	Yes	Yes	No	Not for TS	No	No
Meta-AAD	No	Yes	Yes	No	No	Yes	N/A[5]	No
MetaOD	No	Yes	Yes	No	No	Yes	N/A[5]	Yes
MLBox	No	Yes	No	No	No	Yes	Yes	No
mljar	Time, modes	Yes	Yes	Yes	No	Yes	Yes	No
PyCaret	Time	Yes	Yes	Yes	No	Yes	Not for TS	Yes
PyODDS	No	Yes	No	Yes	No	No	No	A.D. only
TPOT	Time	Yes	SO	No	No	Yes	Yes	No

AD = Anomaly Detection, MOV = Model Output Visualization, SO = returns Serialisable Object, TS = Time series. [1] Uses trials, [2] TensorBoard, [3] Possible via GUI, [4] Some resampling by statistical models and for the meta-learner, [5] uses meta-features, [6] Not in AutoML module, available otherwise

Software Name	ML Models	DL Models	AD	HO	Architecture Optimisation	TCV	Formats Data for TS prediction	Multi-output prediction
AdaNet	Yes[7]	Yes	No	No	Yes	No	No	Yes
AutoGluon	Yes	Yes	No	Yes	Yes	No	Yes	Yes
AutoKeras	No	Yes	No	Yes	Yes	No	Yes	Yes
Auto-PyTorch	No forecasting	Yes	No	Yes	Yes	No	Yes	Yes
auto-sklearn	Yes	MLP*	No	Yes	MLP only	No	No	Yes
AutoTS	Yes	Yes	No	Yes	No	Yes	Yes	Yes
ETNA	Yes	Yes	Outliers only	Yes	No	Yes	Yes	Yes
EvalML	Yes	No	No	Yes	No	Yes	Yes	Yes
FEDOT	Yes	Yes	No	Yes	No	Yes	Yes	Yes
FLAML	Yes	Yes	No	Yes	No	No	No	Yes
H2O	Yes	Yes	Supervised	Yes	No	No	Yes	Yes
Hyperopt-sklearn	Yes	MLP*	No	Yes	MLP only	No	Yes	No
Kats	Forecasting only	Forecasting only	Yes	Yes	Limited	No	Yes	Yes
LightAutoML	Yes	No	No	Yes	No	Yes	Yes	No
Ludwig	No	Yes	Binary Supervised	Yes	Yes	No	Yes	Yes
Meta-AAD	No	Yes	Semi-supervised	No	No	No	No	No
MetaOD	Yes	No	Semi-supervised	Yes	No	No	No	No
MLBox	Yes	Encoders only	No	Yes	No	No	No	No
mljar	Yes	MLP*	No	Yes	MLP only	Custom available	No	No
PyCaret	Yes	MLP*	Yes	Yes	MLP only	Yes	Yes	Yes
PyODDS	AD only	AD only	Supervised	Yes	Yes	No	Yes	No
TPOT	Yes	Yes[8]	No	Yes	No	No	No	No[9]

AD = Anomaly Detection, HO = Hyperparameter Optimisation, MLP = scikit-learn's MLP class, TCV = Time-based Cross-Validation, TS = Time series. [7] Linear Estimator, [8] They recommend custom validation, [9] In progress (https://github.com/EpistasisLab/tpot/pull/1001)

the monthly and total download figures have a correlation of 0.93 (p=0.0007).

The GitHub issue close rate was included as a potential proxy metric for developer engagement, but it did not statistically significantly correlate with the other metrics in Table 3. One potential reason for the lack of correlation is that some of the smaller libraries have GitHub issue completion rates with high variability due to having few issues. The other GitHub-related metrics of stars, forks, commits, open issues, and closed issues all correlate strongly with each other. There is a statistically significant moderate correlation between PyPi downloads from the last month and the functionality score (0.47, p=0.0273). The relative weakness of the correlation may be caused by the discrepancy between the time series-based functionalities evaluated in this review versus the needs of the broader research community, e.g. natural language processing, and image classification. The starting year of a project on PyPi was originally considered as a metric but was excluded due to noisy data; some packages were renamed or deprecated.

The H2O library has the largest codebase with approximately twice as many lines of code as the second largest tool, i.e. EvalML. H2O also has the most commits (>31,000), the largest GitHub issue completion rate (>98%), and the greatest number of total downloads from PyPi (>17 million) although PyCaret has the most downloads from the previous month (>900,000).

