eng

MDPI

Article

Fundamental Components and Principles of Supervised Machine Learning Workflows with Numerical and Categorical Data

Styliani I. Kampezidou [1,*], Archana Tikayat Ray [2], Anirudh Prabhakara Bhat [3], Olivia J. Pinon Fischer [1] and Dimitri N. Mavris [1]

[1] Aerospace Systems Design Laboratory, School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA; olivia.pinon@asdl.gatech.edu (O.J.P.F.); dimitri.mavris@aerospace.gatech.edu (D.N.M.)
[2] AI Fusion Technologies, Toronto, ON M5V 3Z5, Canada; archanatikayatray@gmail.com
[3] Amazon, Toronto, ON M5H 4A9, Canada; anirbhat@amazon.com
* Correspondence: skampezidou@gatech.edu

Abstract: This paper offers a comprehensive examination of the process involved in developing and automating supervised end-to-end machine learning workflows for forecasting and classification purposes. It offers a complete overview of the components (i.e., feature engineering and model selection), principles (i.e., bias–variance decomposition, model complexity, overfitting, model sensitivity to feature assumptions and scaling, and output interpretability), models (i.e., neural networks and regression models), methods (i.e., cross-validation and data augmentation), metrics (i.e., Mean Squared Error and F1-score) and tools that rule most supervised learning applications with numerical and categorical data, as well as their integration, automation, and deployment. The end goal and contribution of this paper is the education and guidance of the non-AI expert academic community regarding complete and rigorous machine learning workflows and data science practices, from problem scoping to design and state-of-the-art automation tools, including basic principles and reasoning in the choice of methods. The paper delves into the critical stages of supervised machine learning workflow development, many of which are often omitted by researchers, and covers foundational concepts essential for understanding and optimizing a functional machine learning workflow, thereby offering a holistic view of task-specific application development for applied researchers who are non-AI experts. This paper may be of significant value to academic researchers developing and prototyping machine learning workflows for their own research or as customer-tailored solutions for government and industry partners.

Keywords: machine learning workflow; supervised learning; numerical data; categorical data; data engineering; extraction, loading, transformation; feature engineering; automated feature extraction; machine learning engineering; training, validation, evaluation; test-driven development; automated machine learning; model deployment

Citation: Kampezidou, S.I.; Tikayat Ray, A.; Bhat, A.P.; Pinon Fischer, O.J.; Mavris, D.N. Fundamental Components and Principles of Supervised Machine Learning Workflows with Numerical and Categorical Data. *Eng* **2024**, *5*, 384–416. https://doi.org/10.3390/eng5010021

Academic Editor: Gian Carlo Cardarilli

Received: 11 December 2023
Revised: 25 February 2024
Accepted: 27 February 2024
Published: 29 February 2024

1. Introduction

Several machine learning applications are implemented as a workflow, which starts with data collection and ends with a model evaluation and simulations or software development. Examples of fields that introduce custom machine learning workflow solutions include, but are not limited to, malware detection and classification [1], software development with adversarial attack classification [2], task fault prediction in workflows developed with cloud services [3], pipeline optimization [4], the classification of forest stand species via remote sensing [5], the detection of mechanical discontinuities in materials and the prediction of martensitic transformation peak temperature of alloys [6,7], the optimization of metabolic pathways and ranking of miRNAs retarding insulin gene transcription in human islets [8,9], large-scale crop yield forecasting [10], classification and forecasting in chemical

engineering [11], predictive modeling in medicine [12], protein engineering/biophysical screening in pharmaceutical sciences [13], the forecasting of oil uptake in batter for food science [14], vegetation height classification, forecasting fractured coal seams, climate-related forecasting in environmental sciences [15–17], and energy systems controlled by occupancy detection or energy demand forecasting [18–22], as well as environmental impact estimation from commercial aviation and aerospace requirements engineering [23–27]. Specifically in Artificial Intelligence (AI), machine learning frameworks have been proposed for a variety of data-driven expert systems, such as recommendation systems [28], decision support systems [29], fault diagnosis [30,31], and crowdsourcing [32], as well as generic data science [33] and big data [34] applications. Machine learning workflows or workflow sub-modules are often automated, i.e., automated classification [35], automated machine learning (AutoML) in healthcare [36], aviation [37,38], biology [39], and agriculture [40,41].

Many researchers, including non-AI experts, develop machine learning workflows for a specific application as the previous references clearly suggest. Therefore, they could benefit from guidance on how to develop rigorous and functional end-to-end machine learning workflows, for their sustainable research or for a government or industry partner, without missing important components or making decisions without considering fundamental AI principles, and also from understanding the state-of-the-art automation tools that can optimize their pipeline design and parameterization. Related work, such as [42–44], addressed critical components of a machine learning workflow, but without a focus on supervised cases with numerical and categorical data, while they were not very thorough on the interconnection of the sub-modules, and lacked the explanation of the basic principles required for the development of the workflow. Moreover, the references of [42] are mostly websites and blogs, as opposed to scientific papers. On the other hand, relative books in pipeline development focus more on coding using a specific language and often lack important concept explanations, such as the bias–variance decomposition [45–47], or may focus only on a specific part of the workflow, such as feature engineering [48], lacking to provide a holistic overview of the entire workflow, the importance of every sub-module and their meaningful, other than random, integration. Other review papers focus entirely on the automation of a very specific machine learning area, such as time series forecasting automation [49], and lack the explanation of basic AI principles and workflow sub-module operation, interconnection and development for the education and guidance of a non-AI researcher, seeking to understand and complete their manual workflow development before automating. Last, there are some machine learning workflow automation papers [41,50] that focus on a very specific application, i.e., biology or computer networks, that are, however, relatively incomplete and missing important AI principles, as well as references of methods outside of what has been published for that specific application under consideration, making them appropriate to guide researchers only on the specific application and not on the broader spectrum of supervised learning with numerical and categorical data.

The contributions of this paper are threefold. First, this work introduces a methodology for the development and thorough understanding of end-to-end supervised machine learning workflows with numerical and categorical data. Second, this methodology is accompanied by all the workflow sub-modules, methods, principles, models, and algorithms, as well as their integration into the workflow, to guide and inform the non-AI researcher on developing rigorous pipelines. Third, it provides state-of-the-art tools for automated machine learning, including but not limited to automated feature engineering, architecture search, hyperparameter search, etc. A systematic and in-depth explanation of machine learning workflow sub-modules and interconnection allows researchers and non-AI professionals to fully comprehend, improve, and excel in machine learning workflow development and automation.

The rest of this paper is organized as follows. Section 2 provides an overview of the end-to-end machine learning workflow architecture. Section 3 presents the data-engineering sub-module, and Section 4 presents the machine learning sub-module, which

includes models and algorithms, as well as training, validation, and evaluation methods. Section 5 analyzes the model deployment step, Section 6 presents the state-of-the-art automation methods in machine learning workflows and related coding practices, and Section 8 provides conclusions.

2. End-to-End Architecture of Machine Learning Workflows

The workflow of every machine learning (ML) project broadly consists of four stages: scoping of the problem, data engineering, modeling, and deployment of the model. In order to deploy and maintain machine learning models in production reliably and efficiently, a set of practices is necessary to automate the machine learning workflow while meeting business and regulatory requirements, known as Machine Learning Operations (MLOps) [51]. MLOps is considered to be the cross-section of Development Operations (DevOps), data engineering, and Machine Learning Engineering [51]. DevOps is another set of practices that combine software development (Dev) and IT operations (Ops) in order to develop useful software and maintain all data resources in data centers. The core principle of DevOps is automation, and therefore Continuous Integration Continuous Delivery or Deployment (CI/CD) is a very important component so that any software updates from developers are integrated and delivered to the latest software version [52]. CI/CD requires continuous development, continuous testing (passing a series of predefined tests; see Section 5 for more details), continuous integration, continuous deployment, and continuous monitoring. Expertise in all three areas shown in Figure 1 is required for a machine learning workflow to transition into a product.

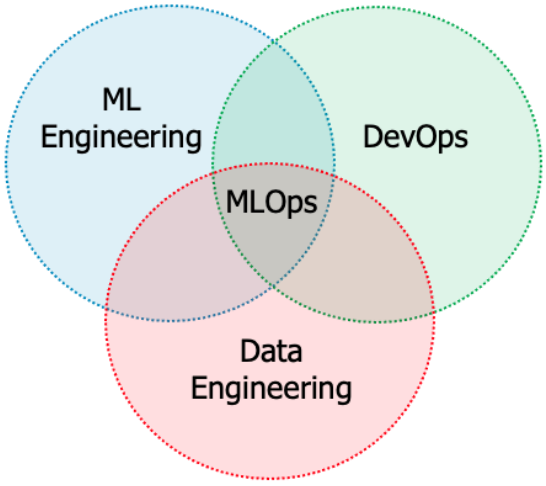

Figure 1. MLOps: the cross-section of Machine Learning Engineering, DevOps, and data engineering.

A machine learning workflow begins with scoping the problem to be solved, examining the available resources, and deciding upon the feasibility of the project. Then, the objectives and requirements are set, and the project is initiated. As Figure 2 shows, data engineering, machine learning model engineering, and model development (production code) are the subsequent stages that will be continuously updated in an automated way every time new data arrive. This workflow will be incorporated into production code and be subject to testing (Section 5) before every new version deployment as well as continuous monitoring and maintenance throughout the entire lifecycle.

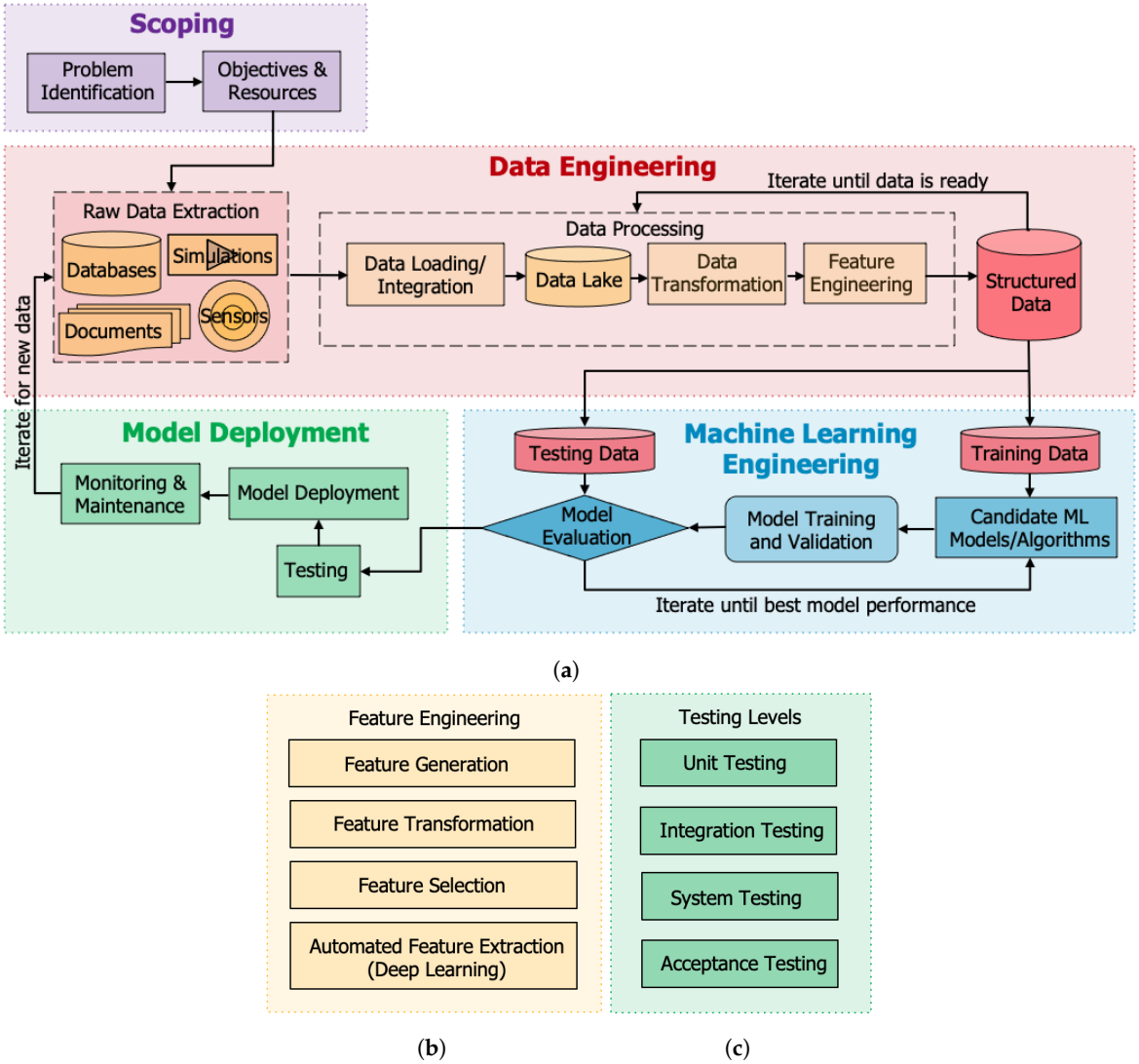

Figure 2. (**a**) The machine learning workflow, (**b**) components of feature engineering, and (**c**) the testing levels of Test-Driven Development (TDD).

3. Data Engineering

This section provides a detailed explanation of the various data-engineering stages.

3.1. Data Pipeline: Extraction, Loading and Transformation

Data can be extracted and fed into a machine learning workflow from several sources, including databases (DBs), documents, sensors, and simulations as shown in Figure 2a. Loading and integrating these data can be challenging, and it is an ongoing process, even after the machine learning model has been deployed and an initial master database has been constructed (continuous integration). All these data are in raw format and require transformation (cleaning), i.e., treating missing values (i.e., imputation [53,54], complete entry removal, matrix completion [55–59] depending on the randomness in the missing data pattern or by producing the missing data with simulations [60]), correcting for data formats (mathematical representation of numbers, time stamps, inconsistent text format, size and resolution of images, etc.), de-duplicating [61] (with or without a unique key per real-world entry) or removing redundant data, dealing with contradicting data, removing outliers (point outliers, collective or contextual outliers), etc. Current industry practices for data engineering include ELT tools (Extract, Load, Transform) [62], which offer data extraction, data integration, and data transformation functions via some Application Programming

Interface (API). ELT is currently preferred as opposed to the previously followed approach ETL (Extract, Transform, Load) [63] because the data are integrated and saved before any transformations occur (data lake implementation). This way, the same data can be accessed by several analysts, scientists, and engineers before any transformations take place, and the scalability of transformation scripts, as the data set increases, is no longer an issue. These transformations are version-controlled, and therefore historical transformations can be re-created by rolling back commits in ELT, while data and models can be easily tested. ELT has been growing quickly due to the following enablers: the development of modern data lakes, access to products that can load code and store data in data lakes, and the necessity to open the ELT process to anyone who has Structured Query Language (SQL) knowledge [64].

3.2. Feature Engineering

Feature engineering consists of feature generation, feature transformation, feature selection, and automated feature extraction for Deep Learning as shown in Figure 2b.

3.2.1. Feature Generation

Feature generation refers to the process of creating new features for the purpose of improving the success metric, which validates the machine learning model (Figure 2a), and therefore, it is a very important step in the machine learning workflow. To extract new features, data visualization and domain expertise [18,65] are necessary, but when unavailable or too complicated, Deep Learning neural network architectures can automatically extract features with additional layers which are trained to simulate the appropriate feature transformations that optimize training and generalization errors [66,67]. However, for a researcher with a deeper understanding of the underlying problem, feature engineering will result in producing features by generating, combining, or transforming existing features (feature transformation explained next) in the direction of machine learning model performance improvement, given an appropriate evaluation metric (Section 4.3). However, it is important to consider a priori a machine learning model in terms of feature learning capabilities. For example, linear regression models are able to learn sums and differences easily but a complex relationship between inputs and outputs would not be learned easily due to model linearity, and therefore, such a model could benefit from generated features that explicitly capture that complicated non-linear relationship [68], or by linearizing these relationships. On the other hand, it is wise to be careful regarding generated features such as ratios because they can be harder to learn by most machine learning models. Moreover, trees are more capable of capturing complex relationships between the inputs and outputs (numerical or categorical) but could still benefit from explicit features generated to capture those [69]. They can also benefit from group aggregate features, such as counts (features that count the total number of times a binary value is present), minimum and maximum feature values, and mean and standard deviation, while they are often used for feature selection, a by-product of the tree training [70]. However, trees can be sensitive to noisy inputs, so it is recommended to remove irrelevant input information, either by removing a feature or by removing a feature's component (i.e., higher frequency noise) [71].

