Mercury: Reusable and Efficient ML Workflows in Finance

Álvaro Ibrain-Rodriguez*
Daniel Sanchez-Santolaya*
Jacobo Chaquet-Ulldemolins*
Marcos Galletero-Romero*
Santiago Basaldúa*
alvaro.ibrain@bbva.com
daniel.sanchez.santolaya@bbva.com
jacobo.chaquet@bbva.com
marcos.galletero@bbva.com
santiago.basaldua@bbva.com

BBVA AI Factory
Madrid, Madrid, Spain

Abstract

Mercury is a Python library developed at BBVA, specifically designed to enhance collaboration among data scientists by enabling the efficient sharing of code, thereby addressing inefficiencies in the development and deployment of predictive models for financial applications. Featuring a modular architecture, Mercury ensures adaptability and ease of use, while significantly enhancing the development process of analytical models. The library's evolution from innersource to open-source has broadened its accessibility and fostered a global community of users and contributors. This paper discusses the main modules in Mercury, including advanced schema management, extensive model and data testing, event prediction analysis, model explainability, and mechanisms for monitoring data and model drift. Together, these components streamline the end-to-end ML model development process, providing a robust solution to the challenges of rapid deployment and scalability in a competitive environment.

Keywords

Finance, machine learning, opensource

ACM Reference Format:
Álvaro Ibrain-Rodriguez, Daniel Sanchez-Santolaya, Jacobo Chaquet-Ulldemolins, Marcos Galletero-Romero, and Santiago Basaldúa. 2018. Mercury: Reusable and Efficient ML Workflows in Finance. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (ICAIF '24).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

*All authors contributed equally to this research.

ICAIF '24, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The development of Artificial Intelligence-based products within large organizations like BBVA demands significant resources, particularly for the design, implementation, and validation of algorithms by teams spread worldwide. Commonly, these teams, despite varying use cases, find themselves developing similar solutions independently, leading to duplicated efforts and substantial resource wastage.Addressing these inefficiencies, Mercury is strategically positioned to complement the functionality of existing open source libraries, such as *scikit-learn*, rather than replicate them, it focuses on complementing these resources by providing tools and features not covered by existing solutions. It facilitates efficient sharing and utilization of code components among teams and enhances the development process by reducing redundancy while maintaining high code quality standards. This is achieved through its modular architecture, which allows teams to use only the components they need, tailored to specific project requirements. Mercury, deployed across BBVA and available to the open-source community via GitHub, aims to mitigate duplications and promotes internal efficiency, streamline development processes, and enhance the globalization and reuse of analytical products by embracing and extending open-source practices. These practices, known as Inner Source Software (ISS) [5, 11, 21], foster a culture of transparency, collaboration, and innovation within organizations—principles that are central to the *Open Source software* (OSS) movement [1, 25]. This approach not only accelerates development cycles [22] but also enhances the quality and adaptability of software solutions, making Mercury a cornerstone for developing robust, scalable machine learning workflows in finance. Mercury's open-source strategy exemplifies and amplifies these benefits, demonstrating a commitment to fostering a collaborative, innovative, and efficient software development environment.

A core contributor to Mercury's evolution is BBVA's internal innovation initiative known as the "X Program". This program is designed to rapidly prototype ideas into tangible solutions through short, focused projects. Each X project brings together small cross-functional teams for a two-to-three-month period to develop a

functional prototype that addresses shared challenges across different business areas. These prototypes, although not fully integrated or production-ready, serve as minimum viable products (MVPs), allowing other teams to adopt, test, and further evolve them. The X Program fosters an agile prototyping process by enabling rapid experimentation, early validation of ideas, and iterative testing. This approach feeds directly into Mercury's architecture, enriching the platform with new modules and features that address real-world financial challenges. By focusing on rapid prototyping and immediate functionality, these projects provide Mercury with a steady influx of innovations, ensuring that it remains a cutting-edge framework for transitioning ML models from prototype to production within financial environments.

Mercury is designed as a highly modular library, with various OSS modules available, such as mercury-dataschema for advanced schema management in data preprocessing, mercury-robust for performing robust testing on machine learning (ML) models and datasets, and others. These modules are accessible along with their source code, binaries, and full documentation at https://www.bbvaaifactory.com/mercury/, empowering developers and researchers to leverage the capabilities of Mercury. This paper describes the mercury modules released to the open source community.

