TU WIEN Informatics

Label Generation for Time Series Data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Johannes Windischbauer, BSc
Matrikelnummer 01552500

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dipl.-Ing. Dr.sc. Jürgen Cito, BSc

Wien, 2. April 2024

Johannes Windischbauer	Jürgen Cito

TU WIEN Informatics

Label Generation for Time Series Data

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Johannes Windischbauer, BSc
Registration Number 01552500

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Ing. Dr.sc. Jürgen Cito, BSc

Vienna, 2nd April, 2024

Johannes Windischbauer	Jürgen Cito

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Johannes Windischbauer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. April 2024

Johannes Windischbauer

Acknowledgements

First, I extend my sincere thanks to my advisor, Jürgen Cito, for his invaluable trust and input throughout this journey.

My gratitude also goes to Dynatrace for providing the opportunity to work on this thesis while being part of their Data Science team, and for their active participation in my research. Special thanks to Magda, Berni, and Thomas for their insightful feedback and support.

Last, but not least, I am deeply grateful to my family and friends for their support and encouragement.

Kurzfassung

Machine Learning (ML) hat sich zu einem zentralen Punkt in Forschung und Industrie entwickelt und revolutioniert wie Daten in verschiedenen Sektoren analysiert und genutzt werden. Ein Herzstück der Effektivität von ML sind beschriftete Daten, auf deren Grundlage diese Algorithmen lernen und Vorhersagen treffen. Da ML-Modelle immer komplexer und umfangreicher werden, steigt auch ihr Bedarf an beschrifteten Trainingsdaten, was eine große Herausforderung darstellt. Die Beschaffung von großen Mengen an beschrifteter Daten stellt sich als schwierig heraus. Zeitreihendaten sind aufgrund ihres dynamischen Charakters eine besondere Herausforderung - Zeitreihen wachsen und entwickeln sich durch gleitende Fenster. Weiters können die Daten Änderungen der Beschriftungen, der Beschriftungsregeln und sogar der subjektiven Interpretation dessen, was diese Beschriftungen darstellen sollen, unterliegen. Darüber hinaus ist die Aufgabe der Beschriftung selbst ein kostspieliges Unterfangen, vor allem aufgrund des hohen Zeit- und Ressourcenaufwands. Aufgrund dessen, sind in diesem Bereich neue Methoden entstanden, die darauf abzielen, diese Herausforderungen zu bewältigen. Ein solcher vielversprechender Ansatz ist die Anwendung von Weak Supervision in Verbindung mit Endmodellen. In dieser Arbeit werden ein Beschriftungsworkflow und eine Demoanwendung entwickelt, um die Leistung der Endmodelle und ihre Beschriftungsgenauigkeit zu testen und mit traditionellen manuellen Beschriftungspraktiken und einem Supervised Ansatz als Benchmark zu vergleichen. Vorläufige Ergebnisse deuten auf den allgemeinen Nutzen dieser fortgeschrittenen Techniken hin, obwohl die spezifischen Ergebnisse nicht verallgemeinerbar sind. Nichtsdestotrotz unterstreicht der positive Ausblick der Studienteilnehmer hinsichtlich der Verwendung dieser Tools für die Kennzeichnung von Zeitreihendaten das wachsende Vertrauen in alternative Kennzeichnungsstrategien und deutet auf eine potenzielle Veränderung der Art und Weise hin, wie Daten für ML-Anwendungen vorbereitet werden.

Abstract

Machine learning (ML) has become a cornerstone in research and industry, revolutionizing how data is analyzed and utilized across various sectors. At the heart of ML's effectiveness lies the crucial need for accurately labeled data, the foundation upon which these algorithms learn and make predictions. As ML models grow in complexity and size, their thirst for vast amounts of data intensifies, presenting a significant challenge. Obtaining accurately labeled data, especially for time series data, is difficult. Time series data is particularly challenging due to its dynamic nature — time series continuously expand, and evolve, labels and labeling rules undergo changes, and even the subjective interpretation of what these labels should represent changes over time. Moreover, the task of labeling itself is a costly endeavor, primarily due to the extensive amount of time and resources required. The field has seen the emergence of new methodologies aimed at mitigating these challenges, acknowledging the importance of labeling. One such promising approach is the application of weak supervision in conjunction with end models. In this work, a labeling workflow and demo application are developed to enable testing and comparing the performance of end models and their labeling accuracy against traditional manual labeling practices and supervised benchmarks. Preliminary results indicate the general feasibility of these advanced techniques, although the specific results are not universally generalizable. Nevertheless, study participants' positive outlook regarding using these tools for time series data labeling highlights growing confidence in alternative labeling strategies, suggesting a potential shift in how data is prepared for ML applications.

Contents

CHAPTER 1

Introduction

Many data science applications, particularly in research and industrial sectors, are increasingly reliant on Machine Learning (ML) models. The efficacy and precision of these models are heavily dependent on the availability and quality of training data. In supervised ML, the significance of well-labeled training data is paramount, as these models learn from input-output mappings provided during training. The labeled data is crucial for the model to understand the relationship between input variables and their corresponding output, enabling it to make accurate predictions when faced with new, unseen data.

The task of acquiring accurately labeled training data is challenging in terms of time and cost, especially in fields that require specialized knowledge for accurate labeling, such as medical imaging or linguistic analysis [1, 2].

In the context of load forecasting and anomaly detection in distributed software systems, the importance of accurately labeled time series data becomes even more pronounced. These applications have unique requirements:

1. **Load Forecasting**: In distributed systems, load forecasting involves predicting future resource usage and demand based on historical data. Accurate predictions enable efficient resource allocation and system scalability. Labeled time series data in this context aids the decision process of which forecasting model to apply, allowing for more accurate, reliable, and resource-efficient forecasting.

2. **Anomaly Detection**: Anomaly detection in distributed software systems is crucial for identifying unusual patterns that may indicate a problem, such as a potential security breach or system failure. Labeled training data is essential for teaching the model what constitutes normal and anomalous behavior. However, what constitutes an anomaly is highly context-dependent and therefore requires flexibility in the

labeled dataset. Without accurately labeled data, the model may fail to recognize critical anomalies, leading to delayed or inappropriate responses to system issues.

The process of labeling data in these applications faces several challenges:

1. **Dynamic Data**: New time series data continually added to the system must be labeled manually, leading to a constant need for data annotation. Additional data points in old data might also make relabeling necessary.

2. **Rule Changes**: Adaptations to labeling rules or criteria require existing data to be relabeled, which can be time-consuming and may lead to inconsistencies.

3. **Evolving Labels**: Introduction or modification of labels necessitates updating the training data with the changed labels to ensure that the model is trained on current data.

4. **Subjectivity in Labeling**: Differing interpretations of labeling criteria by data scientists can lead to inconsistent labeling, introducing bias and errors in the model.

The importance of labeled training data in supervised ML models, especially for load forecasting and anomaly detection in distributed software systems, is fundamental to their success. These models are trained to identify patterns and make predictions based on their training data. Without a robust, accurately labeled dataset, their ability to learn and make precise predictions is significantly compromised. This is particularly critical in applications like load forecasting and anomaly detection, where inaccurate predictions or failure to detect anomalies can have severe implications, affecting system performance, security, and reliability.

Thus, obtaining high-quality labeled data is not merely a preliminary step but a crucial, ongoing task that directly impacts the effectiveness and trustworthiness of ML models in these advanced applications. The dynamic nature of datasets and the need for accurate, scalable, and efficient labeling processes and tools are especially highlighted in the fields of load forecasting and anomaly detection in distributed software systems.

1.1 Motivation

Automatic label generation emerges as a transformative solution in data science, offering to enhance uniformity and accuracy in the labeling process while simultaneously reducing the time and effort required for labeling. The implementation of this particular approach proves beneficial, specifically in the case of handling large datasets where manual labeling is not only impractical but also susceptible to inconsistencies and errors attributable to human involvement [3].

In recent years, various methodologies have been developed for labeling textual and image data, prominently featuring the concept of weak supervision [4, 5, 6, 7], auto-encoders [8, 9], and self-supervised learning [10, 11, 12, 13]. Weak supervision learning represents a paradigm shift in how machine learning models are trained compared to supervised learning. It utilizes imperfect or limited supervision signals, making it a fitting method to aid in the automated label-creation process. This approach recognizes that while the supervision signals may not be perfect, they are adequate to guide the model towards meaningful learning. This is especially useful when dealing with large datasets where obtaining perfectly labeled data is impractical or too costly.

The goal is to adapt these principles to the realm of time series data. Time series data poses unique challenges in labeling. This work aims to let users define and apply heuristics, the so-called labeling rules or labeling functions, to sets of time series data. This allows them to programmatically generate a collection of labels without the need to individually fit complex and costly statistical models to each time series. By leveraging these heuristics, large volumes of time series data can be efficiently labeled, a task that would be unfeasible with traditional labeling methods.

An innovative approach is proposed by chaining already existing tools and providing a user interface to reduce the costs and efforts associated with labeling time series data. This approach empowers data scientists and domain experts to engage in an iterative process of exploring and labeling time series data, by creating, applying, and refining labeling rules, and by learning from these noisy heuristics. These resulting data sets are larger and more comprehensive than previously hand-labeled datasets which is crucial for training more accurate and effective supervised machine learning models.

1.2 Research Questions

The aim of this thesis is to evaluate the feasibility of an ML supported labeling workflow. During this thesis, we will explore the following research questions (RQs):

1. How effective are simple heuristics and weak supervision in labeling time series data compared to manual labeling? How effective is this approach compared to a supervised approach with less data?
2. How can the labeling workflow be designed for time series data? How can users be aided in the process of writing labeling functions?
3. Which classification models are superior in generalizing beyond weakly labeled time series data and what are the potential trade-offs?