3.2.2. Feature Transformation

Feature transformation includes feature scaling and the application of deterministic transformations, and its significance in the machine learning workflow is explained here. Feature scaling is an important part of the machine learning workflow since models that are smooth functions of the input, such as linear regression, Logistic Regression (LR), Neural Networks (NNs), and other matrix-based models can be affected by the feature scale (values), given that the parameter vector which is updated via the Stochastic Gradient Descent method will be biased towards specific features [72]. Stochastic Gradient Descent algorithms also converge faster [73] with scaled feature values and with smaller derivatives during back-propagation (a by-product of feature scaling). Feature scaling is also necessary

for Principal Component Analysis (PCA), used for dimensionality reduction, which may also be biased since the direction of maximum variance can be affected by large feature values. This issue has been bypassed by creating features with unit variance via scaling with the standard deviation [74]. On the other hand, distance-based algorithms, such as k-Nearest Neighbors (kNN), k-Means, and Support Vector Machines (SVMs) are mostly affected by the range of values because they use distances between data points to determine their similarity; therefore, imposing scaling methods that keep the range the same for all features, such as Min-Max, can provide improved results [75]. Moreover, tree-based supervised algorithms such as Classification and Regression Trees (CARTs), Random Forests (RFs), and Gradient Boosted Decision Trees [71], are not affected by either the scale or the range of the feature values [48] because most splitting criteria utilize one feature at a time, and monotonic feature transformations do not affect the order of the data points [71]. Hence, the same data point (threshold) will eventually be selected to split the specific node of the tree, whether the data points are scaled or not [48]. Last, graphical classifiers like Naive Bayes or Fisher Linear Discriminant Analysis (LDA) that rely on variable distributions also do not require feature scaling. The most common feature-scaling methods are summarized in Table 1, where $\mathbf{x} \in \mathbb{R}^n$ is some feature. For an example of the effect of different scaling methods on different supervised classification models and algorithms, the reader is referred to [75], which provides performance index values for a heart disease data set using different scaling methods on different machine learning models and algorithms. It is up to the researcher to evaluate the impact of different scaling methods on the workflow and decide the best way to move forward based on Table 1 recommendations, as well as by following a similar approach to the heart disease example in [75].

Table 1. Summary of the most common numerical feature-scaling methods.

Scaling Method	Scaled Feature	Scaling Effect	ML Algorithm/Model
Min-Max	$\mathbf{x}' = \frac{\mathbf{x}-\mathbf{x}^{\min}}{x^{\max}-x^{\min}}$	$\mathbf{x}' \in [0,1]$	k-Means, kNN, SVM
Standardization (z-score)	$\mathbf{x}' = \frac{\mathbf{x}-\bar{\mathbf{x}}}{\sigma}$	$\bar{\mathbf{x}}' = 0, \sigma = 1$	Linear/Logistic Regression, NN
l_2-Normalization	$\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}$	$\| \mathbf{x}' \|_2 = 1$	Vector Space Model

To better understand the effect of scaling methods, it is worth mentioning that the principle behind any scaling method in Table 1 that divides by a normalization constant is that the feature is scaled without changing the shape of the single-feature distribution (see Figures 2–18 in [48]). For example, if the data distribution is not Gaussian before standardization, then it will not become Gaussian after. It will, however, have a mean of 0 and a standard deviation of 1 as mentioned in [48]. The interested researcher, however, should be careful in choosing a scaling method in terms of sparsity in the original features. For example, although Min-Max scaling bounds the scaled feature in the $[0, 1]$ range, that is not the case with standardization and l_2-normalization. Min-Max and standardization both subtract a value from the original feature, which may or may not be zero. In the case where the original feature is sparse, doing so may result in a dense scaled feature if the subtracted value is not zero, and that may further burden a classifier model [48].

Feature transformation also includes the application of deterministic, usually invertible, transformations on numerical data. Following data visualization, transformations can be applied to improve feature interpretability (for example, removing high-frequency noise, introducing lag features, feature derivatives, etc. [49]) or conform with the assumptions of machine learning models, such as linear regression (linearity, residual independence, homoscedasticity, and residual normality). For example, in order for a researcher to use linear regression models, a linear relationship between the independent variable and the target variable is required, something not necessary if a neural network architecture was selected. If that relationship is visualized to be non-linear, then a transformation can be decided to

improve the linearity between the transformed independent variable and the target [76]. A common family of such transformations is the power transformations (power monotonic functions), with special cases of the log transformation, the square root transformation, and the multiplicative inverse transformation (reciprocal transformation), all for non-zero data. The power transformations are parameterized by a non-negative value λ, which can be found via statistical estimation methods. When feature values vary in both the positive and negative range, other transformations, such as the multiplicative inverse transformation, the Yeo–Johnson transformation [77], and the inverse hyperbolic sine transformation can be applied. However, for transformations that assume non-negative data, constant offsets can be applied first to the feature values to shift all values in the positive reals. Another reason behind feature transformation is the principle of variance stabilization, which removes the dependency of a population variance from the population mean (for examples, the reader is referred to Figures 2–11 in [48]). Common variance stabilizing transformations are the Fisher transformation [78] for the sample correlation coefficient, the square root transformation or Anscombe transform [79] for Poisson data (count data), the Box–Cox transformation [80] for regression analysis, and the arcsine square root transformation or angular transformation for proportions (binomial data) [81].

Feature transformation is also required for categorical features for the same range and value size reasons as in the numerical data case above. The most common transformation techniques for categorical features are summarized in Table 2, and their advantages and disadvantages are stated here. Researchers should be cautious in using Ordinal Encoding because of its disadvantage in generating ordered numerical features when no order is present in the original categorical data, which may affect the machine learning algorithm or model [48]. On the other hand, One-Hot Encoding does not have that effect; it clearly assigns one value to each category, leaving the 0 category free for missing data. However, it has a sum of 1 amongst the k new features, which implies a linear dependency between those transformed features. That can lead to issues with training linear models since different linear combinations of the dependent features can produce the same predicted target value [48]. By removing one of the degrees of freedom, One-Hot Encoding becomes Dummy Encoding, where the reference category (missing category) is represented by all zeros, which does not allow for easy missing data representation. Moreover, in terms of interpretation, if a linear regression model is used, in the case of One-Hot Encoding, the intercept represents the target variable's mean, while with Dummy Encoding, the intercept refers to the mean value of the target variable for the reference category [48]. Last, with Effect Coding, the resulting linear regression models are easier to interpret because the intercept represents the mean of the target variable. On the other hand, the dense -1 representation of the reference category can be expensive in terms of storage and computation. Since all the aforementioned encoding methods are not scalable for large data sets, scalability is only possible via feature compression. Feature compression is translated into either feature hashing (linear and kernel models) or bin counting (linear models and trees), where instead of the categorical feature, the conditional probability of the target variable under that value is used. The back-off method [48] or the count-min sketch [82] methods are available in the literature for rare categories in large data sets that were previously transformed with bin counting [48].

To summarize the feature transformation sub-module, the interested researcher is encouraged to also consider multivariate feature transformations, which transform several features at a time and may have superior performance in specific circumstances where, for example, the features are correlated [83] and high correlation can lead to feature redundancy. One such example is whitening transformations (linear) which lead to uncorrelated features with an identity covariance matrix [83]. The covariance matrix may first be expressed in a decomposed form via, for example, Cholesky decomposition [84], before the whitening transformation is applied. Other similar transformations include decorrelation, which removes the correlations but leaves variances intact [83]; standardization, which sets the variances to 1 but leaves the correlations intact; and coloring, which turns a random a

vector of white random variables (with identity covariance) into a random vector with a pre-specified covariance matrix.

Table 2. Summary of most common methods for encoding categorical features.

Encoding Method	Original Feature	Transformed Features	Result
Ordinal Encoding	string1, string2, ...	1, 2, ...	Nonordinal categorical data becomes ordinal
One-Hot Encoding	string1, string2, ...	001, 010, ...	k features for k categories, only one bit is "on"
Dummy Encoding	string1, string2, ...	001, 010, ..., (000)	$k-1$ features for k categories, reference category is 0
Effect Encoding	string1, string2, ...	001, 010, ..., (-1-1-1)	k features for k categories, reference category is -1

3.2.3. Feature Selection

Feature selection is another crucial sub-module of the machine learning workflow, often omitted for brevity or due to lack of expertise in this area. However, the lack of features can lead to machine learning model underfitting and redundancy to overfitting, and therefore proper feature selection is important [72], following the feature generation and transformation steps. Feature selection can remove redundant features that are irrelevant to the target variables (adding "noise") by either selecting a subset of the features or by creating a lower-dimensional representation (embedding) of the feature set, which adequately summarizes the necessary information contained in the original feature space [48] but unfortunately without the physical interpretation of the original features (dimensionality reduction is traded for interpretability). Moreover, feature selection serves the purpose of dimensionality reduction [85], which can remove the computational burden from the machine learning model by removing linearly dependent features. A very common method of dimensionality reduction is PCA, which is also a whitening transformation in the sense that the transformed features are no longer correlated [48]. In PCA, the dimensionality k of the embedding is subject to user choice, but the operational cost of PCA can be very high for over a thousand features since it involves Singular Value Decomposition (SVD) of the original feature matrix [48]. A very interesting application of PCA is anomaly detection in time series data [86]. Since PCA projects the features to a linear subspace via linear projection, it is not ideal when the data lie in a nonlinear manifold. It is also not ideal when feature interpretability is necessary. In contrast, k-Means can perform feature selection when data lie in a nonlinear manifold, by clustering features together and selecting one feature to survive from each cluster (representative), while the rest are being discarded [48]. Another approach to dimensionality reduction is autoencoders [87], which, however, suffer from a lack of feature interpretability as well. Common unsupervised and supervised feature selection methods are summarized in Figure 3.

Unsupervised feature selection methods can be categorized as follows (a taxonomy is provided in Figure 1 of [88]). Attention is required to determine whether the method can directly incorporate categorical features or not.

1. Filter methods [88–90]: In this category of methods, feature selection is a pre-processing step to the machine learning model training, and these methods are time-efficient:
   (a) Statistical/information-based: These methods maximize feature relevance by maximizing a dependence measure, such as variance, covariance, entropy [90], linear correlation [91], Laplacian score [92], and mutual information. Representative methods include Feature Selection with Feature Similarity (FSFS) [91] based on Maximal Information Compression Index (MICI) and Relevance Redundancy Feature Selection (RRFS) [93]. Fisher's criterion [94] is only used in supervised learning.
   (b) Spectral/sparsity learning: These methods perform spectral analysis or combine spectral analysis with spectral learning. They find a trade-off between Goodness-of-Fit and a feature similarity measure. Representative methods include Multi-Cluster Feature Selection (MCFS) [95], Unsupervised Discrimi-

native Feature Selection (UDFS) [96], and Non-negative Discriminative Feature Selection (NDFS) [89].

2. Wrapper methods [88–90]: In this category, feature selection is intertwined with the machine learning model training and hence evaluated by the model performance. These methods are more accurate than filter methods but less time-efficient:
   (a) Sequential methods: These methods perform clustering on each feature subset and evaluate the clustering results based on some criterion. They can be based on Expectation Maximization or the Trace Criterion [97], or on the min/max of intra/inter-cluster variance [98] and a decision can be made based on a score that provides feature ranking. Another alternative is the Simplified Silhouette Sequential Forward Selection (SS-SFS) proposed in [99].
   (b) Iterative methods: Ref. [100] performs clustering and feature selection simultaneously by evaluating feature weights called feature saliences. Other iterative methods include Local Learning-based Clustering with Feature Selection (LLC-fs) [101], Embedded Unsupervised Feature Selection (EUFS) [102], and Dependence Guided Unsupervised Feature Selection (DGUFS) [103].
3. Embedded methods: In this category, feature selection is part of the machine learning model training process [104].

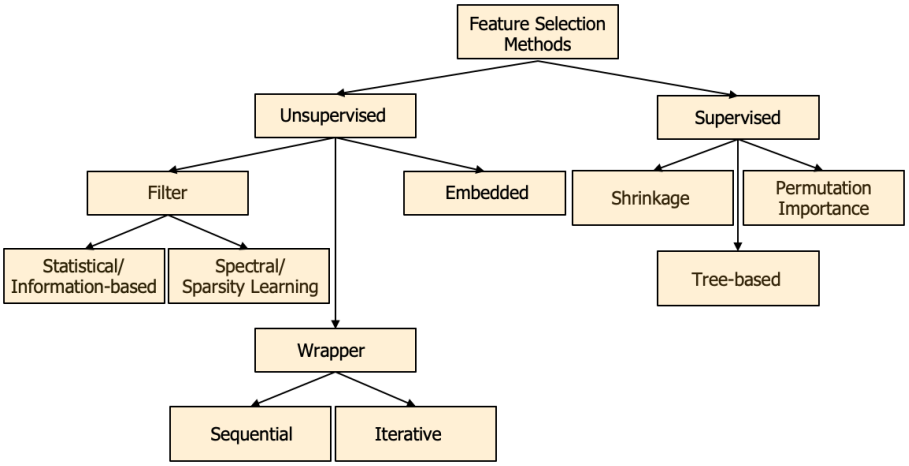

Figure 3. Taxonomy of feature selection methods.

Supervised feature selection methods include the deployment of a supervised machine learning model when target variables are available. After training a supervised machine learning model, a feature-ranking process is used to select the most important features that adequately describe the outputs (one or more target variables). Three main approaches in supervised feature selection are explained below. A more thorough overview of feature selection methods can be found in the book [104] and in chapter 19 of the book [105].

1. Shrinkage-based methods [106]: Single-output or multi-output regression models with L_1- or L_2-regularization can be trained via k-fold cross-validation (CV) to optimize a shrinkage parameter λ which trades-off model bias for variance. Penalization of the model weights with an l_1 norm is appropriate for feature selection because it can introduce feature sparsity (Lasso estimator [107]) when penalization with an l_2 norm (Ridge Regression [69]) does not force feature weights to zero. The combination of the two is called Elastic Net, which is useful when there is a group of features with high pairwise correlations [108]. Multi-output regression models perform better when outputs are correlated, i.e., when multi-task learning is desired, instead of independent task learning [109,110]. Multi-output models utilize an $l_{2,1}$ norm penalization term, which either includes or excludes a feature from the model for all outputs [111]. In the multi-output case, the average weight of a feature across all outputs is obtained, and then these average weights are normalized in the $[0, 1]$ range (relative importance)

with the min-max scaling method so that a rank of feature relative importance is derived [23].

2. Tree-based methods: CART can be trained in a supervised sense and provide feature ranking as a byproduct of the training process [71] in single-output or multi-output Decision Trees (DTs) [112]. DTs are over-sensitive to the training set, irrelevant information, and noise; therefore, prior unsupervised feature selection is strongly encouraged via one of the methods proposed above. Moreover, DTs are known to overfit, and hence, ensembles of DTs [112], such as Bagging (bootstrap aggregation) [113], Boosted Trees [114] and Rotation Forests, are constructed to cope with overfitting. The RF, a characteristic example of Bagging, can generate diverse trees by bootstrap sampling and/or randomly selecting a subset of the features during learning [115,116]. Although an RF is faster and easier to train than a boosted tree [117–119], it is less accurate and sacrifices the intrinsic interpretability (explanation of output value and feature ranking) present in DTs [69]. In particular, feature selection happens inherently in single-output and multi-output DTs as the tree is being constructed since the splitting criteria used at each node select the feature which performs the most successful separation of the remaining examples [71]. Therefore, in RFs, feature ranking is either impurity-based, such as the Mean Decrease in Impurity (MDI), or permutation-based, such as Permutation Importance [115]. MDI is also known as Mean Decrease Gini or Gini Importance.
3. Permutation Importance [115] is not only useful in RFs which have lost the inherent feature-ranking mechanism of the tree but in other supervised machine learning models as well. Permutation Importance is better than MDI because it is not computed on the training set but on the Out-of-Bag (OoB) sample and is, therefore, more useful to inform the feature importance for predictions [115]. Moreover, MDI significantly favors numerical features over categorical ones as well as high-cardinality categorical features (many categories) over low-cardinality ones [115], something that does not happen with Permutation Importance. The Permutation Importance of a feature is calculated as the difference between the original error and the average permuted error of this feature, over a number of specified repetitions [115]. The permuted error of each feature (the OoB error) occurs when that feature is permuted (shuffled). Permutation is a mechanism that breaks the relationship between that feature and the target variables, revealing the importance of a feature to the model training accuracy [120]. In trees and other supervised methods which use a feature-ranking approach to feature selection, the least-performing features in terms of relative importance can be excluded from the feature set.