2 Mercury modules

Mercury has become a major project, featuring 6 OSS modules that expose around 100 components, each offering a wide variety of functionalities across various domains. Aware of the complexity and diversity inherent in the project, the library was architecturally designed with modularity at its core. This design philosophy allows users to selectively install only the components required for their specific project needs. Consequently, Mercury is structured with multiple micro-repositories in a highly modular design, each capable of operating autonomously. This modular configuration permits a tailored and flexible integration, aligning with the unique requirements of individual users. The next subsections briefly describes the main mercury modules released to the OSS community.

2.1 Mercury-dataschema [1]: Advanced Schema Management for Data Preprocessing

Mercury-dataschema enhances data preprocessing by offering advanced schema management capabilities. This module is especially valuable for ensuring that financial data inputs are efficiently and accurately processed, a relevant factor in risk assessment and decision-making processes. The *DataSchema* class within this submodule specializes in the automatic inference of feature types from a provided Pandas DataFrame. This is not just about identifying data types; it involves interpreting the underlying characteristics of a feature to determine their appropriate categorizations. For instance, in credit risk modeling, a feature such as scoring, typically represented as a float, might be intelligently classified by Mercury-dataschema as a categorical variable, thus simplifying further risk segmentation processes. Beyond type inference, the mercury-dataschema module computes various statistics based on the inferred types, facilitating comprehensive data analysis and validation. This functionality ensures consistency across different datasets, validating whether they adhere to the same schema, or employing derived statistics to assess data drift, which can signal shifts in customer behavior or market conditions. mercury-dataschema is a core module used by other Mercury sub-modules, it also offers the flexibility to be used independently, making it a versatile tool in the pre-processing and exploration of data sets. This utility exemplifies how Mercury streamlines data management and analysis processes, enhancing the ability of the bank to develop robust financial products and adapt to new regulatory environments efficiently.

2.2 Mercury-robust [2]: Performing Robust Testing on ML Models and Datasets

Mercury-robust is a framework to perform robust testing on ML models and datasets. ML systems behavior heavily depend on data, and even the most advanced models are easily fooled by almost imperceptible perturbations of their inputs [15, 20].

Consequently, the need to test and monitor production-readiness of ML systems is becoming increasingly evident [4]. In the industrial environment, it is necessary to ensure that when a trained ML model is deployed, it behaves properly in the real world. To this end, it must be provided with a strong resilience to adverse situations and changes in its environment. In addition, it is essential to ensure that future updates to the model do not add potential points of failure in the system, which could jeopardize its reliability and performance. To address these issues, mercury-robust was born as a framework. Designed to address the multifaceted requirements of robust ML model testing, it offers a suite of analytical tools and methodologies. By incorporating robustness assessment into the model development lifecycle, mercury-robust facilitates the identification and mitigation of vulnerabilities, thereby ensuring model reliability and performance in real-world deployments. At a high level, mercury-robust is divided into two types of components: Data Test and Model Test, depending on whether they involve a model or just the data, respectively. Table 1 shows the different tests available [3]. By executing these tests when creating our datasets and training our models, we can avoid issues like introducing a feature with leaking, wasting computational resources and increasing the maintenance cost with unnecessary features, or deploying a model that performs worse in a group of our population. The framework includes the *TestSuite* class, which allows to create a battery of tests that we can re-run every time we update a model.

Mercury-robust can be applied beyond tabular data. As example, at BBVA it is applied in some text classification use cases, like classifying bank transactions or detecting the degree of urgency of a customer message. Annotating text datasets is an error-prone process and, if left unverified, is not unsurprising to find datasets with inconsistencies and inaccuracies in the labels. Based on confident learning [13], the *NoisyLabelsTest* helps to identify issues in the annotated texts and ensure that the datasets that we create to train these models exhibit the required label quality. While text classification models can reach a high accuracy, it is not infrequent to fail to detect behavioral failures such as changing the model

[1] Mercury-dataschema online documentation: https://bbva.github.io/mercury-dataschema/

[2] Mercury-robust online documentation: https://bbva.github.io/mercury-robust/

[3] Mercury-robust cheatsheet summary: https://www.bbvaaifactory.com/mercury/Mercury-robust_cheatsheet.pdf