Table 2 compares the 22 libraries in terms of functionality. While this study weighs features equally, users may value features differently, which would result in varying functionality scores. For example, some may prefer statistical to DL models for time-sensitive applications, whereas other users may only use ML and DL models if their data experiences concept drift. Under Resource Limits, it can be observed that libraries offer varying constraints on model exploration via input parameters, e.g. not all libraries support time limits but may instead rely on specifying the number of iterations which is less precise. H2O, auto-sklearn, and Auto-PyTorch also offer memory limits which may prove beneficial for hardware-constrained environments such as microcontrollers. All of the libraries except for AdaNet provide some level of pre-processing. Larger libraries are not necessarily more popular amongst users as suggested by the lack of a statistically significant correlation between lines of code and total downloads.

Data augmentation functions for time series are uncommon except via resampling. While feature selection is not supported by many of the libraries, it is implicitly irrelevant for those using meta-features (e.g. Meta-AAD, MetaOD). Some libraries such as H2O do offer feature selection outside of their AutoML functionality. unsupervised anomaly detection models are atypical although supervised implementations are more commonly supported (e.g. Ludwig and PyODDS). Support for fully unsupervised anomaly detection via metrics such as Excess-Mass is non-existent. While PyCaret offers unsupervised anomaly detection models, these require a *contamination* hyperparameter to be manually specified which may not be possible for unseen data. The meta-learning module in Kats can simulate labelled data but this may be less effective than modelling using real data (Chatterjee et al., 2022).

Most of the libraries do not support AO outside of scikit-learn's CPU-based MLP classes despite NAS being a popular research topic (He et al., 2021). Conversely, statistical models are less prominent in AutoML papers but there are several AutoML libraries that offer implementations of statistical models in some capacity (e.g. PyCaret and Kats). Most libraries do not support time-based cross-validation.

5 CONCLUSIONS

The prevalence of time series data has motivated research of statistical and ML approaches for forecasting and anomaly detection. Time series data is complex to model due to temporal correlations, a lack of anomaly labels, high dimensionality from lagged data, and the sheer volume of data being produced. Statistical, ML and DL approaches have all been shown to be effective time series models under different experimental conditions (Javeri et al., 2021). AutoML approaches have benefited from increased attention from the research community (Xu et al., 2017), but there are still open problems around efficiency compared to more basic approaches (He et al., 2021). While AutoML has been applied to time series forecasting and anomaly detection, most AutoML research appears to have been applied to more popular research problems such as image and language modelling (He et al., 2021), necessitating a comprehensive review of how effective existing AutoML tools are for time series modelling.

The Python programming environment contains many software libraries that can be used for effective data preprocessing, feature selection, ML and DL modelling, etc. As AutoML tools offer varying degrees of automation and relevancy to time series modelling, this paper presents a review of 22 such libraries across 28 metrics. The lines of code metric correlates strongly with a functionality score de-

Table 3: Spearman Rank Correlations on AutoML Tool Statistics.

Metric	Lines of Code	Repository Activity			GitHub Issues				PyPi Downloads		Func. Score
		Stars	Forks	Commits	Open	Closed	Total	Close Rate	Last Month	Total	
Lines of Code	-	0.38	0.41	**0.58**	**0.47**	**0.65**	**0.61**	0.29	0.42	0.32	**0.77**
Stars	0.38	-	**0.96**	**0.77**	**0.70**	**0.72**	**0.73**	0.04	**0.69**	**0.70**	0.16
Forks	0.41	**0.96**	-	**0.84**	**0.70**	**0.77**	**0.79**	0.05	**0.74**	**0.77**	0.17
Commits	**0.58**	**0.77**	**0.84**	-	**0.84**	**0.92**	**0.91**	0.15	**0.64**	**0.64**	0.38
Open Issues	**0.47**	**0.70**	**0.70**	**0.84**	-	**0.86**	**0.89**	-0.01	**0.54**	**0.58**	0.40
Closed Issues	**0.65**	**0.72**	**0.77**	**0.92**	**0.86**	-	**0.99**	0.36	**0.66**	**0.60**	**0.54**
Total Issues	**0.61**	**0.73**	**0.79**	**0.91**	**0.89**	**0.99**	-	0.25	**0.67**	**0.63**	**0.52**
Close Rate	0.29	0.04	0.05	0.15	-0.01	0.36	0.25	-	0.19	0.08	0.25
Downloads Last Month	0.42	**0.69**	**0.74**	0.64	**0.54**	**0.66**	**0.67**	0.19	-	**0.93**	**0.47**
Total Downloads	0.32	**0.70**	**0.77**	**0.64**	**0.58**	**0.60**	**0.63**	0.08	**0.93**	-	0.27
Functionality Score	**0.77**	0.16	0.17	0.38	0.40	**0.54**	**0.52**	0.25	**0.47**	0.27	-