3.2.4. Automated Feature Extraction

An important note on feature engineering is that as the number of features and samples increases, data engineering, feature engineering, and Machine Learning Engineering methods require scaling and modification to adjust to the new challenges. This concept was first introduced by Richard Bellman, as the Curse of Dimensionality (CoD) [121]. This principle is important to understand the need behind automated feature engineering.

CoD is associated with several challenges. First, the data representation in higher dimensions can be very sparse since the data tend to concentrate in lower-dimensional spaces, and therefore the sample size necessary for learning a statistical or machine learning model grows exponentially [122]. Second, the pairwise distances of the data points in higher-dimensional spaces increase, and they become more homogeneous; hence, observations may appear to be equally distanced from each other [123]. Two additional challenges related to CoD arise in the case of high-dimensional data clustering and anomaly detection. Those are the relevant attribute identification, i.e., the difficulty of describing the relevant quantitative locality of data instances in the high-dimensional space, and hubness, which refers to the tendency of high-dimensional data to contain data points that appear frequently in nearest neighbor clusters, known as hubs [124].

Table 3. Summary of most common automated feature engineering tools for high-dimensional data (FE = feature engineering).

Automated FE Tool	Operation	Tool Tested On	Developer	Paper
ExploreKit	Feature generation and ranking	DT, SVM, RF	UC Berkeley	[125]
One Button Machine	Feature discovery in relational DBs	RF, XGBOOST	IBM	[126]
AutoLearn	Feature generation and selection	kNN, LR, SVM, RF, Adaboost, NN, DT	IIIT	[127]
GeP Feature Construction	Feature generation from GeP on DTs	kNN, DT, Naive Bayes	Wellington University	[128]
Cognito	Feature generation and selection	N/A	IBM	[129]
RLFE	Feature generation and selection	RF	IBM	[130]
LFE	Feature transformation	LR, RF	IBM	[131]

The last component of feature engineering is automated feature extraction, which removes human intervention. This component is very useful in high-dimensional data that suffer from CoD, big data, and Deep Learning. Common examples of high-dimensional big data are time series, images (Chapter 8 in [48]), video data [132], and finance applications [133,134], while extracted features are saved and organized in feature stores (structured data in Figure 2a) for accessibility and re-usability. Core approaches in automated feature extraction include Multi-Relational Decision Tree Learning (MRDTL) [135], Deep Feature Synthesis (DFS) [136] and methods/models such as Deep Neural Networks (DNNs) [137], Adaptive Linear Approximation (ALA), PCA, extraction of Best Fourier Coefficients (BFCs) and Best Daubechies Wavelet Coefficients (BDWs), and statistical moments [138]. Once the features are extracted, then feature selection can be performed via an unsupervised or supervised method, appropriate for the problem dimensions. Recursive Feature Elimination Support Vector Machines (REFSVMs) and RELIEFF algorithms are proposed in [138] for feature selection, and they can take into account feature interaction (correlation) for up to 500 features. For more features, the Pearson correlation coefficient is used to remove highly correlated features before any of the two aforementioned algorithms is applied. Other methods for automated feature extraction are Genetic Programming (GeP) [139], Reconstruction Independent Component Analysis (RICA) [140], which generates sparse representations of whitened data, Sparse Filtering [141], which minimizes an objective function via a standard limited memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) quasi-Newton optimizer [142] and Wavelet–Scattering networks [143–146], which apply wavelet and scattering filters to produce low-variance features from time series and image data in Deep Learning applications. Features extracted by Wavelet–Scattering networks are insensitive to input translations on an invariance scale in the 1-D case [145], and insensitive to rotations in the 2-D case. Invariance to input transformations, such as measurement noise and image rotation and translation, is desired so that the performance of the machine learning model is unaffected. The interested reader is also encouraged to review additional automated feature engineering tools, such as ExploreKit [125], One Button Machine [126], AutoLearn [127], GeP Feature Construction [128], Cognito [129], RLFE [130], and LFE [131] (Table 3).

4. Machine Learning Engineering

4.1. Models and Algorithms for Supervised Learning with Numerical and Categorical Data

Since the area of machine learning models and algorithms is very broad, it would be impossible to cover in one paper, and therefore, this section focuses only on some basic supervised models and algorithms for the purposes of representing this workflow sub-module and its integration with the others. Supervised learning is possible when target variable values are present in regression (continuous range target values) or classification problems (pre-specified target values indicative of a class). Machine learning models are characterized as parametric if they have a specific function form with parameters, whose values can be determined by a data set [122]. The most commonly used machine learning

models and algorithms for supervised learning are summarized in Table 4. Note that DNN can be modeled as Gaussian Processes (GPs) [67] and hence are considered non-parametric as well. The choice of the problem model or algorithm is not trivial and highly depends on the scalability desired as well as the features collected (dimensionality, linearity, signal-to-noise ratio, probabilistic assumptions, correlation, etc.) as explained in the feature-engineering section. Often, researchers will experiment with multiple models/algorithms, but it is wise to narrow down the search for practical purposes. Table 4 summarizes basic criteria for guiding the choice of a model/algorithm, with each individual having additional assumptions that need to be taken into consideration.

Table 4. Summary of common supervised machine learning models and standard algorithm complexity. Symbols: n—samples; p—features; s—support vectors; k—neighbors or trees in a model; d—nodes in a tree; q—maximum number of bins; and γ—constant. The star ★ indicates that complexity varies with architecture (neurons, layers, connections, and activation function [147]) and algorithm (type, epochs). Typically, Gradient Descent has a running complexity of $\mathcal{O}(n^2 p)$.

ML Model/Algorithm	Parametric	Linear	Train, Test, Space Complexity	Paper
Ordinary Least Squares (OLS)	✓	✓	$\mathcal{O}(np^2 + p^3)$, $\mathcal{O}(p)$, $\mathcal{O}(p)$	[148]
Kernel Ridge Regression	✓	✓	$\mathcal{O}(n^3)$, -, $\mathcal{O}(n^2)$	[149]
Lasso Regression (LARS)	✓	✓	$\mathcal{O}(np^2 + p^3)$, -, -	[150]
Elastic Net	✓	✓	$\mathcal{O}(np^2 + p^3)$, -, -	[108]
Logistic Regression	✓	✓	$\mathcal{O}(np)$, $\mathcal{O}(p)$, $\mathcal{O}(p)$	[151]
GP Regression	✗	✗	$\mathcal{O}(n^3)$, -, $\mathcal{O}(n^2)$	[152]
Multi-Layer Perceptron	✓	✗	★	[153,154]
RNN/LSTM	✓	✗	★	-
CNN	✓	✗	★	-
Transformers	✓	✗	★	-
Radial Basis Function NN	✗	✗	★	-
DNN	✓	✗	★	-
Naive Bayes Classifier	✓	✓	$\mathcal{O}(np)$, $\mathcal{O}(p)$, $\mathcal{O}(nq)$	[155]
Bayesian Network	✗	✗	★	[156]
Bayesian Belief Network	✓	✗	★	-
SVM	✗	✓	$\mathcal{O}(n^2)$, $\mathcal{O}(sp)$, $\mathcal{O}(np)$	[157]
PCA	✗	✓	$\mathcal{O}(np\min(n,p) + p^3)$, -, $\mathcal{O}(np)$	[158]
kNN	✗	✗	$\mathcal{O}(knp)$, $\mathcal{O}(np)$, $\mathcal{O}(np)$	[159,160]
CART	✗	✗	$\mathcal{O}(n \cdot \log n \cdot p)$, $\mathcal{O}(p)$, $\mathcal{O}(\text{treedepth})$	[161]
RF	✗	✗	$\mathcal{O}(n \cdot \log n \cdot pk)$, $\mathcal{O}(pk)$, $\mathcal{O}(\text{treedepth} \cdot k)$	[162]
Gradient Boost Decision Tree	✗	✗	$\mathcal{O}(n \cdot \log n \cdot pk)$, $\mathcal{O}(dk + \gamma)$, $\mathcal{O}(\text{treedepth} \cdot k)$	[163]
LDA	✓	✓	$\mathcal{O}(npt + t^3)$, -, $\mathcal{O}(np + nt + pt)$, $t = \min(n,p)$	[164]

4.2. Model Training and Validation

It is important to ensure that the machine learning model is not overfitting during the training process so that better model generalization is achieved in the testing phase and future use of this model. Common approaches to prevent model overfitting include validation, CV [165], data augmentation, removal of features, early stopping of training before optimality, regularization, and ensembling (Bootstrap [166], Bootstrap Aggregating, also known as Bagging [113,167] and Boosting [168]). Although CV and Bootstrap include efficient sample re-use [69], other validation methods can be analytical, such as the Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), Minimum Description Length (MDL), and Structural Risk Minimization (SRM) [69]. Different validation methods provide a different test error estimate, which can be a more or less unbiased estimate of the true underlying test error as mentioned by the authors in [69].

To explain the model validation principle and the model complexity choice better to applied researchers developing pipelines in their field, train and test error estimates are summarized in Figure 4a. The light blue and red lines correspond to the train and test error estimates for different data sets of equal size. The dark blue and red lines are the

averages of the train and test error estimates of the light red and blue lines. The left half of Figure 4a (left of the red line minimum point) corresponds to higher error bias and lower error variance and is a sign of model underfitting due to the limited model's capacity. The right half of the same figure refers to higher model complexity, with higher error variance and lower error bias. That is the region of overfitting and happens when random noise in the data is modeled. The middle part which corresponds to the minimum of the test error (red line), is the ideal model complexity so that the best model generalization is achieved. The test error estimate is further decomposed into its components in Figure 4b. In supervised learning, the test error estimate is calculated by comparing the model's predictive capability with the ground truth. In unsupervised learning, model selection is often conducted by selecting the point of maximum absolute second derivative on the train error curve, also known as the elbow method [169], commonly used in k-Means. For other validation and evaluation metrics, including CV in unsupervised learning, the interested researcher is referred to the following papers [170–172].

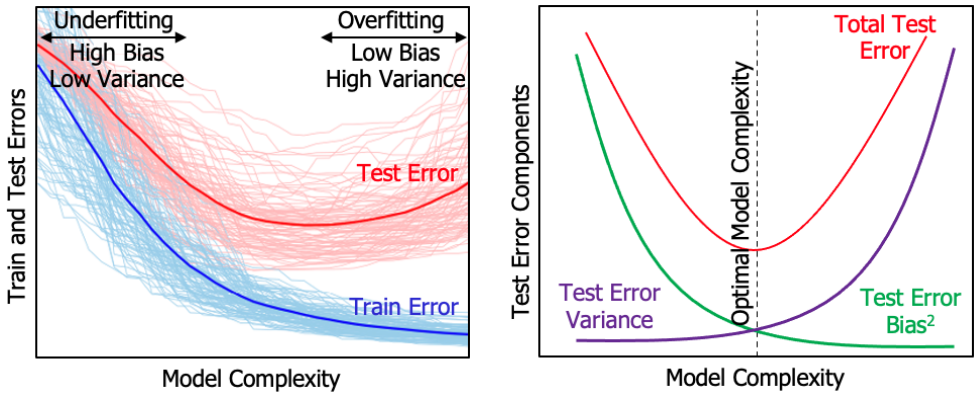

Figure 4. Train error and test error estimates: (**a**) behavior of train and test error estimates as the model complexity varies, figure imported and modified from [69], (**b**) decomposition of test error estimate in its components, according to Equation (1).

All the errors in Figure 4a are estimates based on some data set. Since the bias–variance decomposition is the most fundamental principle in machine learning, this section will provide the proof behind the results shown in Figure 4a,b, and help the interested researcher understand the role this concept plays in every decision made across the pipeline, from choosing a metric to using CV, regularization, feature selection, etc. The test error is decomposed in Figure 4b, according to Equation (1). Assume for simplicity that $y = f(x) + \epsilon$, where $y \in \mathbb{R}$ is the target variable, $f(x) \in \mathbb{R}$ is the machine learning model, $x \in \mathbb{R}$ is the model input, $\epsilon \in \mathbb{R}$ is noise with mean $\mathbb{E}[\epsilon] = 0$ and variance σ_ϵ^2, and $\hat{f}(x) \in \mathbb{R}$ is the fitted model to a specific training set. Then, the expected test error is decomposed as follows, with the help of the identity $(a+b+c)^2 = a^2 + b^2 + c^2 + 2ab + 2bc + 2ca$:

$$
\begin{aligned}
&= \mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[\left(f(x) + \epsilon - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[\left(f(x) + \epsilon - \hat{f}(x) + \mathbb{E}\left[\hat{f}(x)\right] - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right] \\
&= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)^2\right] + 2\mathbb{E}\left[\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\epsilon\right] + 2\mathbb{E}\left[\epsilon\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)\right] \\
&\quad + 2\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\right] \\
&= \mathbb{E}\left[(\epsilon - 0)^2\right] + \left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2 + \mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[(\epsilon - \mathbb{E}[\epsilon])^2\right] + \left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2 + \mathrm{Var}\left[\hat{f}(x)\right] \\
&= \sigma_\epsilon^2 + \mathrm{Bias}^2\left[\hat{f}(x)\right] + \mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{irreducibleerror} + \mathrm{Bias}^2\left[\hat{f}(x)\right] + \mathrm{Var}\left[\hat{f}(x)\right],
\end{aligned}
\tag{1}
$$

where the cross-terms are zero due to $f(x)$ being deterministic, i.e., $f(x) = \mathbb{E}[f(x)]$, the independence of ϵ and $\hat{f}(x)$ and the assumption that $\mathbb{E}[\epsilon] = 0$,

$$
\begin{aligned}
&= 2\mathbb{E}\left[\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\epsilon\right] + 2\mathbb{E}\left[\epsilon\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)\right] + 2\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\right] \\
&= 2\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\mathbb{E}[\epsilon] + 2\mathbb{E}[\epsilon]\mathbb{E}\left[\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right] + 2\mathbb{E}\left[\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right]\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right) \\
&= 2\left(\mathbb{E}\left[\hat{f}(x)\right] - \mathbb{E}\left[\hat{f}(x)\right]\right)\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right) \\
&= 0.
\end{aligned}
$$

The first term in (1), corresponds to the variance of the target variable around its true mean since $\mathbb{E}\left[\epsilon^2\right] = \mathbb{E}\left[(\epsilon - 0)^2\right] = \mathbb{E}\left[(\epsilon - \mathbb{E}[\epsilon])^2\right] = \mathbb{E}\left[(y - f(x) - (\mathbb{E}[y] - \mathbb{E}[f(x)]))^2\right] = \mathbb{E}\left[(y - f(x) - \mathbb{E}[y] + \mathbb{E}[f(x)])^2\right] = \mathbb{E}\left[(y - f(x) - \mathbb{E}[y] + f(x))^2\right] = \mathbb{E}[y - \mathbb{E}[y]] = \mathrm{Var}(y)$ and cannot be avoided. The second term, the squared bias, shows how the average of the estimate differs from the true mean, i.e., $\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2 = \left(\mathbb{E}[f(x)] - \mathbb{E}\left[\hat{f}(x)\right]\right)^2$. The third term describes the variance of $\hat{f}(x)$, i.e., the expected squared deviation of $\hat{f}(x)$ from its mean [69].