Data Test	
SameSchemaTest	Ensures that a DataFrame exhibits identical columns and feature types as specified in the DataSchema.
DriftTest	Verifies that the distributions of individual features have not undergone significant changes between the data used for training and data used for inference.
LinearCombinationsTest	Validates the absence of redundant or unnecessary columns.
NoDuplicatesTest	Confirms the absence of duplicate samples in the dataset, as this can introduce bias into performance metrics.
SampleLeakingTest	Checks for the presence of test or validation dataset samples that are already included in the training dataset.
LabelLeakingTest	Ensures that no feature leaks information about the target variable.
NoisyLabelsTest	Validates the quality of dataset labels, identifying low-quality labels characterized by a high number of mislabeled samples or unclear label separation.
CohortPerformanceTest	Evaluates whether a specified metric performs disproportionately worse for a particular subset of the data. For example, it can be configured to assess if a accuracy of the model is lower for one gender compared to another.
Model Test	
ModelReproducibilityTest	Trains a model twice and verifies that the predictions (or a specific metric) from the two versions do not exhibit significant differences.
FeatureCheckerTest	Estimates feature importance and re-trains the model by iteratively removing the least important features.
TreeCoverageTest	Specifically designed for tree-based models, this test ensures that, given a test dataset, the samples "activate" a minimum number of branches in their respective tree(s) once a model is trained.
ModelSimplicityChecker	Compares the performance of the model against a simpler baseline (typically a linear model by default, though users can specify their own baseline).
ClassificationInvarianceTest	Evaluates whether the model remains unchanged when subjected to perturbations that should not impact the sample labels.
DriftResistanceTest	Assesses the model's resistance to drift in the data, ensuring its robustness under changing environmental conditions.

Table 1: Mercury-robust Tests

prediction when applying label-preserving perturbations. For example, our model detecting the urgency of a customer message should not change the prediction when a person's name is changed in the message. By applying the *ClassificationInvarianceTest* we check that each new version of this model is not susceptible to these behavioral flaws.

2.3 Mercury-settrie [4]: A High-Performance Implementation with Text Indexing applications

Mercury-settrie is a highly efficient C++ implementation of the settrie algorithm [18] to find subsets and supersets from a given query set within a collection of sets stored in the structure. It derives from the trie data structure [17] and both have wide applications in data mining, object-relational databases, rule-based expert systems and recommender systems. The key idea is a sorted tree of elements as shown in figure 1.

A typical application is finding the list of documents containing a specific set of words. Conceptually, each document can be viewed as a set of words and the collection itself is a collection of sets stored in a settrie structure. Given a set of words used to query,

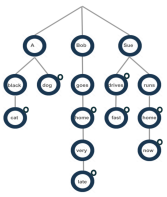

Figure 1: A settrie structure storing sets of words in a tree. The circle markers identify the end of each set. Note how the sets {Sue, runs, home} and {Sue, runs, home, now} share their common elements.

the superset of that set, which is retrieved in logarithmic time, is the list of documents. Mercury-settrie can be used in finance for regulatory compliance and regulatory reporting, where financial institutions need to quickly identify and retrieve all documents containing terms related to specific regulatory requirements or audits. For example, in BBVA Mercury-settrie is used to efficiently locate all transaction records with specific characteristics for use in different processes, facilitating faster response times to regulatory queries or generating data sets to train other models.

[4] Mercury-settrie online documentation: https://bbva.github.io/mercury-settrie/

Notably, this data structure finds utility in auto-complete applications as well.

The library has a Python 3 interface. It provides the speed and limited storage from the C++ implementation combined with a Pythonic API supporting iterators and automatic serialization. Remarkably, an off-the-shelf computer can store representations in RAM corresponding to terabytes of documents and deliver query results at a pace far exceeding typing speeds. In performance comparison with pure Python implementations, mercury-settrie demonstrates superiority by being approximately 200 times faster and 20 times more memory efficient.

2.4 Mercury-reels [5]: Analyzing Transactional Data for Event Prediction

Mercury-reels is a library to analyze sequences of events extracted from transactional data to predict when related target events may occur in the future. These events can be automatically discovered or manually defined. As show in figure 2, Mercury-reels identifies events by assigning them event codes and creates clips, which are sequences of (code, time of occurrence) tuples for each client. Using these clips, a model can be generated to predict the time at which target events may occur in the future.