1.3 Methodological Approach

We will utilize a methodological approach based on the widely recognized CRISP-DM process model, by Wirth and Hipp [14] in order to address the previously presented

research questions. The approach will entail a comprehensive review of the latest literature, tools, and applications related to data labeling prior to assessing the suitability of tools for weak supervision labeling use cases. We will prepare a collection of time series data for testing purposes. A labeling application will be designed and developed to enable testers to demonstrate the application and generate data. The data generated from the labeling process will be used to scrutinize the modeling performance and applied models, and to gain valuable insights.

1.4 Structure

The thesis is structured in the following way. Chapter 2 provides a comprehensive background about machine learning, performance metrics, and weak supervision, before introducing the classification models used in the application. Furthermore, the necessities for data preparation and the fundamentals of data labeling are introduced. Chapter 3 presents the latest research on data labeling approaches, before focusing on already available tools and the distinction of this work. Chapter 4 outlines our approach and the implementation. The methodology and the results can be found in Chapter 5. Chapter 6 follows up with a discussion of the results based on the posed research question and takes threats to validity into consideration. Chapter 7 concludes this thesis. Supplemental material can be found in Chapter 8.

CHAPTER 2

Background

2.1 Machine Learning

A brief overview of the historical inception of machine learning will be presented, as outlined by Zhi-Hua Zhou [15], before delving deeper into the relevant concepts for this thesis. The concept of machine learning was first mentioned by A. Turing in his 1950 paper [16]. This period also saw the development of A. Samuel's computer checkers program and the early stages of neural-network-based connectionism learning, exemplified by F. Rosenblatt's Perceptron [17] and B. Widrow's Adaline [18]. Coinciding with the broader field of artificial intelligence, the focus during this era was on logical reasoning. A seminal contribution was the Logic Theorist program by A. Newell and H. Simon [19], which successfully proved theorems from "Principia Mathematica" [20]. This period was characterized by the belief that machine intelligence could be achieved through logical reasoning.

The period of the years from 1950 to 1970 witnessed the growth of logic-representation-based symbolism learning and decision-theory-based learning. It was also during this time that the foundations of statistical learning theory were laid. The mid-1970s marked a shift in focus towards knowledge acquisition, led by researchers like E. A. Feigenbaum. This era saw the creation of successful expert systems in various fields. However, the challenge of converting knowledge into a format suitable for machine learning, known as "Feigenbaum's knowledge acquisition bottleneck," became apparent [21].

By the 1980s, machine learning had established itself as a distinct field within computer science. This decade was marked by key events such as the First International Workshop on Machine Learning [1] and the publication of "Machine Learning: An Artificial Intelligence Approach" [22]. Between the 80s and the mid-90s decision trees, neural networks and

[1] http://www.machinelearning.org/icml.html

		Actual Class	
		P	**N**
Predicted Class	**P**	**TP**	FP
	N	FN	**TN**

Table 2.1: Confusion Matrix

support vector machines gained prominence until culminating in today's big data era with the emergence of deep learning.

In the present era, machine learning has emerged as a promising solution to tackle numerous complex problems and enhance various applications. The three fundamental problems that serve as the building blocks of machine learning are: Classification, Regression, and Clustering. These problems are so versatile that most of the real-world problems can be derived from one of these three categories or a combination of them.

The task of label generation can be categorized as a classification task. Having established the problem category, it is now pertinent to consider the approach to measuring its performance.

Performance Measures

Machine learning models are evaluated based on various performance measures [23].

These measures are derived from the confusion matrix, as illustrated in Table 2.1, which is a 2x2 matrix that represents the classification results in the case of binary classification. The matrix consists of four elements: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). TP represents all correctly classified positive instances, while TN represents all correctly classified negative instances. FP represents all falsely classified positive instances, while FN represents all falsely classified negative instances [24].

Hence, the confusion matrix allows for a quick overview of the amount of type I errors (FP) and type II errors (FN) and is, moreover, a useful tool for calculating the model's performance metrics [25].

There are several commonly used measures that can be derived directly from the confusion matrix, including accuracy, precision, recall, error rate, f1-score, and many more. Therefore, the confusion matrix is an important tool for evaluating machine learning models.

When it comes to classification problems, accuracy, and error rate are the most commonly used performance measures. However, to gain a better understanding of the model's performance, independent of the dataset, a decision was made to use precision, recall, and the F1-score. The accuracy and error rate, while often useful, can be misleading in unbalanced datasets.

Accuracy is the proportion of correctly classified samples. In a binary classification problem, this is defined as follows in Equation 2.1:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

while in multiclass classification it is simply defined as depicted in Equation 2.2.

$$Accuracy = \frac{correct\ predictions}{all\ predictions} \tag{2.2}$$

We have opted to omit the error since the error rate is the inverse of the accuracy, as illustrated in Equation 2.3. Typically, these values are given in percentages.

$$Err = 1 - Accuracy \tag{2.3}$$

Precision is the amount of rightly identified positive instances divided by all positively classified ones. It is therefore given as the following fraction, in Equation 2.4,:

$$Precision = \frac{TP}{TP + FP}\ . \tag{2.4}$$

Precision is a necessary measure because accuracy does not paint the whole picture. Especially, in unbalanced datasets it can happen that there is a high accuracy as it is only predicting one class. This, however, does not mean that the predictions are right. Therefore, precision offers another dimension.

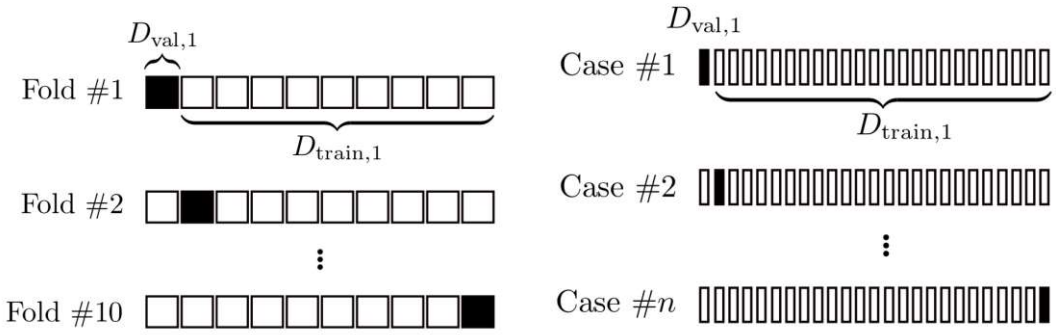
Recall on the other hand measures how many of the positive instances have been identified correctly and is illustrated by Equation 2.5.

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

The final performance metric we are using is the so-called **F1-score**, given in Equation 2.6. The F1-score is the harmonic mean of precision and recall and will thus provide a more balanced picture of the model performance.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.6}$$

After defining the measures, the next step is to determine where and how to apply them. This leads us to the crucial aspect of cross-validation, which involves validating the performance of the measures across different datasets to ensure their accuracy, precision, and reliability.



(a) K-Fold Cross Validation (b) Leave-One-Out Cross-Validation

Figure 2.1: Cross Validation Methods Visualized [26]

Cross-Validation

Cross-validation is a crucial technique in machine learning used to evaluate a model's performance on unseen data while minimizing the risk of overfitting. The technique is widely used in various fields, including machine learning, to assess a model's generalization ability. The process involves dividing a sample of data into complementary subsets, where one subset is used for training, and the other subset is used for validation [26, 27]. There exist diverse methods of cross-validation, however, we will focus on examining the following three techniques. It is imperative to note that each method is unique in its own way and is used to validate and refine the performance of predictive models. Hence, understanding the differences between them is crucial when deciding on the most suitable technique to use for a given dataset [28, 29].

K-fold cross-validation is a popular variation where the data set is divided into K equal-sized subsets, and the analysis is repeated K times, with each of the K subsets used once as the validation data. This method helps to ensure that every data point is used for both training and validation. A visual representation can be found in Figure 2.1a. This is an efficient method for small and large datasets, can, however, cause high variability in estimates, by being sensitive to the choice of folds. Increasing K increases the computational workload until K equals the sample set size and we are essentially performing the next type of cross-validation introduced below.

Leave-One-Out Cross-Validation (LOOCV) is another variation of cross-validation displayed in Figure 2.1b, where the model is trained on all the data except one point and tested on that single point. This process is repeated for each data point in the dataset, making it an exhaustive and expensive approach. It is especially useful for small datasets because there is a minimal amount of bias and causes high computational costs for large datasets.

The **Stratified Cross-Validation** is a modification of K-fold cross-validation that ensures that each fold of the dataset has the same proportion of observations with a given

categorical label. This technique is especially beneficial in situations where the dataset is imbalanced with unequal representation of categorical labels [30] and improves the model assessment. The major drawback of this method is the increased implementation complexity.

After the completion of each fold in a cross-validation process, the performance measures are computed and recorded. These measures are then stored for later analysis and comparison. Finally, to get an overall estimate of the model's performance, the recorded measures are averaged across all the folds.

Cross-validation is a fundamental technique in machine learning that aids in the evaluation and selection of models. While it does have its limitations, such as computational expense and variability, its ability to provide a more accurate assessment of a model's predictive performance is well worth it. The choice of a particular cross-validation approach is influenced by the unique requirements of the problem at hand, as well as the characteristics of the data.