Having observed the behavior of the expected test error on testing data in Figure 4b, it is suggested to perform model selection on the best bias–variance trade-off point. Another set, the validation set, that was not used for training, is used for model selection, and the model performance is then evaluated on the testing set as explained in Section 4.3. Common ways of splitting the data set are 70% or 50% for the train set, 20% or 25% for the validation set, and 10% or 25% for the test set [69]. When splitting includes random sampling, then the model may generalize better. However, it is advised to avoid random sampling (shuffling) in time series data because of the signal autocorrelation. Moreover, it is suggested to use stratified training data in the case of classification to ensure that all classes are equally represented in the training set so that the classifier is not biased towards a specific class. Although in the case of big data or Deep Learning, the previous splitting approach would be sufficient (Chapter 7 in [69]), in small data sets with a higher danger of overfitting, it would not. Hence, different forms of CV are suggested to prevent bias and overfitting by re-using the data set multiple times. CV is avoided in Deep Learning because training can become computationally intractable, and therefore overfitting is managed in different ways. In big data sets, following training, the model performance is evaluated on the validation set. The validation set is used to tune model hyperparameters and prevent overfitting by early stopping, regularization, etc. The model performance is tested on the testing set that was not used for training or validation.

In cases where the data set is small, some form of CV is utilized as a way to avoid overfitting to that particular data set (Table 5). The first category of CV is Exhaustive CV. The data set of size n is split into a training set of size $n - p$ and a validation set of size p, in all possible ways. Every time, a model is fit and a validation error occurs. The average validation error across all validation sets is estimated for each hyperparameter value so that the optimal value can be selected at the minimum average error. Other variations of Exhaustive CV include the $p = 1$ case, which is less computationally heavy than the p case, or $p = 2$, which is an almost unbiased method for estimating the area under the Receiver Operating Characteristic (ROC) curve for binary classifiers [173].

The second category of CV includes the Non-Exhaustive methods, an approximation of the Exhaustive Leave-p-out CV, which is computationally tractable. The most popular method, k-fold CV, requires splitting the data set randomly in k sets, and keeping a different set for validation every time as shown in Figure 5. Again, the average error is calculated and utilized for hyperparameter estimation, by constructing a plot of average kfold CV error vs hyperparameter values. Popular choices for k are 5, 10, and 20. It is worth mentioning that the Leave-1-out CV has low bias and high variance, whereas the kfold CV with $k = 5$ or

$k = 10$, is a better compromise [174,175]. The next type of Non-Exhaustive CV method is the Holdout method, which randomly assigns p points in the training set and $n - p$ points in the testing set. It involves a single iteration and may be considered the same as the simple validation approach [175,176]. The last Non-Exhaustive CV method is the Repeated Random Sub-Sampling Validation, also known as Monte Carlo CV, which is similar to the k-fold CV process, only with random validation sets, which means that some observations may never be part of the validation set [105].

Table 5. Summary of CV methods.

CV Category	Specific CV Method	Result
Exhaustive CV	Leave-p-out CV	$C_p^n = \frac{n!}{(n-p)!p!}$ models trained
Non-Exhaustive CV	Leave-one-out CV	$C_1^n = n$ models trained
	k-fold CV	k models trained
	Holdout	1 model trained
	Repeated Random Sub-Sampling Validation (also known as Monte Carlo CV)	k models trained
Nested CV	k*l-fold CV	$k \cdot l$ models trained
	k-fold CV with validation and test set	k models trained with test set

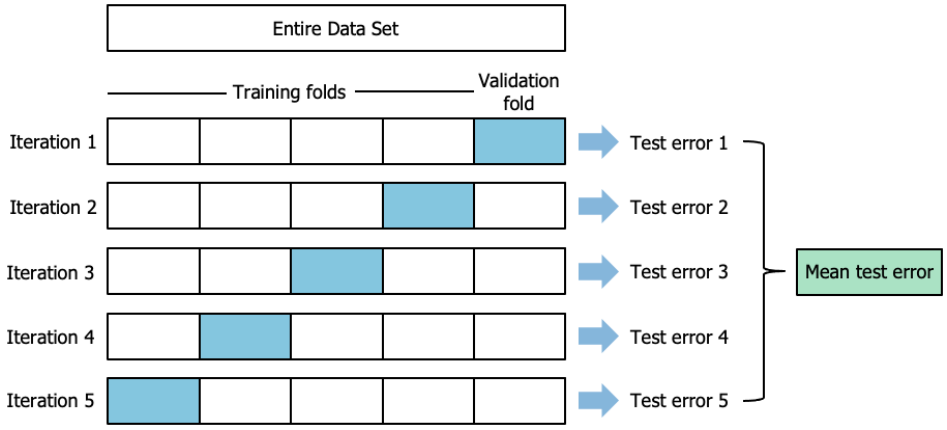

Figure 5. Overview of k-fold CV method, for $k = 5$.

The third category of CV methods is the Nested CV, where CV is used for hyperparameter tuning and error estimation simultaneously. Nested CV methods consider a testing set, in addition to the training and validation sets of the previous categories. The k*l-fold CV splits the data set into k sets and iteratively, each of the k sets is split into l sets, where $l - 1$ are used for training and the other for validation. The inner sets are used to train the model and tune its hyperparameters and the outer is used for testing in order to provide an unbiased evaluation of the model fit. A special case of k*l-fold CV is the k-fold CV with validation and testing sets for $l = k - 1$. One by one, a testing set is selected, and one by one, one of the remaining sets is used as a validation set while the other $k - 2$ sets are used as training sets until all possible combinations have been evaluated.

4.3. Model Evaluation

Following the training and tuning of the model on the training and validation sets explained in Section 4.2 comes model evaluation. This step includes methods and metrics to evaluate that the model indeed performs adequately for the purpose for which it was developed. This step includes utilizing a training set based on which prediction is performed and the evaluation of performance indices. Moreover, evaluation refers to the verification

of the model's assumptions after the predicted values are collected. One such example is linear regression models, the assumptions of which are summarized below [177]:

1. Linear relationship between each feature and each target variable: this assumption can be verified in the testing set by constructing scatter plots of each output vs each feature.
2. Homoscedasticity, i.e., constant residual variance for all the values of a feature: this assumption can be verified by plotting the residuals vs. each feature in the testing set.
3. Independence of residual observations, which is the same as the independence of target variable observations (commonly violated in time series data): this can be verified by checking that the autocorrelation of the residual observations is non-zero with the Durbin–Watson test since that would indicate sample dependence.
4. Normality of the target variable observations: this can be verified by constructing QQ plots of the residuals against the theoretical normal distribution and observing the straightness of the produced line.

It is worth mentioning that if some of the above assumptions do not hold, feature transformations can be applied to satisfy them, such as the ones explained in Section 3.2 of this dissertation. For additional solutions, in order to conform with the linear regression model assumptions, the reader is referred to [177]. After model training, the assumptions above should also be verified on the testing set.

Supervised regression and classification models are not only evaluated based on their modeling assumptions but also based on performance metrics. Some commonly used performance indices in supervised learning are summarized in Table 6, where the following shortcuts are used: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TPs/TNs are when the model predicts an observation to belong/not belong to a class and it does/does not indeed belong to that class. Equivalently, FPs/FNs are when the model predicts that an observation belongs/does not belong to a class and it does not/does belong to that class. Note that N is the number of samples, k is the number of independent variables, y^i is an observed value, $\hat{y^i}$ is a predicted value, and $\overline{y}$ is the sample average of the observed data.

Table 6. Summary of most common performance indices commonly used to evaluate the performance of regression and classification models.

Performance Index	Formula	Purpose
Mean Squared Error (MSE)	$\frac{\sum_{i=1}^{N}\left(y^i-\hat{y}^i\right)^2}{N}$	Regression
Root Mean Squared Error (RMSE)	$\sqrt{\frac{\sum_{i=1}^{N}\left(y^i-\hat{y}^i\right)^2}{N}}$	Regression
Mean Absolute Error (MAE)	$\sum_{i=1}^{N}\frac{\lvert y^i-\hat{y}^i\rvert}{N}$	Regression
Mean Absolute Percentage Error (MAPE)	$\sum_{i=1}^{N}\frac{\lvert y^i-\hat{y}^i\rvert}{N}\cdot 100$	Regression
Coefficient of Determination (R^2)	$1-\frac{\sum_{i=1}^{N}\left(y^i-\hat{y}^i\right)^2}{\sum_{i=1}^{N}\left(y^i-\overline{y}^i\right)^2}$	Regression
Adjusted Coefficient of Determination (A-R^2)	$1-\left(\frac{N-1}{N-k-1}\right)\left(1-R^2\right)$	Regression
Confusion Matrix	TP, TN, FP, FN	Classification
Accuracy	$\frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$	Classification
Balanced Accuracy	$\frac{\text{Precision+Recall}}{2}$	Classification
Misclassification	$\frac{\text{FP+FN}}{\text{TP+TN+FP+FN}}$	Classification
F1-Score	$\frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision+Recall}}$	Classification
F-Score	$\left(1+\beta^2\right)\frac{\text{Precision}\cdot\text{Recall}}{\beta^2\text{Precision+Recall}}$	Classification
Receiver Operating Characteristic (ROC)	Graphical	Classification
Area Under Curve (AUC)	Graphical	Classification
Total Operating Characteristic (TOC)	Graphical	Classification

The choice of the evaluation metric is of great importance, as the wrong choice can lead to the optimization of an irrelevant objective or the introduction of bias as, for example, in the case of using accuracy instead of balanced accuracy when the data set is unbalanced in terms of number of data points from each class [178]. The interested researcher may look into [179] for an empirical analysis of different model performance criteria.

5. Model Deployment

The final phase in the machine learning workflow of Figure 2a is the model deployment where businesses aim to harness its value. As model deployment is, for the most part, a task for software developers in the industry, this section is introduced here for completion purposes. Often, non-AI expert academics work closely with the industry to develop customer-tailored ML workflows, which are then deployed by in-house software developers. However, on the academic side, researchers first test their ideas in prototype code with industry data, very often performing unit testing before sharing their code with the industry partner. A short overview of the model deployment steps to follow provides non-AI expert academics with an insight into how their prototype code can be developed more efficiently for integration with other code and extensive testing.

The model deployment process encompasses a series of steps undertaken before, during, and after the model is put into production, effectively facilitating the productization of the model. Model deployment steps include implementing best practices for deploying a model to production, which involves the standardization of code (TDD [180], Object-Oriented Programming (OOP) or Functional Programming paradigms, and the application of Design Patterns [181,182]), rigorous testing (bug identification, performance testing, integration testing, robustness testing, A/B testing [183], etc.), and ensuring the security (deploying in a private network or Virtual Private Cloud (VPC) [184]) and monitoring of the deployed ML model (for the model or data drifts, bugs, failures, etc.). Subsequently, the deployment undergoes continuous updates, which include the integration of new data, additional data resources, transformations, models, etc., into the workflow, with new model versions being continuously delivered and deployed into the product. The inherent complexity of large-scale systems necessitates the implementation of workflows with sound design principles and robust testing. The next subsections provide all the steps involved in transitioning a model into a production environment.

5.1. Testing

This subsection describes the different types of testing that are performed before a model is put into production.

5.1.1. Unit Testing

Testing is a very important component of a machine learning workflow deployment into production for industry purposes and begins with unit testing. Unit testing verifies that software subprograms function as expected in isolation [185] and focuses on scrutinizing components within the software stack rather than predicting the model's output, given its inherent unpredictability. Various testing frameworks like Pytest [186], Unittest [187], JUnit [188], Mockito [189], etc., can help generate dummy data sets and test expected outputs. Multiple scripted scenarios, encompassing both success and failure cases, are executed in the programming language to assess functionality and identify bugs. A comprehensive suite of unit tests is run during each deployment to ensure the code behaves as intended.

5.1.2. Performance Testing

In addition to unit testing, performance testing is frequently employed to assess the time required for training and gather details about the hardware configurations involved in the training process. This information has been proven to be crucial for future development and testing endeavors. Performance tests also evaluate the model's response time under normal and peak loads, ensuring adherence to the Service Level Agreement (SLA) [190]

and preventing compromise in availability due to request timeouts. Collecting statistics on the ML model's responsiveness under varying workloads aids in identifying potential issues and guarantees that the system performs within specified operational parameters.

5.1.3. Integration Testing

The next level of testing is integration testing, a vital component of CI/CD, which tests the integration of a new or updated ML model and the rest of the code within the software stack before promoting it to production. Integration testing verifies the interaction between software components [185] by testing them together based on functional threads or exhaustively all with all (also known as "big bang" testing). For example, an ML model that predicts the number of neighborhood houses sold must seamlessly interact with gateways, web servers, databases, and other networking layers [191]. Therefore, integration tests would be performed in this case to ensure that the interaction of the model with these services goes as expected. This rigorous integration testing helps identify and rectify potential issues, reducing the risk of disruptions in the live system.

5.1.4. System Testing

Most functional failures must have been identified during the unit and integration testing. System testing examines the behavior of the entire system for the purposes of system security, speed, accuracy, and reliability (non-functional requirements) and for proper interfacing with other applications, utilities, hardware devices, and the operating environment [185]. Installation testing is one form of system testing that may appear in software systems that host an ML model [185]. System and acceptance testing are tests used across software engineering, whether ML is involved or not, but they are not always present in the ML pipeline testing process.

5.1.5. Acceptance Testing

Acceptance testing checks the system behavior against the customer's requirements, and the customers undertake, or specify, typical tasks to check that their requirements have been met or that the organization has identified these for the software's target market. The developers of the system may or may not be involved in this testing level [185].

5.1.6. A/B Testing

Organizations derive value from making data-driven decisions regarding a model's real-world performance. A/B testing [192] serves the understanding of the relationship between changes made to the model and the actual outcomes observed in user behavior. The model users are divided into the control group A and the treatment group B. The control group responds to the existing model and the treatment group to the new/modified model. User responses are recorded, and metrics are subsequently computed and compared. This comparison helps to assess the model's performance before deploying it to the entire user base.

5.2. Model Deployment

After successfully completing various tests in the CI/CD pipeline, the approved changes are deployed to the production environment. The deployed model can be executed either through a scheduled process or triggered by certain events. Scheduled model execution, for example, includes generating an overnight report predicting sales for the next day, which runs to provide up-to-date predictions for business planning. Trigger-based execution, on the other hand, involves responding to specific events, such as user recommendations triggered by a user's search keywords, in order to deliver real-time, personalized results, thus enhancing the user's experience.

5.3. Monitoring and Maintenance

The consideration of model and data drift is of paramount importance during the protracted phases of model monitoring and maintenance. Model drift [193], driven by alterations in the statistical properties of the target variable over time, can result in a degradation of the model's predictive performance. Simultaneously, data drift [194] pertains to variations in the distribution of input features, posing challenges to the model's generalization capabilities. In the lifecycle of an ML model after deployment, sustained vigilance entails regular assessment of performance metrics, and juxtaposing them against benchmarks established during the model's training phase. In the context of contemporary MLOps, the automation of drift detection and response mechanisms is increasingly pivotal to upholding model efficacy and alignment with evolving data dynamics.

5.4. Security Considerations

The infrastructure in which the ML model operates often faces threats, attacks, and risks that can jeopardize assets and availability. Therefore, it is imperative to assess and prepare for these scenarios by regularly applying software patches to known vulnerabilities. Organizations conduct thorough audits of open-source code for any vulnerabilities before incorporating it into production. A recommended strategy to mitigate risk is the creation of internally approved repositories for various versions of software used in building the model and the overall software stack. This centralized control over software versions ensures that only vetted and secure components are employed, reducing the likelihood of vulnerabilities compromising the system. If the ML model is exclusively serving customers within the organization, it is advisable to deploy the code in a private network or a VPC when hosted in the cloud. Adhering to the Principle of Least Privilege (PoLP) [195] is crucial when configuring both the software and ML model, regardless of whether the audience is internal or external. By implementing PoLP, access permissions are restricted to the minimum necessary for users or systems to perform their intended tasks. This helps reduce the potential attack surface and enhances overall security. In addition, monitoring usage patterns helps detect and mitigate any potential Distributed Denial-of-Service (DDoS) [196] attacks on the system. By closely observing network traffic and user interactions, abnormal patterns indicative of a DDoS attack can be identified and appropriate measures can be taken to ensure the continued availability and performance of the ML model.