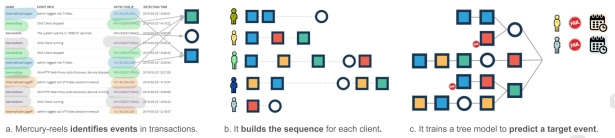

Figure 2: The main phases in a reels pipeline: a. identifying events, b. building sequences and c. training a model.

Originally conceived for analyzing web navigation transactional data, mercury-reels finds natural applications in cybersecurity and any scenario where event prediction or risk scoring based on historical data is pertinent. The definition of relevant events can be derived from transactional data or established through domain expertise. Alternatively, the Reels event optimizer facilitates a semi-automated approach for iteratively learning and refining event predictions. Implemented in C++ with a Python interface, mercury-reels boasts single-threaded efficiency, seamlessly accommodating millions of clients and billions of records with hundreds of thousands of events. For enhanced parallelization, data can be partitioned by dividing the client set into smaller subsets and operate on each subset independently. The library provides the tool, a filter by client ids, to do it easily.

Mercury-reels uses discrete events. Even in cases where continuous time series analysis is more appropriate, the prediction made by Reels can be used as a score in other methods. This leverages the sequence-of-events perspective other methods may not capture as well. The library offers flexible support for defining relevant predictive events, accommodating manual, fully automatic, or assisted approaches. Notably, Reels facilitates the prediction of target events within or outside the transactional dataset. Implemented in C++, Reels prioritizes optimized performance, while its Pythonic interface ensures seamless interoperability with serializable objects, iterators, and interfaces with popular data processing libraries like pandas and pyspark.

[5]Mercury-reels online documentation: https://bbva.github.io/mercury-reels/

2.5 Mercury-Explainability [6]: Explainability in highly regulated industries

Mercury-explainability stands as a comprehensive library with implementations of different state-of-the-art methods in the field of explainability. Particularly relevant in highly regulated sectors such as financial services, with regulations such as the General Data Protection Regulation (GDPR) in the European Union, which requires transparency in automated decisions. An application of this module can be seen in loan prediction algorithms. These algorithms, while not directly employing protected categories such as race or gender, can use variables significantly correlated with these categories, which can lead to inadvertent discriminatory practices. Mercury explainability helps to disentangle these complex relationships by providing clear and understandable explanations for the decisions of the model, thus ensuring compliance and avoiding any unintended bias. For example, in the scenario where a lending model might be indirectly discriminating based on zip codes closely correlated with ethnic demographics, Mercury-explainability can pinpoint how each characteristic influences the loan approval process, thus allowing the institution to adjust the model to eliminate unfair bias. This capability is crucial not only to meet regulatory requirements, but also to maintain the confidence of clients and regulators. It ensures that all automated decisions are transparent and defensible, allowing for a thorough understanding and auditing of the loan approval process. Extensive works [3, 6, 7] has explored this imperative, leading to the compilation of numerous state-of-the-art interpretability algorithms within Mercury. They are designed to work efficiently and to be easily integrated with the main ML frameworks. The basic block of mercury-explainability is the ***Explainer*** class. Each explainer offers a unique approach to model interpretation, often yielding an Explanation object encapsulating the results.

[6]Mercury-explainability online documentation: https://bbva.github.io/mercury-explainability/

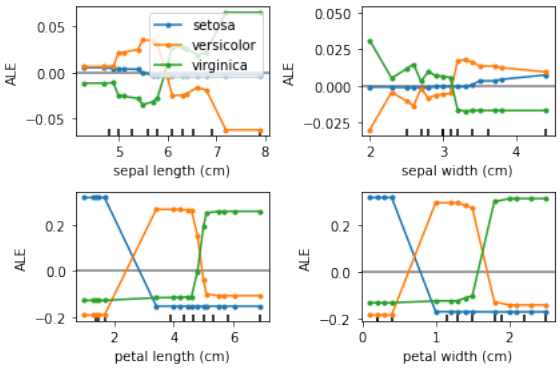

Figure 3: Accumulated Local Effects.