2.1.1 Learning Types

The two most common machine learning types are supervised and unsupervised learning. These types cover the widest array of machine learning topics and problems. However, label generation for time series data, and in general, lies somewhere in between these types, hence there is also the category of weakly- or semi-supervised learning that combines the two types to solve a specific subset of problems [31].

Supervised Learning

In supervised learning an algorithm receives a set of examples with known labels as training data. The algorithm then uses this data to make predictions for new, unseen data points. This type of learning is commonly used for classification, regression, and ranking problems. For instance, in the case of the often cited spam detection, the algorithm is trained with a set of examples labeled as spam or not spam, and then it can classify new emails, comments, or other textual content as either spam or not spam based on what it learned during training.

Unsupervised Learning

In the case of unsupervised learning, the learner exclusively receives unlabeled training data and makes predictions for all unseen points. It is most commonly used for clustering problems to recognize patterns in complex data. This also makes evaluating the performance of the learner difficult.

Weakly Supervised Learning

Now, weakly supervised learning, also called semi-supervised learning is situated in between. There are two possible scenarios where the application of weak supervision

can lead to better results. Either there is data available, but not all data points are labeled or there is data available with imprecise labeled data which makes training and generalization more difficult. Therefore, semi-supervised learning is commonly applied in settings where unlabeled data is easily accessible but labels are expensive to obtain, which is the case for labeling time series data. Various other types of problems arising in applications, including classification, regression, or ranking tasks, can also be framed as instances of semi-supervised learning which is why large systems commonly use some form of it already [32]. The evidence also points in the direction that the error rate scales in the same way as for labeled data [6].

There are, of course, more types of machine learning including reinforcement learning, online learning, and active learning. However, as mentioned above, labeling time series data fits perfectly into the category of weak supervision tasks.

2.1.2 Models for Classification Problems

After having determined that weak supervision is the appropriate tool for the issue, we will explore the potential models and their functionalities. A table outlining the pros and cons as well as data set sizes can be found at the end of the section in Table 2.3.

The problem can be formally described in the following form, assuming a binary classification problem for the sake of keeping the length of the section at an appropriate level.

In classification problems, we have a set of input features called X and want to map these to so-called labels or classes y. The mapping is achieved by applying a function f to X to receive the labels y. We are trying to find and tune f so that it resembles an unknown relation R as closely as possible. This is achieved by adapting the model parameters so that the difference between the input labels and the model predictions is minimized.

Formally, let $X_i = (x_{1,i}; x_{2,i}; ...; x_{d,i})$ be a data point in a set X with N points where $0 < i \leq N$. $(x_{1,i}; x_{2,i}; ...; x_{d,i})$ with $d > 0$ are called independent variables or features of a certain data point X_i. There is no limitation on the type of the features.

Let $y = (y_1, y_2, ..., y_N)$ be the output for the same N points, $0 < i \leq N$, also called dependent variables, these are, in the case of binary classification, either 0 or 1. More generally, the classification element is given as element y of a finite set S and $|S| = 2$ in binary classification.

The mathematical foundation of the mapping is given in Equation 2.7.

$$R(X_i) = y_i \rightarrow f(X_i) \approx y_i \tag{2.7}$$

A representation of the resulting feature matrix resulting from this definition is shown in table 2.2.

A summary of the following models can be found at the end of Section 2.1.2 in Table 2.3.

	$x_{1,i}$	$x_{2,i}$	...	$x_{d,i}$	y_i
X_1	$x_{1,1}$	$x_{2,1}$	...	$x_{d,1}$	y_1
X_2	$x_{1,2}$	$x_{2,2}$	...	$x_{d,2}$	y_2
...	...	...	...	...	...
X_N	$x_{1,N}$	$x_{2,N}$	...	$x_{d,N}$	y_N

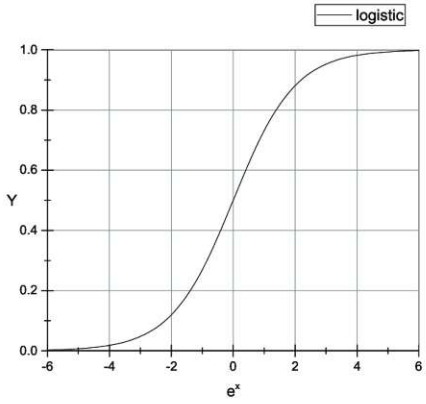
Table 2.2: Sample feature matrix including the y label vector per feature.

Logistic Regression

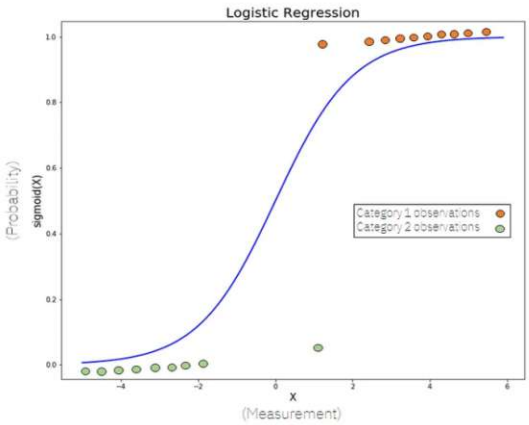
Logistic regression is a statistical technique that is commonly used to analyze a dataset with one or more independent variables that determine the outcome of a dependent variable. The outcome is a dichotomous variable, which means that there are only two possible values. Therefore, this model is extensively used for binary classification in various fields, such as machine learning, medical sciences, and social sciences, hence it is also applicable for the use in the project.

The model not only provides a binary output for classification but also generates a probability score for the outcome. It is often considered a good baseline model for classification tasks because of its simplicity and ease of interpretation, and its ability to handle large feature sets efficiently. However, there are some limitations. Logistic regression assumes that there is a linear relationship between the independent and dependent variables, which means that it might not perform well in scenarios where the relationship is not linear. Additionally, it may not work well with unbalanced datasets.

We will now briefly introduce the mathematical foundation of logistic regression based on the book written by B. Hosmer and S. Lemeshow [33].



(a) Logistic function curve: This is a standard logistic function, and it has a typical "S" shape (sigmoid curve) [34].



(b) Logistic function fitted to data[2].

Figure 2.2: Logistic functions visualized.

The logistic function in Figure 2.2a, given by Equation 2.8,

$$y = \frac{1}{1 + e^{-z}} , \tag{2.8}$$

is the base for the logistic regression, where e is the Euler's number and therefore the base of the natural logarithm, and z is an input to the function.

The odds of an outcome are then modeled by the probability as shown in Equation 2.9.

$$odds = \frac{P(y = 1|x)}{1 - P(y = 1|x)} \tag{2.9}$$

To linearize the relationship between the independent variables and the probability, we take the logarithm of the odds, which is modeled by Equation 2.10.

$$logit(P) = log(\frac{P(y = 1|x)}{1 - P(y = 1|x)}) \tag{2.10}$$

Logit stands for **log**istic un**it** [35]. As already described earlier, the relationship between independent and dependent variables is assumed to be linear and is given by the following Equation 2.11, where β_i for $0 \leq i \leq d$ are regression coefficients mapping the effect of d independent variables on the outcome.

$$logit(P) = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_N * X_N \tag{2.11}$$

This leads to the desired logistic formula in Equation 2.12.

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_N * X_N)}} \tag{2.12}$$

In the last step, the coefficients are estimated to get definitive values for classification.

During training or so-called fitting, these coefficients are estimated by the Maximum Likelihood Estimation (MLE). MLE seeks to find the set of parameters that maximize the likelihood of observing the sample data.

The logistic regression model can be modified to address problems with more than two potential outcomes, transitioning to a multinomial logistic regression to accommodate multiclass issues. While this adaptation involves several changes, a detailed discussion falls beyond the scope of this brief overview. However, for those interested in exploring the intricacies of this extension, the chapter "Polytomous Logistic Regression" in "Applied Logistic Regression Analysis" by Menard [36], which provides a comprehensive exploration of the subject, is recommended.

[2] https://towardsdatascience.com/logistic-regression-explained-9ee73cede081, accessed 04.01.2024

Probabilistic Models

Other models assume that any input provided does not contain any uncertainty. However, in the context of weak supervision, models that work with uncertainty can improve the outcome. Two probabilistic models based on the classic Bayes' theorem, shown in Equation 2.13, are the probabilistic logistic regression model and the Naive Bayes model.

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)} \tag{2.13}$$

Probabilistic Logistic Regression There are various adaptions for the logistic regression model to incorporate uncertainty [37, 38, 39, 40], but one specific probabilistic model is the Bayesian logistic regression as described by A. Johnson et al. in their book Bayes Rules! [41]. Bayesian logistic regression undergoes the following changes from the standard logistic regression. In the standard model, the values we are estimating for β_i are fixed. This means that once the training is completed we assume to know the values. The Bayesian model on the other hand assigns prior distributions to the coefficients instead, e.g., $\beta_i \sim Normal(\mu, \sigma^2)$, or other probability distributions, depending on the need. As a result, we get our prior distribution, $P(\beta)$.

Once the prior distributions are defined we use Bayes' theorem to calculate the posterior distribution, $P(\beta|data)$, of the parameters:

$$P(\beta|data) \propto P(data|\beta) * P(\beta) \ . \tag{2.14}$$

In addition to the prior, we use the likelihood of observing the given data, $P(data|\beta)$, under the assumed logistic model parameterized by β. Instead of the single logistic function, we have derived earlier, we need to calculate a distribution of predicted outcomes, which is given by

$$P(y = 1|x, data) = \int \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_N * X_N)}} P(\beta|data) \ d\beta \tag{2.15}$$

Lastly, the parameter estimation needs to be changed from MLE to a Bayesian inference method to estimate the posterior distribution of β.