To conclude, model production code is developed via TDD, which continuously creates tests to fail the system's operation and then upgrades the system to pass the test [180]. This is a slow process, but it leads to fewer bugs and re-usable code through various levels of testing [185,197] as shown in Figure 2c. Testing, deployment, monitoring, maintenance, and security follow the development. For comprehensive insights into both static and dynamic testing, the book by Lewis [198] is highly recommended. For guidance on adhering to good coding practices, it is advised to consult the works of Martin [199] and Thomas et al. [200]. These references offer valuable resources for researchers seeking in-depth knowledge in the domains of coding and testing practices for software development.

6. Automation in Machine Learning Workflows

Section 3.2.4 analyzed the automation of feature extraction. This section focuses on the automation of the entire machine learning workflow and optimization of pipeline hyperparameters and is often referred to as AutoML. According to [201], AutoML makes decisions in a data-driven, objective, and automated way, with the user only providing the data, and AutoML automatically determining the best approach. AutoML is particularly useful for domain scientists who do not focus on learning in-depth machine learning practices, as well as big data and Deep Learning applications [201]. The automation of AI expert systems, i.e., AI software with task-specific machine learning workflow for decision-making in the place of a human, is also possible via AutoML. The focal points of automation encompass various facets such as data engineering, feature engineering, model selection, ensembling, algorithm selection, and hyperparameter optimization, as

well as considerations of time, memory, and complexity constraints within the pipeline. Additionally, automation extends to the determination of evaluation and validation metrics. It is noteworthy that while certain aspects are more commonly automated, the spectrum of automated processes is diverse, catering to the nuanced requirements of the machine learning pipeline.

6.1. AutoML Methods

AutoML methods are utilized for tuning machine learning workflow hyperparameters, also known as Automated Hyperparameter Optimization (AHO), including those decided before training and those decided during training. For example, the number of layers and neurons in a NN is tuned before training, while the NN weights are tuned during training. Although hyperparameter optimization has been considered an art, AutoML aims to automate this part of the machine learning workflow to make machine learning more accessible to non-technical professionals [36], and help optimize the functionality of developed pipelines.

According to [201], AHO can be very useful because it reduces the human effort necessary for machine learning and improves machine learning algorithm performance [202,203] and the reproducibility of scientific results [204,205]. Hyperparameter optimization is a very challenging problem because (a) function evaluations can be extremely expensive for large models (i.e., DNN), large data sets, or very complex machine learning workflows, (b) the configuration space can be complex (continuous, discrete, conditional hyperparameters) and high-dimensional, (c) access to the loss function's gradient with respect to the hyperparameters is often impossible, and (d) generalized optimization is not possible, as it depends on the data set size which varies [36]. The most common methods for hyperparameter optimization are summarized below.

1. Black-box hyperparameter optimization:
    (a) Model-free black-box optimization methods include grid search in a finite range, which, however, suffers from the CoD and random search, where random search samples configurations at random until a certain budget for the search is exhausted [206]. This works better than grid search when some hyperparameters are much more important than others, which is very often the case [206]. Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) [207], is one of the most competitive black-box optimization algorithms.
    (b) Bayesian optimization has gained interest due to DNN tuning for image classification [203,208], speech recognition [209] and neural language modeling [202]. For an in-depth introduction to Bayesian optimization, the interested reader is referred to [210,211]. Many recent advances in Bayesian optimization do not treat hyperparameter tuning as a black-box anymore, i.e., multi-fidelity hyperparameter turning, Bayesian optimization with meta-learning, and Bayesian optimization taking the pipeline structure into account [212,213].
2. Multi-fidelity optimization: These methods are less costly than black-box optimization methods, which approximately assess the quality of hyperparameter settings. Multi-fidelity methods introduce heuristics inside an algorithm, using low-fidelity approximations of the actual loss function to reduce runtime. Such heuristics include hyperparameter tuning on a small data subset or feature subset and training for a few iterations by using CV or down-sampled images. Learning curve-based prediction for early stopping is used, as well as Bandit-based (successive halving [214] and Hyperband [215]) algorithms for algorithm selection based on low-fidelity algorithm approximations. Moreover, Bayesian Optimization Hyperband (BOHB) [216] combines Bayesian optimization and HyperBand to achieve a combination of strong anytime performance (quick improvements in the beginning by using low fidelities in HyperBand) and strong final performance (good performance in the long run by replacing HyperBand's random search by Bayesian optimization). For adaptive fidelity options, see [36].

Another AutoML family of methods, besides AHO, is meta-learning. This family aims to systematize the developer's experience based on which a new model is built. First, meta-data need to be collected that describe previously learned tasks and models and include algorithm configurations and hyperparameter values, workflow architectures, model evaluations (accuracy and training time), optimal model parameters, and meta-features. Second, learning from meta-data needs to take place in order to extract knowledge that can guide future optimal models and tasks. Examples of such methods are transfer learning [217] and few-shot learning [218]. For more related content on learning from model evaluations, task properties, and prior models, please see [36]. A very popular example of AutoML and the aforementioned methods is neural architecture search (NAS), which aims to automate the discovery of new NN architectures [219]. Core approaches to NAS can be found in Table 7.

Table 7. Overview of main NAS methods in AutoML.

Method	Approach to Speed-Up	Paper
Lower fidelity estimates	Less epochs, data subsets, downscaled models/data, etc.	[215,216,220–223]
Learning curve extrapolation	Performance extrapolated after few epochs	[224–227]
Weight inheritance/network morphisms	Models warm-started with inherited weights	[228–232]
One-Shot models/weight sharing	One-shot model's weights shared across architectures	[233–238]

The scalability of AHO is an open-ended research topic that seems to concern multi-fidelity optimization, gradient-based methods, and meta-learning methods [36]. Parallel computing is expected to play an important role in the scalability of AHO, with parallel Bayesian optimization already implemented [239]. With the only exception being DNN [203,208,210,216,240–243], Bayesian optimization has not yet been applied successfully to data sets larger than a few thousand data points. For solutions related to overfitting, generalization, and arbitrary pipeline sizes, the reader is referred to [36]. State-of-the-art approaches to scalable AutoML with data privacy include federated learning [244].

6.2. AutoML Systems

AutoML systems, also known as pipeline optimizers, are popular software that try to automate the machine learning workflow. An overview of available AutoML systems is provided in Table 8. For a complete overview of machine learning algorithms and coding languages supported by these systems, the reader is referred to [245]. Semi-automated pipeline optimizers AutoComplete [246] and PennAI are also mentioned in [245]. A performance comparison for AutoML systems is provided in [247].

Table 8. Overview of available AutoML systems (DE = data engineering, FE = feature engineering).

Software	Problem Automated	AutoML Method	Paper
Auto-WEKA	CASH	Bayesian optimization	[248]
Auto-WEKA 2.0	CASH with parallel runs	Bayesian optimization	[249]
Hyperopt-Sklearn	Space search of random hyperparameters	Bayesian otimization	[250]
Auto-Sklearn	Improved CASH with algorithm ensembles	Bayesian optimization	[251,252]
TPOT	Classification with FE	GeP	[253]
Auto-Net	Automates DNN tuning	Bayesian optimization	[36]
Auto-Net 2.0	Automates DNN tuning	BOHB	[36]
Automatic Statistician	Automates data science	Various	[36]
AutoPytorch	Algo. selection, ensemble constr., hyperpar. tuning	Bayesian opt., meta-learn.	[254]
AutoKeras	NAS, hyperpar. tuning in DNN	Bayesian opt. guides network morphism	[255]

Table 8. *Cont.*

Software	Problem Automated	AutoML Method	Paper
NNI	NAS, hyperpar. tuning, model compression, FE	One-shot modes, etc.	[256,257]
TPOT	Hyperpar. tuning, model selection	GeP	[253]
AutoGluon	Hyperpar. tuning	-	[258]
H2O	DE, FE, hyperpar. tuning, ensemble model selection	Random grid search, Bayesian opt.	[259]
FEDOT	Hyperparameter tuning	Evolutionary algorithms	-
Auto-Sklearn 2	Model selection	Meta-learning, bandit strategy	[252]
FLAML	Algorithm selection, hyperpar. tuning	Search strategies	[260]
AutoGluon-TS	Ensemble constr. for time-series forecasting	Probabilistic time-series	[261]
AutoTS	Time-series data analysis	Various	[262]

7. A Supervised Classification Workflow Example

An example of supervised classification machine learning workflow, based on this paper's methodology, is presented in Figure 6 and describes the approach taken by the IARPA-funded researchers in [18] to classify a room as occupied or empty (binary classification) from one CO_2 and one temperature sensor measurement. Applications include energy efficiency, indoor air quality, emergency evacuation, and other applications. The data streams collected from the two sensors were saved locally, where the data-engineering code treated the incoming values for missing data and timestamp consistency. Target variable values (occupancy class 0 or 1) were collected from the humans involved in the experiment, for model training and validation. Following that, a new feature, the HVAC state, was generated from the application of a domain-expertise deterministic transformation on the CO_2 and temperature data, which helps enrich the input–output correlations by introducing additional information to this poor-input experiment (limited sensors and limited data set challenge). Additional feature transformation took place by locally smoothing high-frequency noise on the CO_2 data with a Savitky–Golay (FIR) filter [263], and feature extraction by producing the numerical derivatives of the smoothed CO_2 signal and lagged inputs in real-time. All the input invariant features were utilized for training a supervised binary classifier, thus skipping any feature selection, in this feature-poor application, where all features were proved highly important. Although the automated feature extraction methods proposed in this paper may have revealed better features, or the extraction of the same features with less human labor, this opportunity was not taken advantage of in the [18] project, and will remain as future work for the authors. A feed-forward neural network was trained on all the aforementioned features, with its architecture optimized manually via the tracking of the training, validation, and testing errors, according to the bias–variance decomposition principle analyzed in this paper as opposed to the AutoML methods now available and presented in Section 6. The model evaluation took place via several classification metrics, including accuracy, balanced accuracy, F1-score, and custom application-related metrics (success rate, average detection delay, etc.). Model deployment is missing from this academic project. A rigorous methodology similar to the one presented in this paper was followed in [18], and resulted in highly accurate and mathematically rigorous results.

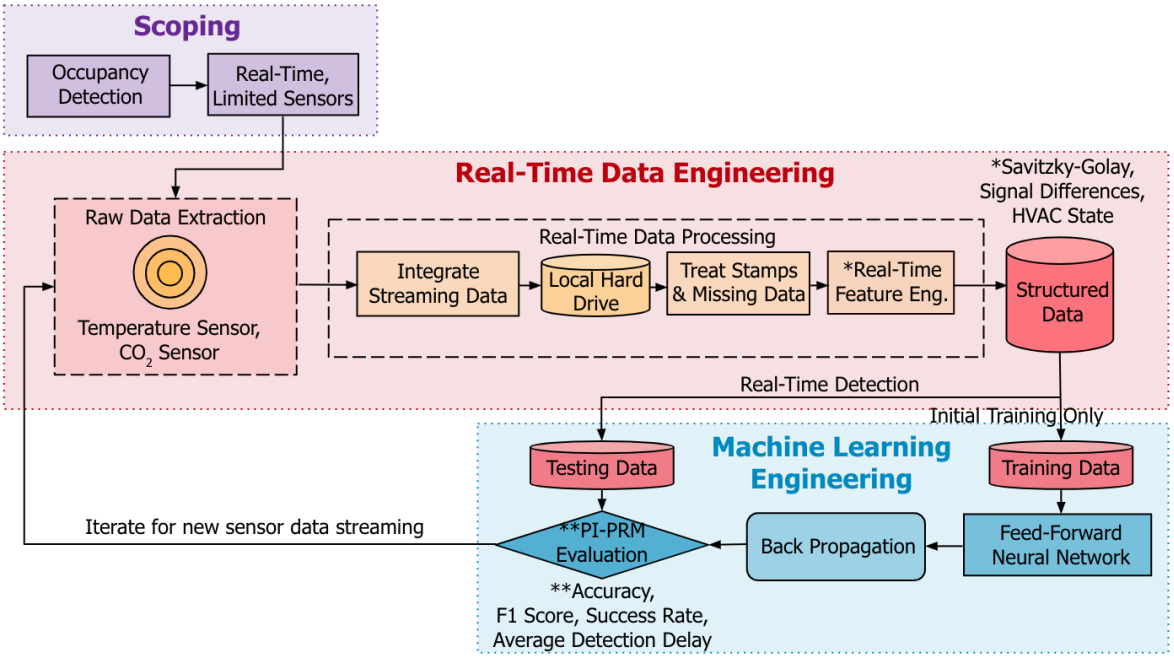

Figure 6. Machine learning workflow example developed by academic researchers for occupancy detection, based on the methodology proposed in this paper. Model deployment and automation were not part of this academic work [18].

8. Discussion

This work provides an all-inclusive supervised machine learning workflow development and automation methodology for numerical and categorical data, along with an adequate literature review and associated industry practices. The machine learning workflow is analyzed from the early project brainstorming and scoping stage to data engineering, feature engineering, machine learning model engineering, workflow automation, and model deployment. Guidance is provided to the non-AI expert academic on how to develop and integrate the pieces of a rigorous, complete, functional, and optimized machine learning workflow for their application, without missing important sub-modules, whether it is for their own research or for a government or industry partner. Important AI principles (i.e., bias–variance decomposition, curse of dimensionality, model complexity, overfitting, model sensitivity to feature assumptions and scaling, output interpretability, etc.) are explained, and their utilization in making important algorithm or tuning choices in the workflow is emphasized. State-of-the-art tools on feature extraction automation, neural architecture search, model selection, hyperparameter tuning, algorithm selection, model compression, etc., are provided and explained (i.e., Bayesian optimization, Genetic Programming, and random grid search) for the optimization of the machine learning workflow under development.

Author Contributions: Conceptualization: S.I.K. and D.N.M.; methodology: S.I.K.; investigation: S.I.K. and A.T.R.; writing: S.I.K., A.T.R. and A.P.B.; visualization: S.I.K.; supervision: O.J.P.F. and D.N.M.; project administration: D.N.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: Author Archana Tikayat Ray was employed by the company AI Fusion Technologies. Author Anirudh Prabhakara Bhat was employed by the company Amazon. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AHO	Automated Hyperparameter Optimization
AIC	Akaike Information Criterion
ALA	Adaptive Linear Approximation
API	Application Programming Interface
AUC	Area Under Curve
Auto-WEKA	Automatic Model Selection and Hyperparameter Optimization
BDW	Best Daubechies Wavelet Coefficients
BFC	Best Fourier Coefficients
BIC	Bayesian Information Criterion
BOHB	Bayesian Optimization Hyperband
CART	Classification and Regression Tree
CASH	Combined Algorithm Selection and Hyperparameter optimization
CI/CD	Continuous Integration Continuous Delivery or Deployment
CMA-ES	Covariance Matrix Adaption Evolutionary Strategy
CoD	Curse of Dimensionality
CV	Cross-Validation
DB	Database
DDoS	Distributed Denial-of-Service
DevOps	Development Operations
DFS	Deep Feature Synthesis
DGUFS	Dependence Guided Unsupervised Feature Selection
DNN	Deep Neural Network
DT	Decision Tree
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
EUFS	Embedded Unsupervised Feature Selection
FIR	Finite Impulse Response
FN	False Negative
FP	False Positive
FSFS	Feature Selection with Feature Similarity
GeP	Genetic Programming
GP	Gaussian Process
HVAC	Heating Ventilation and Air Conditioning
IARPA	Intelligence Advanced Research Projects Activity
KDD	Knowledge Discovery from Data
kNN	k-Nearest Neighbors
LARS	Lasso Regression
LBFGS	Broyden–Fletcher–Goldfarb–Shanno
LDS	Linear Discriminant Analysis
LLC-fs	Local Learning-based Clustering with feature selection
LR	Logistic Regression
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MCFS	Multi-Cluster Feature Selection
MDI	Mean Decrease in Impurity
MDL	Minimum Description Length
MICI	Maximal Information Compression Index
ML	Machine Learning
MLOps	Machine Learning Operations