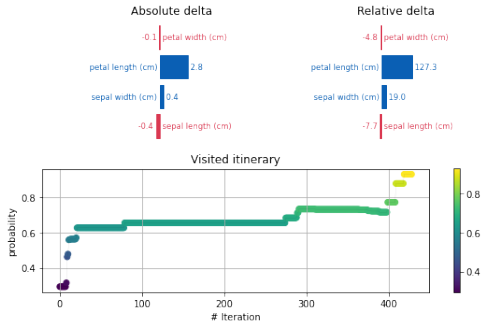

Figure 4: Counterfactual Explanations.

- Accumulated Local Effects (ALE)[2, 12]: show how model inputs affect the prediction on average. ALE Plots tend to be more useful in cases where there are correlations between different model inputs. For example, if we have a model that predicts the probability of default when granting credit, it can help us understand that our model tends to decrease the probability of default when the model input "monthly income" increases. In Figure 3 we present plots that illustrate the impact of various input features on model predictions across different classes. These visualizations clearly demonstrate how changes in feature values influence the outcome.
- Counterfactual Explanations (CE): This method looks for the necessary changes in inputs to achieve a predefined model output, rather than the actual prediction [8, 23]. In the context of a loan impact prediction model, Counterfactual Explanations can elucidate the adjustments needed in input features to reduce the probability of default. Within the mercury-explainer framework, two counterfactual methods, namely *CounterFactualExplainerBasic* and *CounterfactualProtoExplainer*, provide avenues for exploring these alternate scenarios. In Figure 4 is presented an illustrative example of a counterfactual explanation generated by Mercury-interpretability. The first two charts detail the adjustments needed in each feature to achieve the desired outcome, illustrating the absolute and relative changes necessary for reaching a specific decision threshold. These adjustments indicate how much each feature's value should be increased or decreased. The final plot tracks the evolution of the probability of achieving the target outcome across successive iterations, showcasing the trajectory of the decision-making process as adjustments are made to the model inputs.
- ClusteringTreeExplainer (CTE): Leveraging DecisionTree models, this explainer elucidates the behavior of clustering models, offering valuable insights into the underlying structure of the data. This method is based on the papers Iterative Mistake Minimization (IMM) [9] and ExKMC [10]. By constructing decision trees aligned with clustering results, this method facilitates the interpretation of cluster groupings within the data, contributing to enhanced model understanding.
- Anchors Explanations (AE): offers insights into model predictions by identifying rules in input features that consistently lead to specific outcomes [16]. These rules highlight subsets of inputs where model predictions remain unchanged, irrespective of variations in other inputs, thus enhancing interpretability and trust in the model's decisions.

With mercury-explainability, users can navigate the intricacies of model decision-making processes, fostering transparency and trust in ML systems.

deviation

2.6 Mercury-monitoring [7]: data drift and model degradation detection

Mercury-monitoring is a library designed to monitor data and model drift, which are critical for the correct deployment of ML systems in production environments. The performance of ML models relies on the consistency of their input data. When working in these environments, these models may experience performance degradation over time due to abrupt or gradual shifts in input data distribution. Classical software systems trigger alerts upon encountering unexpected inputs. In contrast, ML models often fail without any indicative warning unless equipped with appropriate monitoring tools to detect such drifts [14]. These drifts can occur for various reasons, including changes in the operational environment, such as economic fluctuations. Other factors include the introduction of new competitor products and the emergence of novel fraud techniques. Changes in input units can also cause drifts. Seasonal phenomena, such as a sudden surge in inquiries about tax declarations during relevant months, can impact text classifiers aimed at identifying customer message themes.

At BBVA, we include monitoring in critical processes such as early debt recovery, whose goal is to prevent and assist the client in overcoming situations in which they have fallen behind their payments. Monitoring our debt recovery models helps us adjust the models to current economic realities. These methods often rely on capturing the statistical properties of metrics, input data or output data. Nonetheless, in problems such as regression in loan repayment risk, the accuracy of model predictions remains uncertain until the loans are repaid, thus rendering those mentioned methods incapable of monitoring these processes. For those