Naive Bayes Johnson et al. [41] also present the Naive Bayes model in their book. The classifier utilizes Bayes' theorem to compute the probability of each category given a set of features. This probability is calculated by multiplying the probability of each feature given the category by the prior probability of the category. The algorithm then chooses the category with the highest probability as the prediction for the given input. Furthermore, there is an independence assumption is made that features are conditionally independent given the class label.

Naive Bayes can yield highly accurate prediction models but compared to the probabilistic logistic regression it lacks the regression coefficients, causing a loss of information on the relationships between the variables.

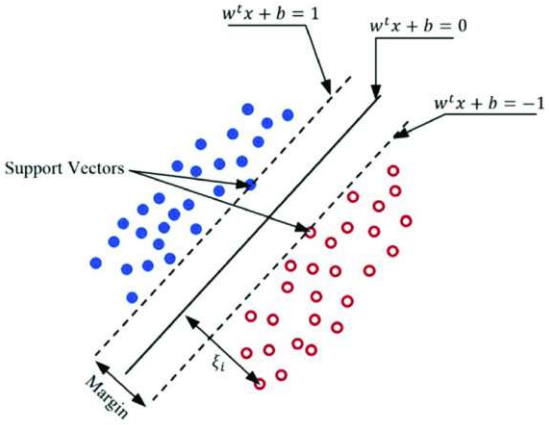


Figure 2.3: Graphical illustration of support vector machine (SVM) working principles [46]

Support Vector Machines

Another popular model choice for classification problems developed in the mid-1990's by Vapnik et al. in machine learning are Support Vector Machines (SVMs) [42, 43, 44, 45]. The underlying principle of SVMs is to identify a hyperplane that distinctly classifies data points in an N-dimensional space, where N represents the number of features in the dataset.

Given a dataset $\{(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\}$ where N is the number of data points and $y_x \in \{-1, 1\}$, we want to find the hyperplane with the maximum distance between the nearest points of each category, as can be seen in Figure 2.3. The features are represented as a d-dimensional vector, for each of the N data points.

Given this, the hyperplane is defined in Equation 2.16.

$$w^t x - b = 0 \,, \tag{2.16}$$

where w is the normal vector to the hyperplane and b is a scalar offset, called bias. The aforementioned maximum distance can then be described as $\frac{2}{||w||}$. In 1995, Vapnik and Cortes [43] coined the term *support vectors* for the data points closest to the hyperplane because conceptually only these influence the decision function.

The goal of maximizing the distance between the points and the hyperplane is achieved by minimizing $||w||^2$ subject to $y_i(w^t * x_i - b) \geq 1$, for all i, where $0 \leq i \leq N$.

However, most classification tasks are not as clear-cut, which results in a need for various approaches to mitigate the issues. One approach is using soft margins by introducing an additional slack variable allowing for misclassification. Another trick that is applied to most SVMs is the kernel trick. This involves mapping the input data into a higher-dimensional space where a linear separation will become possible.

Similar to logistic regression an adaption to SVMs is necessary in the case of multinomial classification. In the space of SVMs there are two main techniques to adapting to fit

more classes: one-versus-all (OvA) and one-versus-one (OvO). In the OvA approach a separate binary SVM is trained for each class to distinguish that class from all others, resulting in as many SVMs as there are classes. In the OvO approach, an SVM is trained for every pair of classes, leading to $\frac{C^2-C}{2}$ SVMs for C classes compared to the C SVMs in the OvA approach. Both methods classify each data point in each SVM and apply a voting mechanism or probability calculation afterward to get a final label. Depending on the number of classes this can be a computationally expensive process. One positive aspect of these approaches is, however, that applying soft margins and kernels is still possible.

Tree-based Ensemble Methods

The following ensemble methods are built upon decision trees. Decision trees are tree-like classifiers, where branches are decisions and leaf nodes are outcomes. They can be based on branching strategies of various complexity, but the decisions are overall easy to interpret and understand. However, decision trees are sensible to overfitting, especially on deep trees. This is where ensemble methods shine, utilizing multiple decision trees and aggregating the results for a final prediction [47].

Random Forest Classifier A random forest classifier is a variation of a decision tree [48]. It is designed to reduce bias and sensitivity to the training data by creating a forest of decision trees, most often based on subsets of the data, and combining their predictions. The algorithm works by randomly selecting subsets of features and samples from the training data to create indipendent decision trees. This allows for parallel execution and also helps to ensure that the trees are diverse and not overfitting to the data.The final prediction is then made most often by a majority voting on the predictions of all the trees in the forest.

This approach can improve the accuracy and robustness of the model, making it a popular choice in many applications.

Gradient Boosting Classifier Gradient Boosting Classifiers are flexible in regard to choosing their learners and while most often based on decision trees, can be based on a variety of models, including linear models such as logistic regression or support vector machines. Compared to the random forest classifier, the models are built sequentially and correct errors made by the previous models in a process called "boosting" [49]. In the final aggregation step, the results are summed up and learners with a higher precision are given more weight.

For the computational and time cost brought on by the sequential execution gradient boosting classifiers can achieve even higher accuracy on complex data compared to random forest classifiers but are therefore also prone to overfitting.

Table 2.3 summarizes key facts about the models in this section.

Classifier	Pros	Cons	Optimal Data Size	Feature Scale
Logistic Regression	• Efficient • Interpretable	• Assumes linearity between features	Small to Medium	Low to Moderate
Probabilistic Logistic Regression	• Can deal with uncertainty • Good for small datasets	• Complex • Computationally expensive	Small	Low to Moderate
Naive Bayes	• Fast • Interpretable	• Assumes feature idependence • Poor performance with correlated	Medium to Large	High
Support Vector Machine	• Works in high dimensional space • Versatile	• Sensitive to noise	Small to Medium	High
Random Forest Classifier	• Can handle large datasets • Can handle missing values	• Complex • Computationally expensive	Medium to Large	High
Gradient Boosting Classifier	• Effective	• Prone to overfitting • Complex	Medium	Moderate to High

Table 2.3: Summary of model stats, pros and cons.

2.2 Data Preparation

Nevertheless, before any machine learning model can be applied to data there is an inherent need for preparing the data beforehand. The preparation is a multi-stage process from collection to feature extraction.

2.2.1 Data Collection

Data collection is the first step in any data analysis. It involves gathering information from various sources to build a dataset that meets specific needs. This process is particularly critical for time series data, where observations are recorded sequentially over regular intervals. The quality, reliability, and consistency of data collection directly impact the accuracy and validity of subsequent analysis.

For time series data, it is essential to ensure consistent intervals, such as hourly, daily, or monthly, and comprehensive data with minimal gaps or missing periods. Depending on the project, data can be collected from sensors in real-time, historical records, databases, or public data repositories. The goal is to collect a dataset that accurately represents the issue and has minimal errors. Having no erroneous data points is impossible in most cases.

This stage often also involves setting up appropriate data collection mechanisms, ensuring data integrity, as well as addressing data privacy and ethical issues, especially when dealing with sensitive or personal information [50].

2.2.2 Data Cleaning

Especially when dealing with time series data, data cleaning is necessary, due to the sequential nature of the data and its dependency on time-based consistency. The primary objective of data cleaning is to identify and correct anomalies, fill in missing values, and remove noise while preserving the underlying patterns and trends of the time series.

Data cleaning techniques for time series data can be more complex than other data types. For instance, handling missing values might involve techniques such as interpolation [51, 52, 53], where missing data points are filled in based on the values of nearby points, or imputation [54, 55, 56], where statistical methods are used to estimate missing values. Outliers, or extreme values that significantly deviate from the rest of the data may need to be adjusted or removed, depending on their nature and impact on the overall data analysis [57].

From the survey on time series data cleaning approaches by Wang and Wang [58], we can conclude that while some approaches might have very little overhead they might not be accurate because they can alter correct data and that the approach should be chosen depending on the problem as not every approach is suitable.

Therefore, effective data cleaning is crucial as it impacts the reliability and accuracy of subsequent analyses. Poorly cleaned data can lead to misleading results, making this step a fundamental part of any time series analysis process.

2.2.3 Feature Extraction

Once the data is cleaned up, features can be engineered and extracted. Feature engineering is the process of analyzing a set of time series and creating features, by applying mathematical and statistical operations on the time series [59]. These features are often based on domain knowledge so that they are representative of the time series characteristics. Feature extraction from time series data involves distilling these important characteristics or attributes from the raw data into a more manageable set of data points or features by applying a set of operations on the time series. To gain an understanding of the underlying distribution of the data, statistical features are extracted over the complete series or with a sliding window approach. These features include, for example, the mean and variance, the skewness, and kurtosis. Furthermore, this process is focused on identifying patterns, trends, and relationships within the time series that are significant for the analysis or predictive modeling. Common methods include Fourier transforms to capture periodic patterns and seasonality and wavelet transforms for decomposing the time series into different frequency components. Autocorrelation features can be used to quantify how the current value of the series is related to its previous values. However, features that extract more information and have more difficult underlying math have a significantly higher increase in computational cost if the time length of the time series increases.

In recent years, there has been a surge in tools and approaches for feature extraction as the relevance of data analysis continues to rise [60, 61, 62, 63]. T. Henderson and B. Fulcher have shown that feature extraction is a promising approach to tackling industry and scientific problems but that a selection of features still has to happen since there are lots of correlations between features making them redundant [64]. A. Renault et al. [65] have shown in their paper that `tsfresh` and `catch22` are among the best performers in feature extraction for classification tasks.