MRDTL	Multi-Relational Decision Tree Learning
MSE	Mean Squared Error
NAS	Neural Automated Search
NDFS	Non-negative Discriminative Feature Selection
NN	Neural Network
NNI	Neural Network Intelligence
OLS	Ordinary Least Squares
OoB	Out-of-Bag
OOP	Object Oriented Programming
PoLP	Principle of Least Privilege
PCA	Principal Component Analysis
REFSVM	Recursive Feature Elimination Support Vector Machines
RF	Random Forest
RICA	Reconstruction Independent Component Analysis
RMSE	Root Mean Squared Error
ROC	Receiver Operating Characteristic
RRFS	Relevance Redundancy Feature Selection
SLA	Service Level Agreement
SQL	Structured Query Language
SRM	Structural Risk Minimization
SS-SFS	Simplified Silhouette Sequential Forward Selection
SVD	Singular Value Decomposition
SVM	Support Vector Machines
TDD	Test-Driven Development
TN	True Negative
TOC	Total Operating Characteristic
TP	True Positive
TPOT	Tree-based Pipeline Optimization Tool
UDFS	Unsupervised Discriminative Feature Selection
VPC	Virtual Private Cloud

References

1. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
2. Bravo-Rocca, G.; Liu, P.; Guitart, J.; Dholakia, A.; Ellison, D.; Falkanger, J.; Hodak, M. Scanflow: A multi-graph framework for Machine Learning workflow management, supervision, and debugging. *Expert Syst. Appl.* **2022**, *202*, 117232. [CrossRef]
3. Bala, A.; Chana, I. Intelligent failure prediction models for scientific workflows. *Expert Syst. Appl.* **2015**, *42*, 980–989. [CrossRef]
4. Quemy, A. Two-stage optimization for machine learning workflow. *Inf. Syst.* **2020**, *92*, 101483. [CrossRef]
5. Grabska, E.; Frantz, D.; Ostapowicz, K. Evaluation of machine learning algorithms for forest stand species mapping using Sentinel-2 imagery and environmental data in the Polish Carpathians. *Remote Sens. Environ.* **2020**, *251*, 112103. [CrossRef]
6. Liu, R.; Misra, S. A generalized machine learning workflow to visualize mechanical discontinuity. *J. Pet. Sci. Eng.* **2022**, *210*, 109963. [CrossRef]
7. He, S.; Wang, Y.; Zhang, Z.; Xiao, F.; Zuo, S.; Zhou, Y.; Cai, X.; Jin, X. Interpretable machine learning workflow for evaluation of the transformation temperatures of TiZrHfNiCoCu high entropy shape memory alloys. *Mater. Des.* **2023**, *225*, 111513. [CrossRef]
8. Zhou, Y.; Li, G.; Dong, J.; Xing, X.h.; Dai, J.; Zhang, C. MiYA, an efficient machine-learning workflow in conjunction with the YeastFab assembly strategy for combinatorial optimization of heterologous metabolic pathways in Saccharomyces cerevisiae. *Metab. Eng.* **2018**, *47*, 294–302. [CrossRef] [PubMed]
9. Wong, W.K.; Joglekar, M.V.; Saini, V.; Jiang, G.; Dong, C.X.; Chaitarvornkit, A.; Maciag, G.J.; Gerace, D.; Farr, R.J.; Satoor, S.N.; et al. Machine learning workflows identify a microRNA signature of insulin transcription in human tissues. *Iscience* **2021**, *24*, 102379. [CrossRef] [PubMed]
10. Paudel, D.; Boogaard, H.; de Wit, A.; Janssen, S.; Osinga, S.; Pylianidis, C.; Athanasiadis, I.N. Machine learning for large-scale crop yield forecasting. *Agric. Syst.* **2021**, *187*, 103016. [CrossRef]
11. Haghighatlari, M.; Hachmann, J. Advances of machine learning in molecular modeling and simulation. *Curr. Opin. Chem. Eng.* **2019**, *23*, 51–57. [CrossRef]
12. Reker, D. Practical considerations for active machine learning in drug discovery. *Drug Discov. Today Technol.* **2019**, *32*, 73–79. [CrossRef] [PubMed]
13. Narayanan, H.; Dingfelder, F.; Butté, A.; Lorenzen, N.; Sokolov, M.; Arosio, P. Machine learning for biologics: Opportunities for protein engineering, developability, and formulation. *Trends Pharmacol. Sci.* **2021**, *42*, 151–165. [CrossRef] [PubMed]

14. Jeong, S.; Kwak, J.; Lee, S. Machine learning workflow for the oil uptake prediction of rice flour in a batter-coated fried system. *Innov. Food Sci. Emerg. Technol.* **2021**, *74*, 102796. [CrossRef]
15. Li, W.; Niu, Z.; Shang, R.; Qin, Y.; Wang, L.; Chen, H. High-resolution mapping of forest canopy height using machine learning by coupling ICESat-2 LiDAR with Sentinel-1, Sentinel-2 and Landsat-8 data. *Int. J. Appl. Earth Obs. Geoinf.* **2020**, *92*, 102163. [CrossRef]
16. Lv, A.; Cheng, L.; Aghighi, M.A.; Masoumi, H.; Roshan, H. A novel workflow based on physics-informed machine learning to determine the permeability profile of fractured coal seams using downhole geophysical logs. *Mar. Pet. Geol.* **2021**, *131*, 105171. [CrossRef]
17. Gharib, A.; Davies, E.G. A workflow to address pitfalls and challenges in applying machine learning models to hydrology. *Adv. Water Resour.* **2021**, *152*, 103920. [CrossRef]
18. Kampezidou, S.I.; Ray, A.T.; Duncan, S.; Balchanos, M.G.; Mavris, D.N. Real-time occupancy detection with physics-informed pattern-recognition machines based on limited CO2 and temperature sensors. *Energy Build.* **2021**, *242*, 110863. [CrossRef]
19. Fu, H.; Kampezidou, S.; Sung, W.; Duncan, S.; Mavris, D.N. A Data-driven Situational Awareness Approach to Monitoring Campus-wide Power Consumption. In Proceedings of the 2018 International Energy Conversion Engineering Conference, Cincinnati, OH, USA, 9–11 July 2018; p. 4414.
20. Kampezidou, S.; Wiegman, H. Energy and power savings assessment in buildings via conservation voltage reduction. In Proceedings of the 2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), Washington, DC, USA, 23–26 April 2017; pp. 1–5.
21. Kampezidou, S.I.; Romberg, J.; Vamvoudakis, K.G.; Mavris, D.N. Scalable Online Learning of Approximate Stackelberg Solutions in Energy Trading Games with Demand Response Aggregators. *arXiv* **2023**, arXiv:2304.02086.
22. Kampezidou, S.I.; Romberg, J.; Vamvoudakis, K.G.; Mavris, D.N. Online Adaptive Learning in Energy Trading Stackelberg Games with Time-Coupling Constraints. In Proceedings of the 2021 American Control Conference (ACC), New Orleans, LA, USA, 25–28 May 2021; pp. 718–723.
23. Gao, Z.; Kampezidou, S.I.; Behere, A.; Puranik, T.G.; Rajaram, D.; Mavris, D.N. Multi-level aircraft feature representation and selection for aviation environmental impact analysis. *Transp. Res. Part C Emerg. Technol.* **2022**, *143*, 103824. [CrossRef]
24. Tikayat Ray, A.; Cole, B.F.; Pinon Fischer, O.J.; White, R.T.; Mavris, D.N. aeroBERT-Classifier: Classification of Aerospace Requirements Using BERT. *Aerospace* **2023**, *10*, 279. [CrossRef]
25. Tikayat Ray, A.; Pinon Fischer, O.J.; Mavris, D.N.; White, R.T.; Cole, B.F. aeroBERT-NER: Named-Entity Recognition for Aerospace Requirements Engineering using BERT. In Proceedings of the AIAA SCITECH 2023 Forum, National Harbor, MD, USA, 23–27 January 2023. [CrossRef]
26. Tikayat Ray, A. Standardization of Engineering Requirements Using Large Language Models. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2023. [CrossRef]
27. Tikayat Ray, A.; Cole, B.F.; Pinon Fischer, O.J.; Bhat, A.P.; White, R.T.; Mavris, D.N. Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models. *Systems* **2023**, *11*, 352. [CrossRef]
28. Shrivastava, R.; Sisodia, D.S.; Nagwani, N.K. Deep neural network-based multi-stakeholder recommendation system exploiting multi-criteria ratings for preference learning. *Expert Syst. Appl.* **2023**, *213*, 119071. [CrossRef]
29. van Dinter, R.; Catal, C.; Tekinerdogan, B. A decision support system for automating document retrieval and citation screening. *Expert Syst. Appl.* **2021**, *182*, 115261. [CrossRef]
30. Li, X.; Zheng, J.; Li, M.; Ma, W.; Hu, Y. One-shot neural architecture search for fault diagnosis using vibration signals. *Expert Syst. Appl.* **2022**, *190*, 116027. [CrossRef]
31. Kim, J.; Comuzzi, M. A diagnostic framework for imbalanced classification in business process predictive monitoring. *Expert Syst. Appl.* **2021**, *184*, 115536. [CrossRef]
32. Jin, Y.; Carman, M.; Zhu, Y.; Xiang, Y. A technical survey on statistical modelling and design methods for crowdsourcing quality control. *Artif. Intell.* **2020**, *287*, 103351. [CrossRef]
33. Boeschoten, S.; Catal, C.; Tekinerdogan, B.; Lommen, A.; Blokland, M. The automation of the development of classification models and improvement of model quality using feature engineering techniques. *Expert Syst. Appl.* **2023**, *213*, 118912. [CrossRef]
34. Zhang, Y.; Kwong, S.; Wang, S. Machine learning based video coding optimizations: A survey. *Inf. Sci.* **2020**, *506*, 395–423. [CrossRef]
35. Moniz, N.; Cerqueira, V. Automated imbalanced classification via meta-learning. *Expert Syst. Appl.* **2021**, *178*, 115011. [CrossRef]
36. Waring, J.; Lindvall, C.; Umeton, R. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artif. Intell. Med.* **2020**, *104*, 101822. [CrossRef]
37. Kefalas, M.; Baratchi, M.; Apostolidis, A.; van den Herik, D.; Bäck, T. Automated machine learning for remaining useful life estimation of aircraft engines. In Proceedings of the 2021 IEEE International Conference on Prognostics and Health Management (ICPHM), Detroit, MI, USA, 7–9 June 2021; pp. 1–9.
38. Tikayat Ray, A.; Bhat, A.P.; White, R.T.; Nguyen, V.M.; Pinon Fischer, O.J.; Mavris, D.N. Examining the Potential of Generative Language Models for Aviation Safety Analysis: Case Study and Insights Using the Aviation Safety Reporting System (ASRS). *Aerospace* **2023**, *10*, 770. [CrossRef]
39. Hayashi, M.; Tamai, K.; Owashi, Y.; Miura, K. Automated machine learning for identification of pest aphid species (Hemiptera: Aphididae). *Appl. Entomol. Zool.* **2019**, *54*, 487–490. [CrossRef]

40. Espejo-Garcia, B.; Malounas, I.; Vali, E.; Fountas, S. Testing the Suitability of Automated Machine Learning for Weeds Identification. *AI* **2021**, *2*, 34–47. [CrossRef]
41. Koh, J.C.; Spangenberg, G.; Kant, S. Automated machine learning for high-throughput image-based plant phenotyping. *Remote Sens.* **2021**, *13*, 858. [CrossRef]
42. Warnett, S.J.; Zdun, U. Architectural design decisions for the machine learning workflow. *Computer* **2022**, *55*, 40–51. [CrossRef]
43. Khalilnejad, A.; Karimi, A.M.; Kamath, S.; Haddadian, R.; French, R.H.; Abramson, A.R. Automated pipeline framework for processing of large-scale building energy time series data. *PLoS ONE* **2020**, *15*, e0240461. [CrossRef]
44. Michael, N.; Cucuringu, M.; Howison, S. OFTER: An Online Pipeline for Time Series Forecasting. *arXiv* **2023**, arXiv:2304.03877. [CrossRef]
45. Hapke, H.; Nelson, C. *Building Machine Learning Pipelines*; O'Reilly Media: Sebastopol, CA, USA, 2020.
46. Kolodiazhnyi, K. *Hands-On Machine Learning with C++: Build, Train, and Deploy End-To-End Machine Learning and Deep Learning Pipelines*; Packt Publishing Ltd.: Burmingham, UK, 2020.
47. El-Amir, H.; Hamdy, M. *Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow*; Apress: New York, NY, USA, 2019.
48. Zheng, A.; Casari, A. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2018.
49. Meisenbacher, S.; Turowski, M.; Phipps, K.; Rätz, M.; Müller, D.; Hagenmeyer, V.; Mikut, R. Review of automated time series forecasting pipelines. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2022**, *12*, e1475. [CrossRef]
50. Wang, M.; Cui, Y.; Wang, X.; Xiao, S.; Jiang, J. Machine learning for networking: Workflow, advances and opportunities. *IEEE Netw.* **2017**, *32*, 92–99. [CrossRef]
51. Kreuzberger, D.; Kühl, N.; Hirschl, S. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access* **2023**, *11*, 31866–31879. [CrossRef]
52. di Laurea, I.S. MLOps-Standardizing the Machine Learning Workflow. Ph.D. Thesis, University of Bologna, Bologna, Italy, 2021.
53. Allison, P.D. *Missing Data*; Sage Publications: Los Angeles, CA, USA, 2001.
54. Little, R.J.; Rubin, D.B. *Statistical Analysis with Missing Data*; John Wiley & Sons: Hoboken, NJ, USA, 2019; Volume 793.
55. Candes, E.; Recht, B. Exact matrix completion via convex optimization. *Commun. ACM* **2012**, *55*, 111–119. [CrossRef]
56. Candès, E.J.; Tao, T. The power of convex relaxation: Near-optimal matrix completion. *IEEE Trans. Inf. Theory* **2010**, *56*, 2053–2080. [CrossRef]
57. Candes, E.J.; Plan, Y. Matrix completion with noise. *Proc. IEEE* **2010**, *98*, 925–936. [CrossRef]
58. Johnson, C.R. Matrix completion problems: A survey. In *Proceedings of the Matrix Theory and Applications*; American Mathematical Society: Providence, RI, USA, 1990; Volume 40, pp. 171–198.
59. Recht, B. A simpler approach to matrix completion. *J. Mach. Learn. Res.* **2011**, *12*, 3413–3430.
60. Kennedy, A.; Nash, G.; Rattenbury, N.; Kempa-Liehr, A.W. Modelling the projected separation of microlensing events using systematic time-series feature engineering. *Astron. Comput.* **2021**, *35*, 100460. [CrossRef]
61. Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* **2006**, *19*, 1–16. [CrossRef]
62. Hlupić, T.; Oreščanin, D.; Ružak, D.; Baranović, M. An overview of current data lake architecture models. In Proceedings of the 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 23–27 May 2022; pp. 1082–1087.
63. Vassiliadis, P. A survey of extract–transform–load technology. *Int. J. Data Warehous. Min.* **2009**, *5*, 1–27. [CrossRef]
64. Vassiliadis, P.; Simitsis, A. Extraction, Transformation, and Loading. In *Encyclopedia of Database Systems*; Springer: Boston, MA, USA, 2009; Volume 10.
65. Dash, T.; Chitlangia, S.; Ahuja, A.; Srinivasan, A. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Sci. Rep.* **2022**, *12*, 1040. [CrossRef] [PubMed]
66. Dara, S.; Tumma, P. Feature extraction by using deep learning: A survey. In Proceedings of the 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 29–31 March 2018; pp. 1795–1801.
67. Lee, J.; Bahri, Y.; Novak, R.; Schoenholz, S.S.; Pennington, J.; Sohl-Dickstein, J. Deep neural networks as gaussian processes. *arXiv* **2017**, arXiv:1711.00165.
68. Benoit, K. Linear regression models with logarithmic transformations. *Lond. Sch. Econ.* **2011**, *22*, 23–36.
69. Hastie, T.; Tibshirani, R.; Friedman, J.H.; Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*; Springer: New York, NY, USA, 2009; Volume 2.
70. Piryonesi, S.M.; El-Diraby, T.E. Role of data analytics in infrastructure asset management: Overcoming data size and quality problems. *J. Transp. Eng. Part B Pavements* **2020**, *146*, 04020022. [CrossRef]
71. Breiman, L.; Friedman, J.H.; Olshen, R.A.; Stone, C.J. *Classification and Regression Trees*; Routledge: Abingdon, UK, 2017.
72. Grus, J. *Data Science from Scratch: First Principles with Python*; O'Reilly Media: Sebastopol, CA, USA, 2019.
73. Sharma, V. A Study on Data Scaling Methods for Machine Learning. *Int. J. Glob. Acad. Sci. Res.* **2022**, *1*, 23–33. [CrossRef]
74. Leznik, M.; Tofallis, C. *Estimating Invariant Principal Components Using Diagonal Regression*; University of Hertfordshire: Hatfield, UK, 2005.