[7] Mercury-monitoring online documentation: https://bbva.github.io/mercury-monitoring/

scenarios, Mercury-Monitoring incorporates a component dedicated to predicting model performance in the absence of outputs. This predictive methodology, as outlined in [19], diverges from traditional approaches by not requiring explicit assumptions about distributional changes between source and service data. Instead, it utilizes common variations and errors within datasets to train a performance predictor for the model. This system is designed to autonomously trigger alerts upon detecting performance drops in unseen service data, similar to those observed with the introduction of synthetic errors. While numerous resources [24] provide extensive descriptions of various types of data drift, in general, we can distinguish between:

(1) Concept drift: Changes over time on the statistical properties between input features and the target variable, that cause model predictions to become less accurate.
(2) Feature drift: Variations in the distribution of the model's input variables over time compared to the data used during training.
(3) Label drift: Error drift that manifests when the distribution of input variables remains constant, but the target variable undergoes changes.

For those methods that detect changes in input and output data, Mercury-Monitoring incorporates algorithms grouped into two different categories. First, we can find components that address the issue by looking separately at each feature in a dataset using statistical methods. We can use the *KSDrift*, *Chi2Drift* and *HistogramDistanceDrift* components in this category. Alternatively, we can use components that focus on the joint distribution of the features. Here we find the *DomainClassifierDrift*, *AutoencoderDriftDetector*, and the *DensityDriftDetector* components, which are all model-based detectors.

3 Conclusions

In this paper, we introduced Mercury, a Python library developed by BBVA aimed at fostering collaboration among data scientists through efficient code sharing. Mercury offers a wide range of state-of-the-art algorithms and utilities, empowering data scientists to accelerate the creation of analytical models across various domains. One of the key advantages of Mercury lies in its modular design, allowing users to selectively install only the components they require, thereby enhancing reusability and scalability. By using all the modules in Mercury, data scientists can reduce the time-to-value and time-to-market of their analytical solutions. This acceleration enables the swift innovation and deployment of ML models. A critical enabler of this acceleration is Mercury's integration with the agile prototyping processes from the X Programs. By supporting rapid prototyping, Mercury allows teams to quickly experiment with new ideas, validate them in early stages, and bring them into production. This iterative process fosters innovation by reducing development cycles and facilitating early feedback, while minimizing risks associated with large-scale deployments. Through prototyping, teams can create MVPs that are functional enough for testing and adoption by other teams, accelerating the transition from idea to operational model in real-world financial applications. Collectively, the integrated components of Mercury enhance the entire ML model development lifecycle. This integration offers a solid foundation to overcome the common challenges of rapid deployment and scalability, ensuring competitive advantage in dynamic market environments. Furthermore, the transition of Mercury from an ISS model to an OSS framework marks a milestone. By embracing open-source practices, Mercury not only fosters collaboration and knowledge sharing within BBVA but also invites external contributions and feedback from the wider developer community. This openness facilitates continuous improvement and innovation, driving Mercury towards becoming a more robust and versatile ML systems. Looking ahead, Mercury continues to evolve and new lines of development are being explored in areas such as embedding techniques, synthetic data generation and graph analysis. These additions are intended to further enrich the capabilities of the library.

In conclusion, Mercury simplifies development and deployment of ML models for enterprises, offering flexibility, efficiency, and openness. Its modular architecture, coupled with its integration of state-of-the-art algorithms, promises to revolutionize the way data scientists approach analytical challenges, ultimately leading to greater productivity and innovation in the field of artificial intelligence.