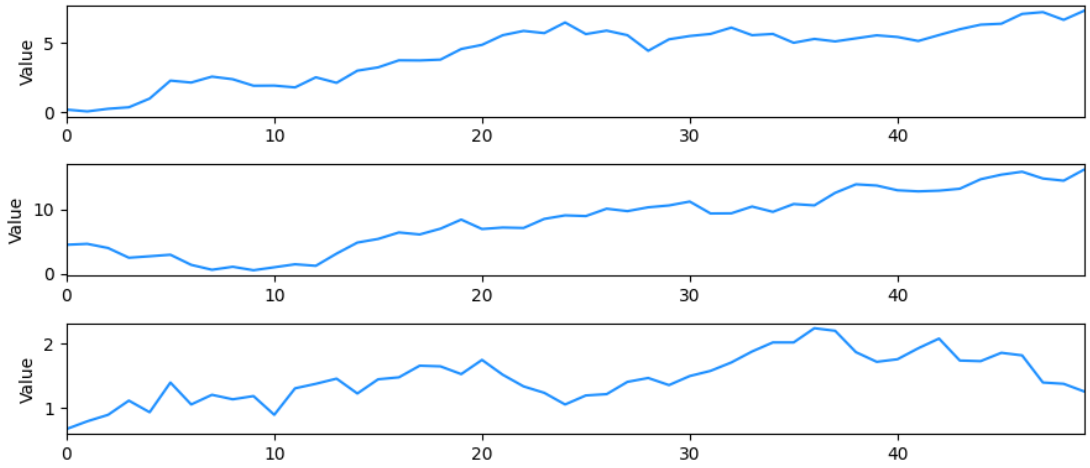


Figure 2.4: Set of example time series for feature extraction.

	Raw Data	Mean	Variance	Skewness
TS1	[0.19, 0.06, 0.25, 0.36 ... 7.12, 7.25, 6.69, 7.36]	4.35	4.05	-0.65
TS2	[4.5, 4.64, 3.99, 2.47 ... 15.91, 14.85, 14.51, 16.27]	8.28	21.67	-0.11
TS3	[0.68, 0.8, 0.9, 1.12 ... 1.82, 1.4, 1.38, 1.26]	1.48	0.13	0.05

Table 2.4: Feature extraction table

Feature extraction therefore aims to transform the raw time series data into a set of representative features that encapsulate the distinguishing information in a form that is more amenable to analysis or machine learning algorithms.

An example illustrates the process of converting raw time series data, as shown in Figure 2.4, into a feature matrix, which is presented in Table 2.4. The example time series data is created using a cumulative sum over 50 random predictions, and scaling the resulting series by a negative value.

2.3 Data Labeling

Now that we have defined our models and have prepared our data we are going to focus on the labeling aspect. The history of data labeling is closely tied to the history of machine learning. From 1950 onwards data was manually labeled, the knowledge age did not change much. With the establishment of machine learning as a distinct field, there was a rise in semi-automated, machine-supported labeling. From 2000 onwards crowdsourcing services like Amazon's Mechanical Turk gained popularity allowing the outsourcing of the labeling to a human workforce [66, 67].

In more recent years approaches have emerged that try to eliminate the need to label

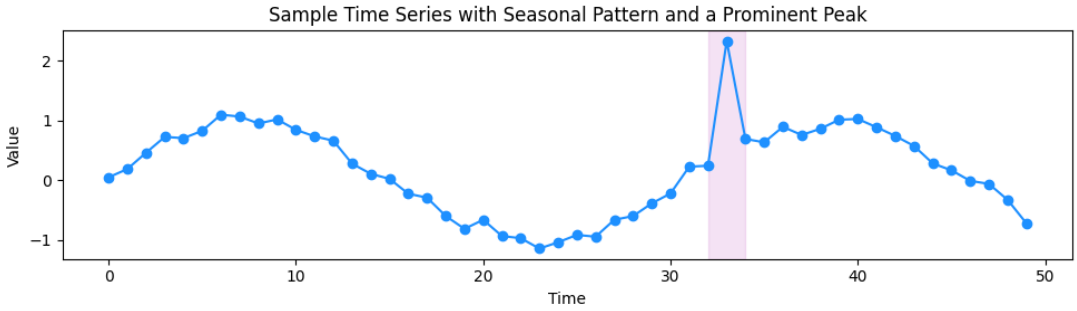


Figure 2.5: A time series that can be labeled seasonal with an annotated outlier

data such as self-learning and unsupervised learning, as well as approaches to leverage machine learning to generate labels, such as in the case of weak supervision.

2.3.1 Annotation and Labeling

While these words are often used interchangeably there is a difference.

Time series labeling or classification involves assigning a label or category to an entire time series or specific segments of it. Labels are predefined categories or classes. For instance, in a time series data of stock prices, periods could be labeled as "bull market" or "bear market" based on the trend observed in the data. Labeling is commonly used in supervised learning, where these labels serve as training data for models to learn from. Data labeling is usually performed on larger sets of data.

On the other hand, time series annotation, while similar to labeling, involves providing more detailed information or notes to specific points or intervals in the time series data. Annotations can include labels, but they can also provide additional context or explanations. For instance, in a time series of weather data, a sudden spike in temperature might be annotated with a note mentioning a heatwave event during those days. Annotations are helpful for data analysis and interpretation, providing insights that might not be evident from the raw data itself. Annotation is typically done on single or small batches of time series [68].

In summary, labeling is about categorizing or classifying data, while annotation is about enriching data with additional context or information. Both are essential in data analysis, particularly when working with complex time series data, where understanding the nuances and patterns over time is crucial, an example is shown in Figure 2.5.

2.3.2 Gold Labels

The task of labeling is both challenging and time-intensive, driving the push towards automating the labeling process. Given that large models require vast datasets, the manual labor involved in labeling this data represents a significant cost. Automating

parts of this process aims to mitigate these issues by reducing the need for manual intervention.

In assessing the effectiveness of automated labeling approaches, the metrics established earlier can be applied. The main challenge, however, lies in establishing a benchmark for comparison. These benchmark or "ground truth" labels are known as gold labels. These gold labels are supposed to be right and hence are used for evaluating the accuracy of any labeling tool. Despite their importance, it is important to acknowledge that these gold standards are not flawless either. Human-generated labels are susceptible to errors and exhibit bias [69, 70, 71].

This reality further underscores the potential value of automated labeling methods and brings us directly to the next chapter about the current state of data labeling.

CHAPTER 3

Related Work

This chapter provides a broad overview of the research directions in regard to data, and more specifically time series data, labeling from a research, as well as a practical application perspective.

3.1 Related Research

The research area of label generation for machine learning training data has been gaining significant traction in recent years, emerging as a critical domain in the field of artificial intelligence. This growing interest stems from the fundamental role that accurately labeled data plays in the development and performance of machine learning models. While there are many promising results in the recent research one main hindrance for comparison is the lack of standardized benchmarks [72]. The current research directions can be broadly categorized into the following four approaches.

3.1.1 Deep Learning and Neural Networks

These approaches leverage complex and novel neural network architectures to extract meaning full features and classify time series data seemingly without the need for manual input.

Zhao et al. (2017) [73] explored deep learning techniques to improve labeling performance and developed a novel convolutional neural network (CNN) for time series classification. They showed that their CNN outperforms the state-of-the-art methods at the time, by automatically internalizing the structure of the input time series. However, their approach is limited by the fact that the time series need a fixed length, as well as by the time it takes to train the network and adapt its parameters.

Fawaz et al. (2019) [74] re-implemented nine end-to-end deep learning classifiers and evaluated their accuracies for 85 univariate time series data sets. They show in their

review that deep neural networks can be highly effective in time series classification, with advanced architectures bringing significant improvements in uni- and multivariate time series classification tasks.

Yu et al. (2021) [75] apply recurrent neural network (RNN) based autoencoders to multivariate time series and evaluate the performance of different variations. Their main findings are that contrary to the general practice of rearranging the sequence of input and output time series this has no significant benefit. Furthermore, they show that only gated RNN architectures are able to encode time series data and that there is no significant difference between the gated architectures they studied. Lastly, they show that bidirectional RNN autoencoders slightly outperform unidirectional ones.

3.1.2 Semi-Supervised Learning and Weak Supervision

Semi-supervised and weak supervision approaches are applicable if there is a subset of labeled data and a large set of unlabeled or inaccurately labeled data. These methods are designed to enhance and learning efficiency and performance on large data sets. The methods are not necessarily based on classifying time series data.

Ratner et al. (2018) [76] propose a framework that allows them to break down problems into sub tasks, solving them individually and noisily. After combining the results in a matrix and solving the completion problem they achieved better labeling performances. They have shown that their approach improves the end model performance on a named entity recognition, relation extraction, and medical document classification problem, compared to a hand-labeled approach, a majority voting model, and a prior weak supervision denoising approach.

The work of Khattar et al. (2019) [77] is directly based on the work of Ratner et al. (2016) [6]. They use programmatic labeling and apply weak supervision learning to wearable sensor data of patients with Parkinson's to predict a negative progression in their health. They are able to show that their performance comes close to the performance of hand-labeled data.

Wang et al. (2022) [78] propose a framework that allows them to include unreliable pseudo-labels into their image classification model. They show that their proposed semi-supervised semantic segmentation framework outperforms many state-of-the-art methods. However, the increased performance necessitates an increased amount of time needed for training.

3.1.3 Time Series Classification with Specific Techniques

This section will delve into a range of specialized techniques designed to classify time series data. These methods are distinct from the approaches described earlier and are particularly useful when dealing with complex or unconventional data sets. By providing an overview, we hope to gain a broader understanding of how else time series data can be analyzed and classified.