Functionality scores are derived from Table 2: 0=not implemented, 0.5=partially implemented, 1=fully implemented. Statistically significant ($p \leq 0.05$) results are presented in **bold** and strong correlations are underlined. Correlations (**?**): <0.1 = Negligible, 0.1-0.39 = Weak, 0.4-0.69 = Moderate, 0.7-0.89 = Strong, ≥0.9 = Very Strong

rived from a quantitative analysis of features (0.77, p=$3.03e-05$). Some other metrics proved to be less useful for the correlation-based analysis employed in this study. For example, the GitHub issue completion rate did not correlate significantly with any other metric. The repository-related metrics such as the number of stars, forks, etc. correlated most strongly with each other which may suggest redundancy. A limitation of the approach taken in this work is that the method of scoring functionality is manual and is therefore time-consuming and open to human error.

AutoML libraries are lacking in terms of fully unsupervised anomaly detection using metrics such as Mass-Volume and Excess-Mass curves (Goix, 2016). Approaches that use unsupervised models rely on synthetic data or manually specified hyperparameters indicating the proportion of expected anomalies.

The presented evaluation could be deepened by the use of automatic code quality scanning tools such as SonarQube (SonarQube, 2022) or GitHub's (Github, 2022) Dependabot for identifying security issues, bugs, test coverage, portability issues, and cyclomatic complexity. The measures chosen in this study do not directly capture the documentation quality of each software tool, but clone detection and coverage analysis have been proposed in other studies (Wingkvist et al., 2010). The number of academic citations relating to a project may be a useful metric for future works, although this is limited to measuring the activity of publishing researchers and may exclude many industry-based practitioners. Alternative metrics for analysis could also include the date of the first commit on GitHub to indicate project lifespan or a 'time-to-fix' metric where start and end dates of GitHub issues are used to assess maintainer activity. Another avenue for future work could compare the aforementioned libraries via empirical analysis across a range of common time series datasets for forecasting and anomaly detection. ML models can have long training times and are highly energy-intensive (Lacoste et al., 2019). Empirical experiments could therefore be expanded to evaluate the environmental impact of training ML models via proxy metrics such as CPU/GPU usage or computation runtimes.

ACKNOWLEDGEMENTS

The authors acknowledge the contribution of Morgain Siede for guidance on statistical analysis. Thanks are extended to MTU for their support via the Recurrent Funding Allocation Model (RFAM) program.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., and Corrado, G. S. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262.

Ali, M. (2020). PyCaret.

Alteryx (2022). Evalml.

Amarasinghe, K., Kenney, K., and Manic, M. (2018). Toward Explainable Deep Neural Network Based Anomaly Detection. In *11th Intl. Conf. on Human System Interaction*. IEEE.

Bahri, M., Salutari, F., Putina, A., and Sozio, M. (2022). AutoML: state of the art with a focus on anomaly detection, challenges, and research directions. *Intl. Journal of Data Science and Analytics*, 14.

Bergmann, D., Connor, D., and Thuemmel, A. (2018). An Evaluation Of Point And Density Forecasts For Selected Eu Farm Gate Milk Prices. *Intl. Journal of Food and Agricultural Economics*, 6.

Bergstra, J. and Bengio, Y. (2018). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13.

Borovykh, A., Bohte, S., and Oosterlee, C. W. (2018). Conditional Time Series Forecasting with Convolutional Neural Networks. *arXiv:1703.04691 [stat]*.

Catlin, C. (2022). AutoTS.

Chatterjee, S., Bopardikar, R., Guerard, M., Thakore, U., and Jiang, X. (2022). MOSPAT: AutoML based Model Selection and Parameter Tuning for Time Series Anomaly Detection. arXiv:2205.11755.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., and Xiao, T. (2015). Mxnet.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.

Choudhary, D., Kejariwal, A., and Orsini, F. (2017). On the Runtime-Efficacy Trade-off of Anomaly Detection Techniques for Real-Time Streaming Data.

Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., and Yang, S. (2017). AdaNet: Adaptive Structural Learning of Artificial Neural Networks. In *Proc. of the 34th Intl. Conf. on Machine Learning*.

De Romblay, A. (2022). MLBox.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. (2020). AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data.

Facebook (2022). Prophet.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2021). Auto-Sklearn 2.0.

Flynn, C. (2022). PyPI Stats.

Francois, C. (2022). Keras: the Python deep learning API.