75. Ahsan, M.M.; Mahmud, M.P.; Saha, P.K.; Gupta, K.D.; Siddique, Z. Effect of data scaling methods on machine learning algorithms and model performance. *Technologies* **2021**, *9*, 52. [CrossRef]
76. Neter, J.; Kutner, M.H.; Nachtsheim, C.J.; Wasserman, W. *Applied Linear Statistical Models*; Marshall University: Untington, WV, USA, 1996.
77. Yeo, I.K.; Johnson, R.A. A new family of power transformations to improve normality or symmetry. *Biometrika* **2000**, *87*, 954–959. [CrossRef]
78. Fisher, R.A. Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika* **1915**, *10*, 507–521. [CrossRef]
79. Anscombe, F.J. The transformation of Poisson, binomial and negative-binomial data. *Biometrika* **1948**, *35*, 246–254. [CrossRef]
80. Box, G.E.; Cox, D.R. An analysis of transformations. *J. R. Stat. Soc. Ser. B* **1964**, *26*, 211–243. [CrossRef]
81. Holland, S. Transformations of Proportions and Percentages. 2015. Available online: http://stratigrafia.org/8370/rtips/proportions.html (accessed on 10 December 2023).
82. Cormode, G.; Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms* **2005**, *55*, 58–75. [CrossRef]
83. Kessy, A.; Lewin, A.; Strimmer, K. Optimal whitening and decorrelation. *Am. Stat.* **2018**, *72*, 309–314. [CrossRef]
84. Higham, N.J. *Analysis of the Cholesky Decomposition of a Semi-Definite Matrix*; University of Manchester: Manchester, UK, 1990.
85. Jain, A.K.; Duin, R.P.W.; Mao, J. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 4–37. [CrossRef]
86. Lakhina, A.; Crovella, M.; Diot, C. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM Comput. Commun. Rev.* **2004**, *34*, 219–230. [CrossRef]
87. Han, K.; Wang, Y.; Zhang, C.; Li, C.; Xu, C. Autoencoder inspired unsupervised feature selection. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 2941–2945.
88. Solorio-Fernández, S.; Carrasco-Ochoa, J.A.; Martínez-Trinidad, J.F. A review of unsupervised feature selection methods. *Artif. Intell. Rev.* **2020**, *53*, 907–948. [CrossRef]
89. Li, Z.; Yang, Y.; Liu, J.; Zhou, X.; Lu, H. Unsupervised feature selection using nonnegative spectral analysis. In Proceedings of the AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–26 July 2012; Volume 26, pp. 1026–1032.
90. Yu, L.; Liu, H. Feature selection for high-dimensional data: A fast correlation-based filter solution. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), Washington, DC, USA, 21–24 August 2003; pp. 856–863.
91. Mitra, P.; Murthy, C.; Pal, S.K. Unsupervised feature selection using feature similarity. *IEEE Trans. Pattern Anal. Mach. Intell.* **2002**, *24*, 301–312. [CrossRef]
92. He, X.; Cai, D.; Niyogi, P. Laplacian score for feature selection. In Proceedings of the 18th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 5–8 December 2005; Volume 18.
93. Ferreira, A.J.; Figueiredo, M.A. An unsupervised approach to feature discretization and selection. *Pattern Recognit.* **2012**, *45*, 3048–3060. [CrossRef]
94. Park, C.H. A feature selection method using hierarchical clustering. In Proceedings of the Mining Intelligence and Knowledge Exploration, Tamil Nadu, India, 18–20 December 2013; pp. 1–6.
95. Cai, D.; Zhang, C.; He, X. Unsupervised feature selection for multi-cluster data. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 25–28 July 2010; pp. 333–342.
96. Yang, Y.; Shen, H.T.; Ma, Z.; Huang, Z.; Zhou, X. ℓ 2, 1-norm regularized discriminative feature selection for unsupervised learning. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, Barcelona, Spain, 16–22 July 2011.
97. Dy, J.G.; Brodley, C.E. Feature selection for unsupervised learning. *J. Mach. Learn. Res.* **2004**, *5*, 845–889.
98. Breaban, M.; Luchian, H. A unifying criterion for unsupervised clustering and feature selection. *Pattern Recognit.* **2011**, *44*, 854–865. [CrossRef]
99. Hruschka, E.R.; Covoes, T.F. Feature selection for cluster analysis: An approach based on the simplified Silhouette criterion. In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), Vienna, Austria, 28–30 November 2005; Volume 1, pp. 32–38.
100. Law, M.H.; Figueiredo, M.A.; Jain, A.K. Simultaneous feature selection and clustering using mixture models. *IEEE Trans. Pattern Anal. Mach. Intell.* **2004**, *26*, 1154–1166. [CrossRef] [PubMed]
101. Zeng, H.; Cheung, Y.m. Feature selection and kernel learning for local learning-based clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *33*, 1532–1547. [CrossRef]
102. Wang, S.; Pedrycz, W.; Zhu, Q.; Zhu, W. Unsupervised feature selection via maximum projection and minimum redundancy. *Knowl. Based Syst.* **2015**, *75*, 19–29. [CrossRef]
103. Guo, J.; Zhu, W. Dependence guided unsupervised feature selection. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
104. Liu, H.; Motoda, H. *Feature Extraction, Construction and Selection: A Data Mining Perspective*; Springer: New York, NY, USA, 1998; Volume 453.
105. Kuhn, M.; Johnson, K. *Applied Predictive Modeling*; Springer: New York, NY, USA, 2013; Volume 26.

106. Hastie, T.; Tibshirani, R.; Wainwright, M. Statistical learning with sparsity. *Monogr. Stat. Appl. Probab.* **2015**, *143*, 143.
107. Tibshirani, R. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B* **1996**, *58*, 267–288. [CrossRef]
108. Zou, H.; Hastie, T. Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B* **2005**, *67*, 301–320. [CrossRef]
109. Obozinski, G.; Taskar, B.; Jordan, M. *Multi-Task Feature Selection*; Technical Report; Department of Statistics, University of California: Berkeley, CA, USA, 2006; Volume 2.
110. Argyriou, A.; Evgeniou, T.; Pontil, M. Multi-task feature learning. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 4–7 December 2006; Volume 19.
111. Yuan, M.; Lin, Y. Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Ser. B* **2006**, *68*, 49–67. [CrossRef]
112. Kocev, D.; Vens, C.; Struyf, J.; Džeroski, S. Ensembles of multi-objective decision trees. In Proceedings of the European Conference on Machine Learning, Warsaw, Poland, 17–21 September 2007; pp. 624–631.
113. Breiman, L. Bagging predictors. *Mach. Learn.* **1996**, *24*, 123–140. [CrossRef]
114. Elith, J.; Leathwick, J.R.; Hastie, T. A working guide to boosted regression trees. *J. Anim. Ecol.* **2008**, *77*, 802–813. [CrossRef]
115. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
116. Kocev, D.; Džeroski, S.; White, M.D.; Newell, G.R.; Griffioen, P. Using single-and multi-target regression trees and ensembles to model a compound index of vegetation condition. *Ecol. Model.* **2009**, *220*, 1159–1168. [CrossRef]
117. Hastie, T.; Tibshirani, R.; Friedman, J. Boosting and additive trees. In *The Elements of Statistical Learning*; Springer: New York, NY, USA, 2009; pp. 337–387.
118. Madeh Piryonesi, S.; El-Diraby, T.E. Using machine learning to examine impact of type of performance indicator on flexible pavement deterioration modeling. *J. Infrastruct. Syst.* **2021**, *27*, 04021005. [CrossRef]
119. Piryonesi, S.M.; El-Diraby, T.E. Data analytics in asset management: Cost-effective prediction of the pavement condition index. *J. Infrastruct. Syst.* **2020**, *26*, 04019036. [CrossRef]
120. Segal, M.; Xiao, Y. Multivariate random forests. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2011**, *1*, 80–87. [CrossRef]
121. Bellman, R. Adaptive Control Processes: A Guided Tour. *J. R. Stat. Soc. Ser. A* **1962**, *125*, 161–162. [CrossRef]
122. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: New York, NY, USA, 2006.
123. Gao, Z. Representative Data and Models for Complex Aerospace Systems Analysis. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2022.
124. Thudumu, S.; Branch, P.; Jin, J.; Singh, J.J. A comprehensive survey of anomaly detection techniques for high dimensional big data. *J. Big Data* **2020**, *7*, 42. [CrossRef]
125. Katz, G.; Shin, E.C.R.; Song, D. Explorekit: Automatic feature generation and selection. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Barcelona, Spain, 12–15 December 2016; pp. 979–984.
126. Lam, H.T.; Thiebaut, J.M.; Sinn, M.; Chen, B.; Mai, T.; Alkan, O. One button machine for automating feature engineering in relational databases. *arXiv* **2017**, arXiv:1706.00327.
127. Kaul, A.; Maheshwary, S.; Pudi, V. Autolearn: Automated feature generation and selection. In Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM), New Orleans, LA, USA, 18–21 November 2017; pp. 217–226.
128. Tran, B.; Xue, B.; Zhang, M. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Comput.* **2016**, *8*, 3–15. [CrossRef]
129. Khurana, U.; Turaga, D.; Samulowitz, H.; Parthasrathy, S. Cognito: Automated feature engineering for supervised learning. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), Barcelona, Spain, 12–15 December 2016; pp. 1304–1307.
130. Khurana, U.; Samulowitz, H.; Turaga, D. Feature engineering for predictive modeling using reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
131. Nargesian, F.; Samulowitz, H.; Khurana, U.; Khalil, E.B.; Turaga, D.S. Learning Feature Engineering for Classification. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; Volume 17, pp. 2529–2535.
132. Li, H.; Chutatape, O. Automated feature extraction in color retinal images by a model based approach. *IEEE Trans. Biomed. Eng.* **2004**, *51*, 246–254. [CrossRef] [PubMed]
133. Dang, D.M.; Jackson, K.R.; Mohammadi, M. Dimension and variance reduction for Monte Carlo methods for high-dimensional models in finance. *Appl. Math. Financ.* **2015**, *22*, 522–552. [CrossRef]
134. Donoho, D.L. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lect.* **2000**, *1*, 32.
135. Atramentov, A.; Leiva, H.; Honavar, V. A multi-relational decision tree learning algorithm–implementation and experiments. In Proceedings of the International Conference on Inductive Logic Programming, Szeged, Hungary, 29 September–1 October 2003; pp. 38–56.
136. Kanter, J.M.; Veeramachaneni, K. Deep feature synthesis: Towards automating data science endeavors. In Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France, 19–21 October 2015; pp. 1–10.
137. Weimer, D.; Scholz-Reiter, B.; Shpitalni, M. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Ann.* **2016**, *65*, 417–420. [CrossRef]
138. Schneider, T.; Helwig, N.; Schütze, A. Industrial condition monitoring with smart sensors using automated feature extraction and selection. *Meas. Sci. Technol.* **2018**, *29*, 094002. [CrossRef]

139. Laird, P.; Saul, R. Automated feature extraction for supervised learning. In Proceedings of the First IEEE Conference on Evolutionary Computation. IEEEWorld Congress on Computational Intelligence, Orlando, FL, USA, 27–29 June 1994; pp. 674–679.
140. Le, Q.; Karpenko, A.; Ngiam, J.; Ng, A. ICA with reconstruction cost for efficient overcomplete feature learning. In Proceedings of the 24th International Conference on Neural Information Processing Systems, Granada, Spain, 12–15 December 2011; Volume 24.
141. Ngiam, J.; Chen, Z.; Bhaskar, S.; Koh, P.; Ng, A. Sparse filtering. In Proceedings of the 24th International Conference on Neural Information Processing Systems, Granada, Spain, 12–15 December 2011; Volume 24.
142. Nocedal, J.; Wright, S.J. *Numerical Optimization*; Springer: New York, NY, USA, 1999.
143. Mallat, S. Group invariant scattering. *Commun. Pure Appl. Math.* **2012**, *65*, 1331–1398. [CrossRef]
144. Bruna, J.; Mallat, S. Invariant scattering convolution networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1872–1886. [CrossRef]
145. Andén, J.; Mallat, S. Deep scattering spectrum. *IEEE Trans. Signal Process.* **2014**, *62*, 4114–4128. [CrossRef]
146. Mallat, S. Understanding deep convolutional networks. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2016**, *374*, 20150203. [CrossRef]
147. Rizk, Y.; Hajj, N.; Mitri, N.; Awad, M. Deep belief networks and cortical algorithms: A comparative study for supervised classification. *Appl. Comput. Inform.* **2019**, *15*, 81–93. [CrossRef]
148. Rifkin, R.M.; Lippert, R.A. *Notes on Regularized Least Squares*; MIT Press: Cambridge, MA, USA, 2007.
149. Yin, R.; Liu, Y.; Wang, W.; Meng, D. Sketch kernel ridge regression using circulant matrix: Algorithm and theory. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 3512–3524. [CrossRef]
150. Efron, B.; Hastie, T.; Johnstone, I.; Tibshirani, R. Least angle regression. *Ann. Stat.* **2004**, *32*, 407–499. [CrossRef]
151. Bulso, N.; Marsili, M.; Roudi, Y. On the complexity of logistic regression models. *Neural Comput.* **2019**, *31*, 1592–1623. [CrossRef]
152. Belyaev, M.; Burnaev, E.; Kapushev, Y. Exact inference for Gaussian process regression in case of big data with the Cartesian product structure. *arXiv* **2014**, arXiv:1403.6573.
153. Serpen, G.; Gao, Z. Complexity analysis of multilayer perceptron neural network embedded into a wireless sensor network. *Procedia Comput. Sci.* **2014**, *36*, 192–197. [CrossRef]
154. Jain, A.K.; Mao, J.; Mohiuddin, K.M. Artificial neural networks: A tutorial. *Computer* **1996**, *29*, 31–44. [CrossRef]
155. Fleizach, C.; Fukushima, S. *A Naive Bayes Classifier on 1998 KDD Cup*; Technical Report; Department of Computer Science and Engineering, University of California: Los Angeles, CA, USA, 1998.
156. Jensen, F.V.; Nielsen, T.D. *Bayesian Networks and Decision Graphs*; Springer: New York, NY, USA, 2007; Volume 2.
157. Claesen, M.; De Smet, F.; Suykens, J.A.; De Moor, B. Fast prediction with SVM models containing RBF kernels. *arXiv* **2014**, arXiv:1403.0736.
158. Cardot, H.; Degras, D. Online principal component analysis in high dimension: Which algorithm to choose? *Int. Stat. Rev.* **2018**, *86*, 29–50. [CrossRef]
159. Veksler, O. *Nonparametric Density Estimation Nearest Neighbors, KNN*; Haifa University: Haifa, Israel, 2013.
160. Raschka, S. STAT 479: Machine Learning Lecture Notes. Available online: https://pages.stat.wisc.edu/~sraschka/teaching/stat479-fs2019/ (accessed on 10 December 2023).
161. Sani, H.M.; Lei, C.; Neagu, D. Computational complexity analysis of decision tree algorithms. In Proceedings of the Artificial Intelligence XXXV: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, 11–13 December 2018; pp. 191–197.
162. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutorials* **2015**, *18*, 1153–1176. [CrossRef]
163. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.Y. Lightgbm: A highly efficient gradient boosting decision tree. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Volume 30.
164. Cai, D.; He, X.; Han, J. Training linear discriminant analysis in linear time. In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, Cancun, Mexico, 7–12 April 2008; pp. 209–217.
165. Refaeilzadeh, P.; Tang, L.; Liu, H. Cross-validation. In *Encyclopedia of Database Systems*; Springer: New York, NY, USA, 2009; Volume 5, pp. 532–538.
166. Efron, B.; Tibshirani, R.J. *An Introduction to the Bootstrap*; CRC Press: Boca Raton, FL, USA, 1994.
167. Efron, B. Bootstrap methods: Another look at the jackknife. In *Breakthroughs in Statistics*; Springer: New York, NY, USA, 1992; pp. 569–593.
168. Breiman, L. *Bias, Variance, and Arcing Classifiers*; Technical Report; Department of Statistics, University of California: Berkeley, CA, USA, 1996.
169. Syakur, M.; Khotimah, B.; Rochman, E.; Satoto, B.D. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Surabaya, Indonesia, 9 November 2017; Volume 336, p. 012017.
170. Palacio-Niño, J.O.; Berzal, F. Evaluation metrics for unsupervised learning algorithms. *arXiv* **2019**, arXiv:1905.05667.
171. Halkidi, M.; Batistakis, Y.; Vazirgiannis, M. On clustering validation techniques. *J. Intell. Inf. Syst.* **2001**, *17*, 107–145. [CrossRef]
172. Perry, P.O. *Cross-Validation for Unsupervised Learning*; Stanford University: Stanford, CA, USA, 2009.