In their review, Gonzalez et al. (2018) [79] compared self-labeling techniques from semi-supervised classification models. They conducted experiments on 35 different datasets using various learning schemes. The researchers concluded that the nearest neighbor rule is a reliable option, and multi-classifier self-labeled approaches show promising results.

Abanda et al. (2019) [80] conducted a review of classifiers that use distance-based techniques for time series analysis. They studied various approaches, such as the 1 nearest neighbor (1-NN) classifier and its derivatives, as well as methods that use distance measures to construct features and kernel constructing approaches. The study found that the newer approaches outperform the plain 1-NN classifier and can close the gap to traditional classifiers. However, these new approaches also come with new weaknesses that should be taken into consideration.

3.1.4 Self-Supervised Learning

Self-supervised learning is a paradigm of unsupervised learning. It represents another shift from traditional learning, which relies on labeled datasets, to unsupervised learning paradigms that allow machines to generate their own supervision signals. Through unsupervised learning, self-supervised models generate data representations that can be used in subsequent machine learning stages. This approach has shown promise in reducing the need for large labeled datasets, thus making it an appealing area of research in recent years.

While most of the aforementioned research is about generating labels, Spathis et al. (2022) [81] focus their work on avoiding the need for labels. They demonstrate the potential of self-supervised models in the domain of intelligent health, though these models can incur high computational costs. Another main issue that arises in health monitoring is the considerable gap between Proof of Concepts (PoCs) and production deployments and the heterogeneity of the sensors on the market. Furthermore, in their review, Rani et al. (2023) [82] emphasize the broad applicability of self-supervised learning, specifically through tasks in healthcare and computer vision.

Each group of papers reveals significant insights and advancements in their respective areas, showcasing the diverse approaches to the ever-present topic of data labeling.

3.2 Available Tools

We have analyzed eleven tools or frameworks for their applicability in the realm of data labeling and annotation. We took a closer look at the following tools and outlined their main use and unique selling points. While we made no difference in the previous section between the main approaches to labeling data, we are going to differentiate now. To group available tools, we separated them by their focus in regard to classifying whole data or annotating parts. Further, we have grouped the tools according to their data type and lastly listed tools using weak supervision and programmatic labeling. A summary of the outlined tools can be found in Table 3.1.

3.2.1 Data Labeling and Annotation Tools

Amazon SageMaker Ground Truth [83] is a data labeling service by Amazon that is fully managed and proprietary. It is designed to create accurate training datasets for machine learning, and it uses a combination of automated and human-in-the-loop labeling processes. This service enables the labeling of large volumes of data including images, text, and videos. It also allows users to hire a workforce to manually label the training data and provides access to the Amazon Mechanical Turk workforce for this purpose.

Label Studio [84] is an open-source platform that provides extensive flexibility and customizability for data annotation across various data types, including images, text, audio, and video. Its adaptability to different project requirements and support for collaborative labeling make it ideal for diverse and complex data labeling tasks. The main focus of the tool is however data annotation, rather than labeling big datasets.

Prodigy by Explosion AI [85] is also a commercial annotation tool that excels in rapid and efficient data labeling, particularly in Natural Language Processing (NLP) and image annotation. It integrates active learning to improve annotation efficiency, making it an excellent choice for quickly refining and improving the quality of labeled data.

Labelbox [86] is a versatile tool that provides capabilities for both data annotation and labeling, with a focus on annotation. It supports creating custom labeling interfaces and collaborative labeling features, making it an ideal choice for those who want to customize their interfaces and collaborate with others while labeling data.

Rubrix [87] is an open-source tool focused on labeling and monitoring data for NLP models. Rubrix is designed for data exploration and integrates human feedback into the model training process. Its streamlined annotation process for NLP, with a focus on integrating human insights into the machine learning workflow, makes it an excellent choice for those looking for an efficient and collaborative tool.

3.2.2 Time Series Data Tools

Visplore [88] specializes in visualization and exploration of time series data, particularly useful for industrial and sensor data analysis. Its high-performance data analysis with a focus on interactive exploration makes it ideal for complex industrial datasets.

Trainset by Geocene [89] is a tool specifically designed for annotating time series data, valuable in applications involving sensor data or sequential data. Its dedicated focus on time series data and user-friendly interface for labeling such data make it an efficient and easy-to-use tool.

3.2.3 Weak Supervision and Programmatic Labeling Tools

Snorkel [6] is an open-source framework that uses weak supervision to generate labeled datasets. Its programmatic approach to labeling reduces the need for extensive manual

Tool Name	Mainly Used For	Focus	Cost
Amazon SageMaker Ground Truth [83]	Various types (images, text, video)	Labeling	Subscription (AWS services)
Label Studio [84]	Various types (images, text, audio, video)	Annotation & Labeling	Free and Paid Plans
Visplore [88]	Time series data	Annotation & Labeling	Subscription-based
Trainset by Geocene [89]	Time series data	Annotation	Free (Open-source)
Snorkel [6]	Various types	Labeling	Free (Open-source)
WeaSEL [4]	Various types	Labeling	Free (Open-source)
Prodigy by Explosion AI [85]	Various types (NLP, images)	Annotation	Commercial (One-time purchase)
Labelbox [86]	Various types (images, text)	Annotation & Labeling	Free and Paid Plans
Rubrix [87]	NLP	Annotation & Labeling	Free (Open-source)
Reef (formerly Snuba) [90]	Various types	Labeling	Free (Open-source)
Darwin	NLP	Labeling	Research/Not Publicly Available

Table 3.1: Data Annotation and Labeling Tools

labeling, making it an efficient choice for those looking for a faster labeling process for large datasets. One of the significant benefits of this approach is also the ability to easily update and improve the labeling functions based on model performance. However, the quality of the labels produced is heavily dependent on the quality of the labeling functions designed by the users and the gold labels the model is evaluated against. Poorly designed functions or bad gold labels can lead to inaccurate labels. As a result, effective labeling functions require a thorough understanding of both the data and the domain. Another point is that it is recommended to train an end model with the generated labels to increase the performance and generalize over the labeling functions.

WeaSEL (Weakly Supervised End-to-end Learning) [4] is designed for weakly supervised learning, integrating model training into the weak supervision process. Its end-to-end learning approach, which encompasses both label generation and model training, is particularly suited for complex machine learning tasks. WeaSEL aims to eliminate the need to separately train an end model by directly binding it into the label generation process. However, WeaSEL does not provide a way to generate the initial set of labels and it is recommended to use WeaSEL in conjunction with Snorkel to generate the initial heuristics that are used for the approach.

Reef [90] focuses on utilizing weak supervision by utilizing partially labeled data and a feature matrix to extract new labels. The aim is to strongly limit human input and therefore save cost. However, Reef only supports binary classification limiting its applicability in many ways.

Darwin [5] is an academic project focusing on automatically generating labeling heuristics based on an initial rule. The aim is to reduce the human input and to generalize from good heuristics by getting input from the annotator. They were able to show improved performance with their rules compared to Reef. Darwin is however not publicly available.

3.3 Distinction from Current Research and Tools

While there is an abundance of data labeling and annotation software available in the market, it is notable that the majority of these tools have not been designed with a primary focus on time series data. This data type, characterized by its sequential nature

and temporal dependencies, presents unique challenges and opportunities for analysis and interpretation. Among the available tools, Snorkel and WeaSEL stand out due to their innovative approach to data labeling, leveraging machine learning techniques to automate this often tedious process. However, a significant limitation of these tools is their lack of a graphical user interface (GUI) for the detailed analysis and development of labeling rules and their subsequent results. This is a crucial aspect as a GUI can greatly enhance the user experience, making the tool more accessible and intuitive, especially for users who are not proficient in programming.

The distinct gap in the market is, therefore, the absence of tools that specialize in the exploration and labeling of batch time series data. This specialization would ideally involve the utilization of already developed models and strategies that are tailored for time series data, enabling more accurate and efficient labeling. The proposed tool aims to fill this gap by providing a comprehensive and user-friendly interface that facilitates the exploration and labeling of time series datasets.

Furthermore, the overarching goal of developing such a tool is to investigate the feasibility of this approach specifically for time series data. Time series datasets are prevalent in various domains such as finance, healthcare, and software system monitoring. Hence, a tool that efficiently labels and annotates this type of data could have significant implications for these fields. It could enable more effective data analysis, leading to better decision-making and advancements in these sectors. The development of this tool could thus be a pivotal step in enhancing the capabilities of data analysis and interpretation in the context of time series data.