Ghaderi, H. and Kabiri, P. (2012). Fourier transform and correlation-based feature selection for fault detection of automobile engines. In *The 16th CSI Intl. Symposium on Artificial Intelligence and Signal Processing*.

Github (2022). Github.

Goix, N. (2016). How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms? *arXiv:1607.01152*.

Grimes, D., Ifrim, G., O'Sullivan, B., and Simonis, H. (2014). Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. *Sustainable Computing: Informatics and Systems*, 4.

Gürtler, M. and Paulsen, T. (2018). The effect of wind and solar power forecasts on day-ahead and intraday electricity prices in Germany. *Energy Economics*, 75.

H2O (2022). H2O.ai.

He, X., Zhao, K., and Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212.

Hesterman, J. Y., Caucci, L., Kupinski, M. A., Barrett, H. H., and Furenlid, L. R. (2010). Maximum-Likelihood Estimation With a Contracting-Grid Search Algorithm. *IEEE Trans. on Nuclear Science*.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, 9.

Hundman, K., Constantinou, V., Laporte, C., Colwell, I., and Soderstrom, T. (2018). Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. *Proc. of the 24th ACM SIGKDD Intl. Conf. on Knowledge Discovery & Data Mining*.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33.

Javeri, I. Y., Toutiaee, M., Arpinar, I. B., Miller, J. A., and Miller, T. W. (2021). Improving Neural Networks for Time-Series Forecasting using Data Augmentation and AutoML. In *2021 IEEE Seventh Intl. Conf. on Big Data Computing Service and Applications*.

Jin, H., Song, Q., and Hu, X. (2019). Auto-Keras: An Efficient Neural Architecture Search System. In *25th Conf. on Knowledge Discovery & Data Mining*. Association for Computing Machinery.

Kavanagh, K., Barrett, M., and Conlon, M. (2017). Short-term electricity load forecasting for the integrated single electricity market (I-SEM). In *2017 52nd Intl. Universities Power Engineering Conf.*

Kaytez, F., Taplamacioglu, M. C., Cam, E., and Hardalac, F. (2015). Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines. *Intl. Journal of Electrical Power & Energy Systems*, 67.

Kim, Y., Wang, P., Zhu, Y., and Mihaylova, L. (2018). A Capsule Network for Traffic Speed Prediction in Complex Road Networks. *arXiv:1807.10603*.

Komer, B., Bergstra, J., and Eliasmith, C. (2019). Hyperopt-Sklearn. In *Automated Machine Learning: Methods, Systems, Challenges*.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the Carbon Emissions of Machine Learning.

Lago, J., De Ridder, F., and De Schutter, B. (2018). Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms. *Applied Energy*, 221.

Li, Y., Chen, Z., Zha, D., Zhou, K., Jin, H., Chen, H., and Hu, X. (2020a). AutoOD: Automated Outlier Detection via Curiosity-guided Search and Self-imitation Learning.

Li, Y., Zha, D., Venugopal, P., Zou, N., and Hu, X. (2020b). PyODDS: An End-to-end Outlier Detection System with Automated Machine Learning. In *Companion Proc. of the Web Conf.*

LinkedIn (2022). Luminol.

Lu, Y. and Xu, L. D. (2019). Internet of Things (IoT) Cybersecurity Research: A Review of Current Research Topics. *IEEE Internet of Things Journal*, 6.

Lynch, C., Kehoe, J., Bain, R., Zhang, F., Flynn, J., O'Leary, C., and Smith, G. (2019). Application of a SVM-based model for day-ahead electricity price prediction for the single electricity market in Ireland. In *39th Intl. Symposium on Forecasting*.

Lynch, C., O'Leary, C., Smith, G., Bain, R., Kehoe, J., Vakaloudis, A., and Linger, R. (2020). A review of open-source machine learning algorithms for twitter text sentiment analysis and image classification. In *Intl. Joint Conf. on Neural Networks*.

Lynch, C., O'Leary, C., Sundareshan, P. G. K., and Akin, Y. (2021). Experimental Analysis of GBM to Expand the Time Horizon of Irish Electricity Price Forecasts. *Energies*, 14.

Meta (2022). Kats.

Molino, P., Dudin, Y., and Miryala, S. S. (2019). Ludwig.

Nguyen, Q. T., Fouchereau, R., Frénod, E., Gerard, C., and Sincholle, V. (2020). Comparison of forecast models of production of dairy cows combining animal and diet parameters. *Computers and Electronics in Agriculture*, 170.

Nikitin, N. O., Vychuzhanin, P., Sarafanov, M., Polonskaia, I. S., Revin, I., Barabanova, I. V., and Maximov, G. (2022). Automated evolutionary approach for the design of composite machine learning pipelines. *Future Generation Computer Systems*, 127.