Acknowledgments

References

[1] Mark Aberdour. 2007. Achieving Quality in Open-Source Software. *IEEE Software* 24, 1 (2007), 58–64. https://doi.org/10.1109/MS.2007.2
[2] Daniel W. Apley and Jingyu Zhu. 2019. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. arXiv:1612.08468 [stat.ME]
[3] Philippe Bracke, Anupam Datta, Carsten Jung, and Shayak Sen. 2019. Machine learning explainability in finance: an application to default risk analysis. (2019).
[4] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. 2017. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. In *Proceedings of IEEE Big Data*.
[5] Maximilian Capraro and Dirk Riehle. 2016. Inner Source Definition, Benefits, and Challenges. *ACM Comput. Surv.* 49, 4, Article 67 (dec 2016), 36 pages. https://doi.org/10.1145/2856821
[6] Jacobo Chaquet-Ulldemolins, Francisco-Javier Gimeno-Blanes, Santiago Moral-Rubio, Sergio Muñoz-Romero, and José-Luis Rojo-Álvarez. 2022. On the Black-Box Challenge for Fraud Detection Using Machine Learning (I): Linear Models and Informative Feature Selection. *Applied Sciences* 12, 7 (2022). https://doi.org/10.3390/app12073328
[7] Jacobo Chaquet-Ulldemolins, Francisco-Javier Gimeno-Blanes, Santiago Moral-Rubio, Sergio Muñoz-Romero, and José-Luis Rojo-Álvarez. 2022. On the Black-Box Challenge for Fraud Detection Using Machine Learning (II): Nonlinear Analysis through Interpretable Autoencoders. *Applied Sciences* 12, 8 (2022). https://doi.org/10.3390/app12083856
[8] Susanne Dandl, Christoph Molnar, Martin Binder, and Bernd Bischl. 2020. *Multi-Objective Counterfactual Explanations*. Springer International Publishing, 448–469. https://doi.org/10.1007/978-3-030-58112-1_31
[9] Sanjoy Dasgupta, Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. 2020. Explainable k-means and k-medians clustering. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 654, 11 pages.
[10] Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. 2020. ExKMC: Expanding Explainable k-Means Clustering. arXiv:2006.02399 [cs.LG]
[11] InnerSource Commons Community. 2020. *InnerSource Patterns*. InnerSource Commons. https://innersourcecommons.org/learn/books/innersource-patterns/.
[12] Christoph Molnar, Gunnar König, Julia Herbinger, Timo Freiesleben, Susanne Dandl, Christian A. Scholbeck, Giuseppe Casalicchio, Moritz Grosse-Wentrup, and Bernd Bischl. 2021. General Pitfalls of Model-Agnostic Interpretation Methods for Machine Learning Models. arXiv:2007.04131 [stat.ML]
[13] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. 2019. Confident Learning: Estimating Uncertainty in Dataset Labels. *J. Artif. Intell. Res.* 70 (2019), 1373–1411. https://api.semanticscholar.org/CorpusID:207870256
[14] Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. 2019. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. arXiv:1810.11953 [stat.ML]

[15] Jonas Rauber, Wieland Brendel, and Matthias Bethge. 2018. Foolbox: A Python toolbox to benchmark the robustness of machine learning models. arXiv:1707.04131 [cs.LG]

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://doi.org/10.1609/aaai.v32i1.11491

[17] Ronald L. Rivest. 1976. Partial-Match Retrieval Algorithms. *SIAM J. Comput.* 5, 1 (1976), 19–50. https://doi.org/10.1137/0205003 arXiv:https://doi.org/10.1137/0205003

[18] Iztok Savnik. 2013. Index Data Structure for Fast Subset and Superset Queries. In *Availability, Reliability, and Security in Information Systems and HCI*, Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 134–148.

[19] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2020. Learning to Validate the Predictions of Black Box Classifiers on Unseen Data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1289–1299. https://doi.org/10.1145/3318464.3380604

[20] Vikash Sehwag, Arjun Nitin Bhagoji, Liwei Song, Chawin Sitawarin, Daniel Cullina, Mung Chiang, and Prateek Mittal. 2019. Analyzing the Robustness of Open-World Machine Learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security* (London, United Kingdom) *(AISec'19)*. Association for Computing Machinery, New York, NY, USA, 105–116. https://doi.org/10.1145/3338501.3357372

[21] Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. 2014. Key factors for adopting inner source. *ACM Trans. Softw. Eng. Methodol.* 23, 2, Article 18 (apr 2014), 35 pages. https://doi.org/10.1145/2533685

[22] Klaas-Jan Stol and Brian Fitzgerald. 2015. Inner Source–Adopting Open Source Development Practices in Organizations: A Tutorial. *IEEE Software* 32 (07 2015). https://doi.org/10.1109/MS.2014.77

[23] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2018. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. arXiv:1711.00399 [cs.AI]

[24] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (April 2016), 964–994. https://doi.org/10.1007/s10618-015-0448-4

[25] Ming-Wei Wu and Ying-Dar Lin. 2001. Open source software development: an overview. *Computer* 34, 6 (2001), 33–38. https://doi.org/10.1109/2.928619