CHAPTER 4

Label Generation

The initial idea for automated label generation is based on the research in Section 2 and builds directly upon the frameworks in Section 3. Therefore, we want to develop a machine learning supported label generation process, decreasing the need for human labeling. Transferring the research on weak supervision labeling for other data types to time series, we should be able to achieve this by leveraging a combination of labeled and unlabeled time series data, along with a set of newly created labeling functions that encapsulate pertinent labeling properties, and weak supervision models. The labeling functions should be designed, as well as refined, by domain experts for optimal performance. To make designing and refining labeling functions as easy as possible a graphical user interface should be used. A more detailed approach is outlined in the following Section 4.1 and followed by the specific implementation in Section 4.2

4.1 Approach

Tackling the idea we iterated through versions of data and workflows, evaluating various subtasks of the labeling process. With the first idea formalized in Figure 4.1, the limiting factor proved to be in the amount of computation time necessary for time series feature extraction. During testing, it has proven to be infeasible to compute time series features on the fly because these calculations can take anywhere from minutes to hours depending on the amount of time series in the analyzed set. As a result, time series feature extraction has been decoupled from the labeling step, shown in Figure 4.2. Therefore, all time series features are calculated beforehand, to enable interactive exploration and labeling. Once a feature matrix is calculated, time series and features can be used for exploration. Users should be able to create labeling functions utilizing the feature matrix and get a preemptive result of the effect of the labeling function. Once there are enough functions for the labeling model to vote on results, the labels can be calculated for a ruleset. The data set will then be weakly labeled with the labeling functions and is inspectable

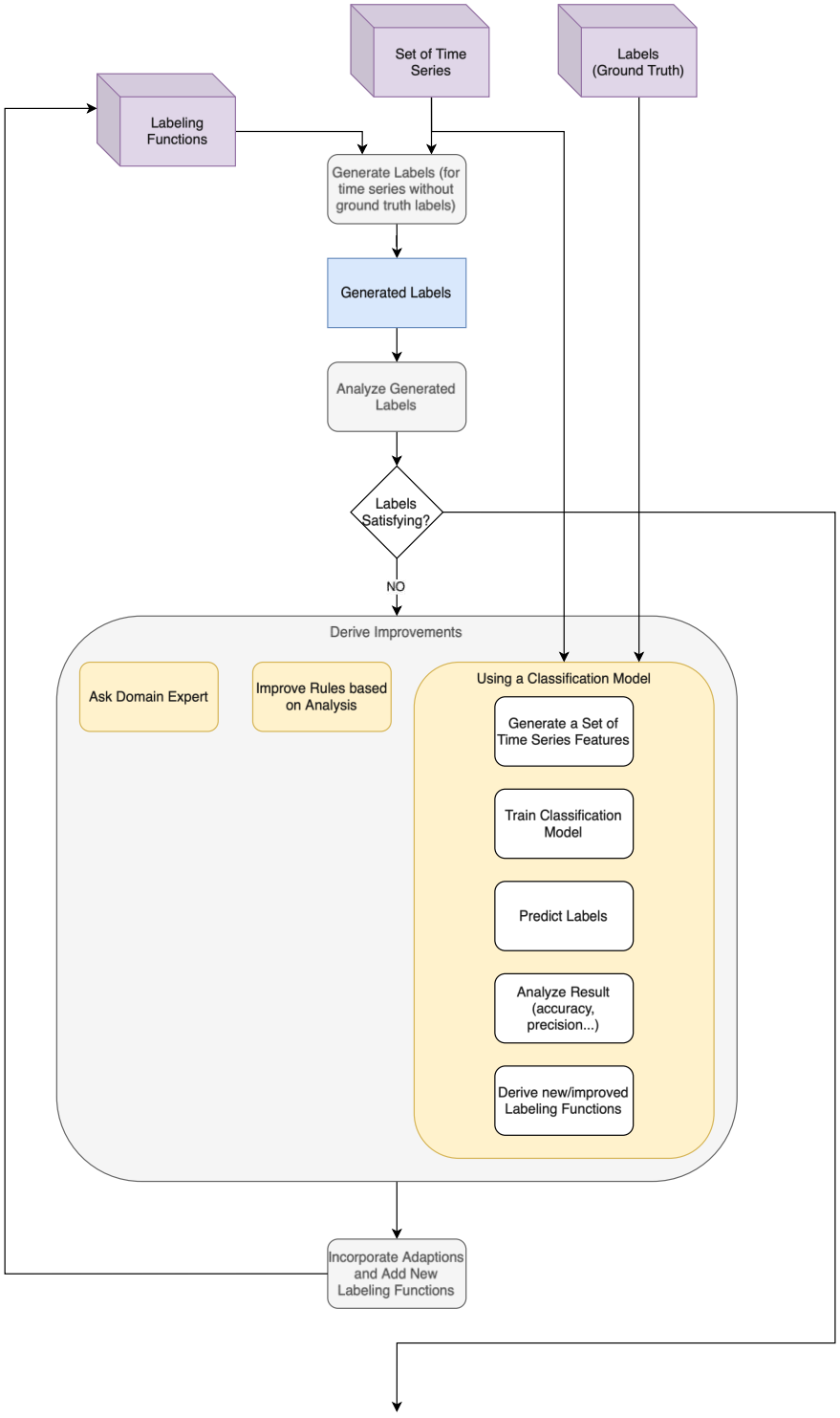


Figure 4.1: Initial labeling workflow flow chart

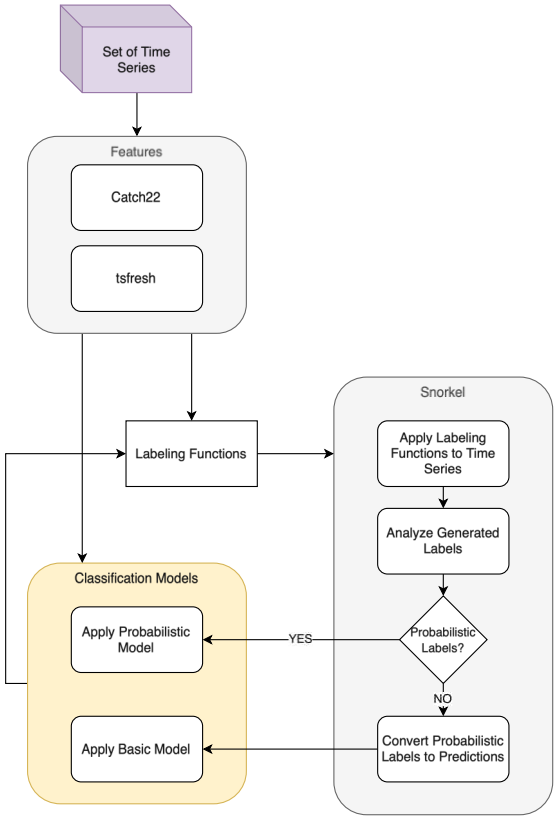


Figure 4.2: Labeling workflow with decoupled feature extraction

afterward. Finally, the data will flow into a classification model to generalize above the labeling heuristics.

In a final iteration, the workflow and approach has been adapted and extended to include all necessary steps for labeling and evaluation. The final label generation workflow, shown in Figure 4.3, was conceptualized to utilize two sources for time series data and predefined labels. The time series data should be provided directly through a database, while the pre-existing labels are read from a `.csv` file and matched to the respective time series. The raw time series data undergoes a preparation step to handle any missing values, after which features are extracted from the data to create a feature matrix (FM). This results in two versions of the FM: one without ground truth labels and another that includes ground truth labels.

Within the scope of the demo application, as indicated by the green box, labeling rules are applied to the FM that lacks ground truth labels, producing an FM that now contains rule-based labels. This FM is subjected to visual testing to assess the extent of changes introduced by the rule-based labeling. If significant changes are observed, the users should cycle back to refine the labeling rules. If the changes are minor, the labeling can proceed to the next phase. The feature matrix including the gold labels will be

split into training, test, and validation sets. In the case of the demo, the validation set is completely separated so that any bias of users can be avoided. The train and test set are then used to train and evaluate a surrogate model. If the labels are deemed unsatisfactory at any point, the users should loop back to enhance the labeling rules or adapt the model parameters, improving the preliminary labeling result. Once the labels meet the satisfaction criteria, the label model is considered validated, concluding the process that has been implemented within the demo application. This enclosed part of the flow represents the iterative development and validation of a labeling system designed for a time series data application.

4.2 Implementation

We have chosen to use **Python** as a programming language because it offers a vast amount of frameworks and dependencies in the area of machine learning that allows rapid development. The frameworks have been evaluated individually for their capabilities and functional interaction in Databricks[3] notebooks. We will now give an overview of the main frameworks and dependencies used in the project.

Frameworks and Dependencies

Streamlit[4] is an open-source Python library allowing developers to rapidly build data dashboards, visualizations, and interactive reports. Streamlit integrates seamlessly with most major data science and machine learning libraries, making it appealing for prototyping and a good choice for the project. However, one major drawback of Streamlit's simplicity is that on every state change, be it a callback or any other change, the whole script reruns. This makes implementation easy in many cases for small prototypes but turns out to be limiting the development in the later stages when larger amounts of data are used.

While Streamlit already offers a large number of so-called widgets for visualization and interaction with the platform, there are many functionalities missing, such as buttons to switch between pages or define dynamic rows. This is where **Streamlit-extras**[5] provides missing quality-of-life features and widgets.

Streamlit-authenticator[6] provides a way to store and retrieve user data. The idea was to make the labeling as collaborative as possible, hence an authentication service was needed to distinguish between users.

Pandas[7] and **Numpy**[8] provide the foundation for time series data manipulation and calculations in the application.