NVIDIA (2017). CUDA Zone.

O'Leary, C. (2020). Capsule Networks for Electricity Price Forecasting. Master's thesis, MTU.

O'Leary, C. and Lynch, C. (2022). An Evaluation of Machine Learning Approaches for Milk Volume Prediction in Ireland. In *Irish Signals and Systems Conf.*

O'Leary, C., Lynch, C., Bain, R., Smith, G., and Grimes, D. (2021). A Comparison of Deep Learning vs Traditional Machine Learning for Electricity Price Forecasting. In *2021 4th Intl. Conf. on Information and Computer Technologies*. IEEE.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proc. of the Genetic and Evolutionary Computation Conf.*

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., and Lin, Z. (2017). Automatic differentiation in PyTorch.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., and Blondel, M. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*.

Plonska, A. and Płoński, P. (2021). MLJAR.

PyPI (2022). The Python Package Index.

Python (2022). Python.org.

Qiu, J., Wu, Q., Ding, G., Xu, Y., and Feng, S. (2016). A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2019). How Does Batch Normalization Help Optimization? *arXiv:1805.11604*.

Seabold, S. and Perktold, J. (2010). Statsmodels: Econometric and Statistical Modeling with Python. *Proc. of the 9th Python in Science Conf.*

Singh, N. and Mohanty, S. R. (2015). A Review of Price Forecasting Problem and Techniques in Deregulated Electricity Markets. *Journal of Power and Energy Engineering*, 03.

Smith, T. (2017). ARIMA estimators for Python.

SonarQube (2022). Code Quality and Code Security.

StackOverflow (2021). Stack Overflow Developer Survey.

Thessen, A. (2016). Adoption of Machine Learning Techniques in Ecology and Earth Science. *One Ecosystem*.

Tinkoff.AI (2022). Etna.

Ugurlu, U., Oksuz, I., and Tas, O. (2018). Electricity Price Forecasting Using Recurrent Neural Networks. *Energies*, 11.

Vakhrushev, A., Ryzhkov, A., Savchenko, M., Simakov, D., Damdinov, R., and Tuzhilin, A. (2022). LightAutoML: AutoML Solution for a Large Financial Services Ecosystem.

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

Wang, L., Zhang, Z., and Chen, J. (2017). Short-Term Electricity Price Forecasting With Stacked Denoising Autoencoders. *IEEE Trans. on Power Systems*, 32.

Warzynski, A., Falas, L., and Schauer, P. (2021). Excess-Mass and Mass-Volume anomaly detection algorithms applicability in unsupervised intrusion detection systems. In *IEEE 30th Intl. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

Wingkvist, A., Ericsson, M., Lincke, R., and Löwe, W. (2010). A Metrics-Based Approach to Technical Documentation Quality. In *2010 7th Conf. on the Quality of Information and Communications Technology*.

Wu, J., Zeng, W., and Yan, F. (2018). Hierarchical Temporal Memory method for time-series-based anomaly detection. *Neurocomputing*, 273.

Xu, D., Wang, Y., Meng, Y., and Zhang, Z. (2017). An Improved Data Anomaly Detection Method Based on Isolation Forest. In *2017 10th Intl. Symposium on Computational Intelligence and Design*, volume 2.

Yu, L., Liu, L., Pu, C., Gursoy, M. E., and Truex, S. (2019). Differentially Private Model Publishing for Deep Learning. *Symposium on Security and Privacy*.

Zha, D., Lai, K.-H., Wan, M., and Hu, X. (2020). Meta-AAD: Active Anomaly Detection with Deep Reinforcement Learning.

Zhang, F., Shine, P., Upton, J., Shaloo, L., and Murphy, M. D. (2018). A Review of Milk Production Forecasting Models: Past & Future Methods. In *Dairy farming operations management, animal welfare and milk production*.

Zhang, F., Upton, J., Shalloo, L., Shine, P., and Murphy, M. D. (2020). Effect of introducing weather parameters on the accuracy of milk production forecast models. *Information Processing in Agriculture*, 7.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., and Xie, C. (2019). Robust log-based anomaly detection on unstable log data. In *Joint Meeting on European Software Engineering Conf. and Symposium on the Foundations of Software Engineering*. ACM.

Zhao, Y., Nasrullah, Z., and Li, Z. (2022). PyOD: A Python Toolbox for Scalable Outlier Detection.

Zhao, Y., Rossi, R. A., and Akoglu, L. (2021). Automating Outlier Detection via Meta-Learning.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 43.