173. Airola, A.; Pahikkala, T.; Waegeman, W.; De Baets, B.; Salakoski, T. An experimental comparison of cross-validation techniques for estimating the area under the ROC curve. *Comput. Stat. Data Anal.* **2011**, *55*, 1828–1844. [CrossRef]
174. Breiman, L.; Spector, P. Submodel selection and evaluation in regression. The X-random case. *Int. Stat. Rev.* **1992**, *60*, 291–319. [CrossRef]
175. Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, QC, Canada, 20–25 August 1995; Volume 14, pp. 1137–1145.
176. Arlot, S.; Celisse, A. A survey of cross-validation procedures for model selection. *Stat. Surv.* **2010**, *4*, 40–79. [CrossRef]
177. McCulloch, C.E.; Searle, S.R. *Generalized, Linear, and Mixed Models*; John Wiley & Sons: Hoboken, NJ, USA, 2004.
178. Kühl, N.; Hirt, R.; Baier, L.; Schmitz, B.; Satzger, G. How to conduct rigorous supervised machine learning in information systems research: The supervised machine learning report card. *Commun. Assoc. Inf. Syst.* **2021**, *48*, 46. [CrossRef]
179. Caruana, R.; Niculescu-Mizil, A. Data mining in metric space: An empirical analysis of supervised learning performance criteria. In Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; pp. 69–78.
180. Beck, K. *Test-Driven Development: By Example*; Addison-Wesley Professional: Boston, MA, USA, 2003.
181. Washizaki, H.; Uchida, H.; Khomh, F.; Guéhéneuc, Y.G. Studying Software Engineering Patterns for Designing Machine Learning Systems. In Proceedings of the 2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP), Tokyo, Japan, 13–14 December 2019; pp. 49–495. [CrossRef]
182. Gamma, E.; Helm, R.; Johnson, R.; Johnson, R.E.; Vlissides, J.; Booch, G. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley Professional: Boston, MA, USA, 1995.
183. Kohavi, R.; Longbotham, R. Online Controlled Experiments and A/B Testing. *Encycl. Mach. Learn. Data Min.* **2017**, *7*, 922–929.
184. Rajasoundaran, S.; Prabu, A.; Routray, S.; Kumar, S.S.; Malla, P.P.; Maloji, S.; Mukherjee, A.; Ghosh, U. Machine learning based deep job exploration and secure transactions in virtual private cloud systems. *Comput. Secur.* **2021**, *109*, 102379. [CrossRef]
185. Abran, A.; Moore, J.W.; Bourque, P.; Dupuis, R.; Tripp, L. *Software Engineering Body of Knowledge*; IEEE: Piscataway, NJ, USA, 2004; p. 25.
186. Pytest: Helps You Write Better Programs. Available online: https://docs.pytest.org/en/7.4.x/ (accessed on 10 December 2023).
187. Unittest: Unit Testing Framework. Available online: https://docs.python.org/3/library/unittest.html (accessed on 10 December 2023).
188. JUnit. Available online: https://junit.org/junit5 (accessed on 10 December 2023).
189. Mockito. Available online: https://site.mockito.org/ (accessed on 10 December 2023).
190. Ardagna, C.A.; Bena, N.; Hebert, C.; Krotsiani, M.; Kloukinas, C.; Spanoudakis, G. Big Data Assurance: An Approach Based on Service-Level Agreements. *Big Data* **2023**, *11*, 239–254. [CrossRef]
191. Mili, A.; Tchier, F. *Software Testing: Concepts and Operations*; John Wiley & Sons: Hoboken, NJ, USA, 2015.
192. Li, P.L.; Chai, X.; Campbell, F.; Liao, J.; Abburu, N.; Kang, M.; Niculescu, I.; Brake, G.; Patil, S.; Dooley, J.; et al. Evolving software to be ML-driven utilizing real-world A/B testing: Experiences, insights, challenges. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Madrid, Spain, 25–28 May 2021; pp. 170–179.
193. Manias, D.M.; Chouman, A.; Shami, A. Model Drift in Dynamic Networks. *IEEE Commun. Mag.* **2023**, *61*, 78–84. [CrossRef]
194. Wani, D.; Ackerman, S.; Farchi, E.; Liu, X.; Chang, H.w.; Lalithsena, S. Data Drift Monitoring for Log Anomaly Detection Pipelines. *arXiv* **2023**, arXiv:2310.14893.
195. Schneider, F. Least privilege and more [computer security]. *IEEE Secur. Priv.* **2003**, *1*, 55–59. [CrossRef]
196. Mahjabin, T.; Xiao, Y.; Sun, G.; Jiang, W. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *Int. J. Distrib. Sens. Netw.* **2017**, *13*, 1550147717741463. [CrossRef]
197. Certified Tester Foundation Level (CTFL) Syllabus. Technical Report, International Software Testing Qualifications Board, Version 2018 v3.1.1. Available online: https://astqb.org/assets/documents/CTFL-2018-Syllabus.pdf (accessed on 10 December 2023).
198. Lewis, W.E. *Software Testing and Continuous Quality Improvement*; Auerbach Publications: Boca Raton, FL, USA, 2004.
199. Martin, R.C. *Clean Code: A Handbook of Agile Software Craftsmanship*; Pearson Education: Upper Saddle River, NJ, USA, 2009.
200. Thomas, D.; Hunt, A. *The Pragmatic Programmer: Your Journey to Mastery*; Addison-Wesley Professional: Boston, MA, USA, 2019.
201. Hutter, F.; Kotthoff, L.; Vanschoren, J. *Automated Machine Learning: Methods, Systems, Challenges*; Springer: New York, NY, USA, 2019.
202. Melis, G.; Dyer, C.; Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv* **2017**, arXiv:1707.05589.
203. Snoek, J.; Larochelle, H.; Adams, R.P. Practical bayesian optimization of machine learning algorithms. In Proceedings of the 25th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; Volume 25.
204. Bergstra, J.; Yamins, D.; Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Proceedings of the International Conference on Machine Learning. PMLR, Atlanta, GA, USA, 16–21 June 2013; pp. 115–123.
205. Sculley, D.; Snoek, J.; Wiltschko, A.; Rahimi, A. Winner's curse? On pace, progress, and empirical rigor. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018), Vancouver, BC, Canada, 30 April–3 May 2018.
206. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.

207. Hansen, N. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 75–102.
208. Snoek, J.; Rippel, O.; Swersky, K.; Kiros, R.; Satish, N.; Sundaram, N.; Patwary, M.; Prabhat, M.; Adams, R. Scalable bayesian optimization using deep neural networks. In Proceedings of the International Conference on Machine Learning. PMLR, Lille, France, 6–11 July 2015; pp. 2171–2180.
209. Dahl, G.E.; Sainath, T.N.; Hinton, G.E. Improving deep neural networks for LVCSR using rectified linear units and dropout. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8609–8613.
210. Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R.P.; De Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* **2015**, *104*, 148–175. [CrossRef]
211. Brochu, E.; Cora, V.M.; De Freitas, N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv* **2010**, arXiv:1012.2599.
212. Zeng, X.; Luo, G. Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. *Health Inf. Sci. Syst.* **2017**, *5*, 2. [CrossRef]
213. Zhang, Y.; Bahadori, M.T.; Su, H.; Sun, J. FLASH: Fast Bayesian optimization for data analytic pipelines. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francsisco, CA, USA, 13–17 August 2016; pp. 2065–2074.
214. Jamieson, K.; Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In Proceedings of the Artificial Intelligence and Statistics. PMLR, Cadiz, Spain, 9–11 May 2016; pp. 240–248.
215. Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **2017**, *18*, 6765–6816.
216. Falkner, S.; Klein, A.; Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 1437–1446.
217. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 1345–1359. [CrossRef]
218. Ravi, S.; Larochelle, H. Optimization as a model for few-shot learning. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
219. Elsken, T.; Metzen, J.H.; Hutter, F. Neural architecture search: A survey. *J. Mach. Learn. Res.* **2019**, *20*, 1997–2017.
220. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8697–8710.
221. Zela, A.; Klein, A.; Falkner, S.; Hutter, F. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv* **2018**, arXiv:1807.06906.
222. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Aging evolution for image classifier architecture search. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 2, p. 2.
223. Runge, F.; Stoll, D.; Falkner, S.; Hutter, F. Learning to design RNA. *arXiv* **2018**, arXiv:1812.11951.
224. Swersky, K.; Snoek, J.; Adams, R.P. Freeze-thaw Bayesian optimization. *arXiv* **2014**, arXiv:1406.3896.
225. Domhan, T.; Springenberg, J.T.; Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015.
226. Klein, A.; Falkner, S.; Springenberg, J.T.; Hutter, F. Learning curve prediction with Bayesian neural networks. In Proceedings of the International Conference on Learning Representations, Diego, CA, USA, 7–9 May 2015.
227. Baker, B.; Gupta, O.; Raskar, R.; Naik, N. Accelerating neural architecture search using performance prediction. *arXiv* **2017**, arXiv:1705.10823.
228. Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y.L.; Tan, J.; Le, Q.V.; Kurakin, A. Large-scale evolution of image classifiers. In Proceedings of the International Conference on Machine Learning. PMLR, Sydney, Australia, 6–11 August 2017; pp. 2902–2911.
229. Elsken, T.; Metzen, J.H.; Hutter, F. Simple and efficient architecture search for convolutional neural networks. *arXiv* **2017**, arXiv:1711.04528.
230. Elsken, T.; Metzen, J.H.; Hutter, F. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv* **2018**, arXiv:1804.09081.
231. Cai, H.; Chen, T.; Zhang, W.; Yu, Y.; Wang, J. Efficient architecture search by network transformation. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
232. Cai, H.; Yang, J.; Zhang, W.; Han, S.; Yu, Y. Path-level network transformation for efficient architecture search. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 678–687.
233. Saxena, S.; Verbeek, J. Convolutional neural fabrics. In Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; Volume 29.
234. Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient neural architecture search via parameters sharing. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4095–4104.
235. Bender, G.; Kindermans, P.J.; Zoph, B.; Vasudevan, V.; Le, Q. Understanding and simplifying one-shot architecture search. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 550–559.
236. Liu, H.; Simonyan, K.; Yang, Y. Darts: Differentiable architecture search. *arXiv* **2018**, arXiv:1806.09055.

237. Cai, H.; Zhu, L.; Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv* **2018**, arXiv:1812.00332.
238. Xie, S.; Zheng, H.; Liu, C.; Lin, L. SNAS: Stochastic neural architecture search. *arXiv* **2018**, arXiv:1812.09926.
239. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. In Proceedings of the 24th International Conference on Neural Information Processing Systems, Granada, Spain, 12–15 December 2011; Volume 24.
240. Desautels, T.; Krause, A.; Burdick, J.W. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *J. Mach. Learn. Res.* **2014**, *15*, 3873–3923.
241. Ginsbourger, D.; Le Riche, R.; Carraro, L. Kriging is well-suited to parallelize optimization. In *Computational Intelligence in Expensive Optimization Problems*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 131–162.
242. Hernández-Lobato, J.M.; Requeima, J.; Pyzer-Knapp, E.O.; Aspuru-Guzik, A. Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In Proceedings of the International Conference on Machine Learning. PMLR, Sydney, Australia, 6–11 August 2017; pp. 1470–1479.
243. Hutter, F.; Hoos, H.H.; Leyton-Brown, K. Parallel algorithm configuration. In Proceedings of the Learning and Intelligent Optimization: 6th International Conference, LION 6, Paris, France, 16–20 January 2012; pp. 55–70.
244. Zhang, C.; Xie, Y.; Bai, H.; Yu, B.; Li, W.; Gao, Y. A survey on federated learning. *Knowl. Based Syst.* **2021**, *216*, 106775. [CrossRef]
245. Nagarajah, T.; Poravi, G. A review on automated machine learning (AutoML) systems. In Proceedings of the 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Bombay, India, 29–31 March 2019; pp. 1–6.
246. Thakur, A.; Krohn-Grimberghe, A. Autocompete: A framework for machine learning competition. *arXiv* **2015**, arXiv:1507.02188.
247. Ferreira, L.; Pilastri, A.; Martins, C.M.; Pires, P.M.; Cortez, P. A comparison of AutoML tools for machine learning, deep learning and XGBoost. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
248. Thornton, C.; Hutter, F.; Hoos, H.H.; Leyton-Brown, K. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA, 11–14 August 2013; pp. 847–855.
249. Kotthoff, L.; Thornton, C.; Hoos, H.H.; Hutter, F.; Leyton-Brown, K. Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA. In *Automated Machine Learning: Methods, Systems, Challenges*; Springer: Cham, Switzerland, 2019; pp. 81–95.
250. Komer, B.; Bergstra, J.; Eliasmith, C. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In Proceedings of the ICML Workshop on AutoML, Austin, TX, USA, 6–12 July 2014; Volume 9, p. 50.
251. Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; Hutter, F. Efficient and robust automated machine learning. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
252. Feurer, M.; Eggensperger, K.; Falkner, S.; Lindauer, M.; Hutter, F. Auto-Sklearn 2.0: Hands-free automl via meta-learning. *J. Mach. Learn. Res.* **2020**, *23*, 1–61.
253. Olson, R.S.; Moore, J.H. TPOT: A tree-based pipeline optimization tool for automating machine learning. In Proceedings of the Workshop on Automatic Machine Learning. PMLR, Cadiz, Spain, 9–11 May 2016; pp. 66–74.
254. Zimmer, L.; Lindauer, M.; Hutter, F. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *43*, 3079–3090. [CrossRef] [PubMed]
255. Jin, H.; Song, Q.; Hu, X. Auto-keras: An efficient neural architecture search system. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 1946–1956.
256. Peng, H.; Du, H.; Yu, H.; Li, Q.; Liao, J.; Fu, J. Cream of the Crop: Distilling Prioritized Paths For One-Shot Neural Architecture Search. In Proceedings of the 34th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 6–12 December 2020.
257. Microsoft Research. NNI Related Publications. Available online: https://nni.readthedocs.io/en/latest/notes/research_publications.html (accessed on 10 December 2023).
258. Erickson, N.; Mueller, J.; Shirkov, A.; Zhang, H.; Larroy, P.; Li, M.; Smola, A. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv* **2020**, arXiv:2003.06505.
259. Pandey, P. A Deep Dive into H2O's AutoML. Available online: https://h2o.ai/blog/2019/a-deep-dive-into-h2os-automl/ (accessed on 10 December 2023).
260. Wang, C.; Wu, Q.; Weimer, M.; Zhu, E. Flaml: A fast and lightweight automl library. *Proc. Mach. Learn. Syst.* **2021**, *3*, 434–447.
261. Shchur, O.; Turkmen, C.; Erickson, N.; Shen, H.; Shirkov, A.; Hu, T.; Wang, Y. AutoGluon-TimeSeries: AutoML for Probabilistic Time Series Forecasting. *arXiv* **2023**, arXiv:2308.05566.
262. Khider, D.; Zhu, F.; Gil, Y. autoTS: Automated machine learning for time series analysis. In Proceedings of the AGU Fall Meeting Abstracts, San Francisco, CA, USA, 9–13 December 2019; Volume 2019, p. PP43D–1637.
263. Schafer, R.W. What is a Savitzky-Golay filter? [lecture notes]. *IEEE Signal Process. Mag.* **2011**, *28*, 111–117. [CrossRef]