[3] https://www.databricks.com/
[4] https://streamlit.io/
[5] https://arnaudmiribel.github.io/streamlit-extras/
[6] https://github.com/mkhorasani/Streamlit-Authenticator/
[7] https://pandas.pydata.org/
[8] https://numpy.org/

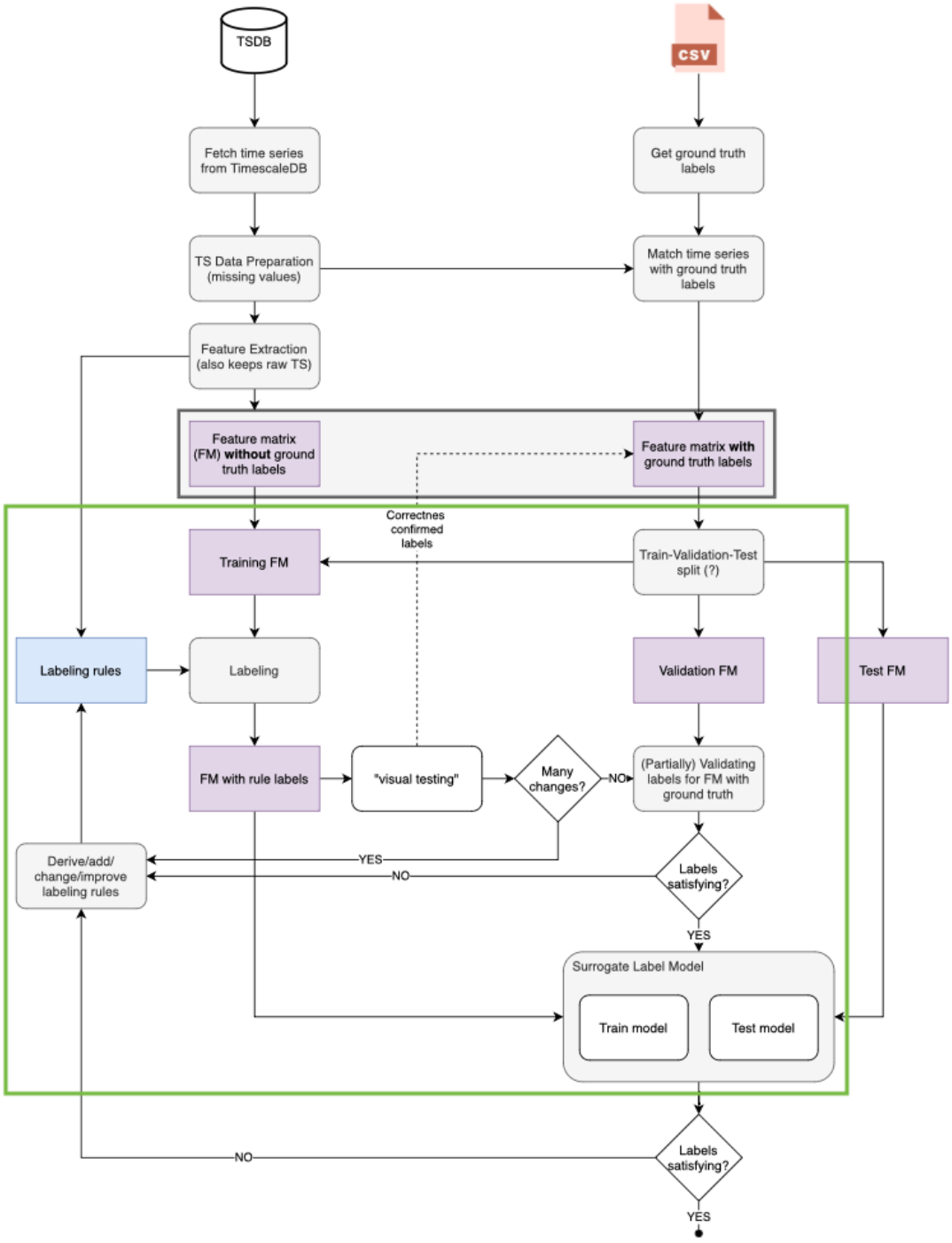


Figure 4.3: Conceptual labeling and data flow. The green box highlights the parts included in the prototype implementation.

The PoC is centered around **Snorkel**[9], which is a library that enables users to create and manage training data using labeling functions. These functions speed up the generation of labeled datasets compared to manual labeling. Snorkel also simplifies the process of developing and iterating machine learning models by enabling quick modifications to the labeling functions, which can be immediately reflected in the training data.

scikit-learn[10] and **PyTorch Lightning**[11] are two powerful libraries in the Python ecosystem that cater to different aspects of a ML pipeline. scikit-learn is widely recognized for providing an extensive suite of tools for data preprocessing, model selection, and evaluation [91]. This makes it an ideal choice for the project. In contrast, PyTorch Lightning is an extension of the PyTorch framework that aims to simplify the deep learning process. It eliminates the need for much of the boilerplate code required in vanilla PyTorch, enabling to focus more on the research aspect of the models. As a result, it provides a structured and scalable approach to building custom models. Together, these libraries build a robust foundation for the surrogate models in the application.

In contrast to Snorkel, **WeaSEL**[12] takes an end-to-end learning approach. The model is learned directly by maximizing its agreement with probabilistic labels generated using a neural network to reparameterize previous probabilistic posteriors. Even though, WeaSEL is therefore more of a holistic model making another end model obsolete, it is used as a so-called surrogate model in the application. Furthermore, Snorkel also enables generating a label matrix, which is crucial as input for the WeaSEL model.

Lastly, we are using **SQLAlchemy**[13] as a database connector. SQLAlchemy enables the creation of SQL schemas by defining the entities in the project, which makes switching between databases easier in case more or different functionality is necessary.

Architecture

In order to prepare the data for the demo, a separate project has been created which contains various scripts for loading, manipulating, and processing the data. The main demo application is divided into two separate projects: the demo project and the surrogate model project. These two applications communicate with each other through remote procedure calls. The separation of these two applications was necessary because Streamlit processes caused issues during the Pytorch Lightning training of the models. It is important to note that the demo is a standalone application that currently offers one scikit-learn logistic regression model, but more advanced models are only available if both applications are up and running. The demo application is fully containerized with Docker[14] and compose, and the startup is automated using a script.

[9] https://www.snorkel.org
[10] https://scikit-learn.org/
[11] https://lightning.ai/docs/pytorch/stable/
[12] https://github.com/autonlab/weasel/
[13] https://www.sqlalchemy.org/
[14] https://www.docker.com/

Model	Input Features	Input Labels	Example Label Input
scikit-learn Logistic Regression	Feature Matrix	Label Predictions	[0]
Pytorch Lightning Logistic Regression	Feature Matrix	Label Probabilities	[0.13 0.87]
Pytorch Lightning Probabilistic Logistic Regression	Feature Matrix	Label Probabilities	[0.13 0.87]
WeaSEL Model	Feature Matrix	Label Matrix	[0 2 0 1 1]

Table 4.1: Surrogate Model Inputs

4.2.1 Models

The following section is split into models provided by Snorkel for the initial labeling and our surrogate models for generalizing beyond the labels generated by Snorkel.

Snorkel Models

Once at least three labeling functions have been designed, they can be used in conjunction with the label models to get preliminary labels and their respective probabilities. Snorkel provides two out-of-the-box label models for users. We have decided to allow users to choose whichever model they prefer. Additionally, there is a choice of how to proceed in case the model ties in multiple classes.

Snorkel Majority Vote The Snorkel Majority Vote model is a method of combining labels from individual labeling functions and assigning each time series the label that received the most votes. This model has several advantages, such as predictability and ease of understanding. Additionally, it does not carry any kind of bias that more advanced models might have. In cases where the majority voting model is unable to decide on a label, the tie-break policy will be used to resolve the issue and an equal probability is assigned to each of the majority classes.

Snorkel Label Model This label model is based on the approach by Ratner et al. [76] The Snorkel Label Model is designed to understand and utilize the conditional probabilities of Labeling Functions (LFs) in producing the accurate, yet unobserved, label Y, symbolized as $P(lf|Y)$. This is achieved by learning a model based on these probabilities and then applying it to adjust and integrate the labels given by the LFs. Using this approach they were able to improve the performance compared to a majority voter model on average by 6.3 points.

Surrogate Models

The surrogate models, in the literature also called end-models, are utilized to further generalize the labels and possibly find patterns in the data beyond what has been specified in the labeling functions. There is an immeasurable amount of possibilities for choosing end models and fine-tuning them for a problem. We have decided to focus more on simplistic models for the performance evaluation.

scikit-learn Logistic Regression (SKL Log Reg) The scikit-learn Logistic Regression is the baseline model used. The main benefit of the model is the fast performance and that it is available directly in the demo application without running the backend supplying the more sophisticated models. Another convenient feature of this model is that it offers a multinomial logistic regression out-of-the-box. To run the model a feature matrix and a vector containing the label prediction is necessary.

PyTorch Lightning Logistic Regression (PTL Log Reg) This model is a basic logistic regression implementation using the Pytorch Lightning framework and probabilistic input labels instead of the label predictions by the label model. This model needs the feature matrix and an $n \times m$ matrix, for n time series and m labels, to accurately model the label probability. The model code is illustrated in Listing 4.1.

```
    self.model = nn.Sequential(
        nn.Linear(input_size, hidden_size1),
        nn.ReLU(),
        nn.Linear(hidden_size1, hidden_size2),
        nn.ReLU(),
        nn.Dropout(0.1),
        nn.Linear(hidden_size2, output_size)
    )
```

Listing 4.1: Pytorch Lightning Logistic Regression Model

PyTorch Lightning Probabilistic Logistic Regression (PTL Prob Log Reg) using Monte Carlo Dropout The foundation and inspiration for the probabilistic approach were the "Probabilistic Logistic Regression with TensorFlow" article by L. Roque [92] and the "Introduction to Bayesian Logistic Regression" article by M. Kana [93] about the implementation of probabilistic methods for logistic regression. Yet, the undertaken approach is only a way to estimate the uncertainty and not a full Bayesian inference approach. This model is used to gauge if more advanced models might make sense to be included. This model, as shown in Listing 4.2, has the same input as the previous Pytorch Lightning model.

```
    self.model = nn.Sequential(
        nn.Linear(input_size, hidden_size1),
        nn.ReLU(),
        nn.Dropout(0.1),  # Dropout layer for Bayesian approximation
        nn.Linear(hidden_size1, hidden_size2),
        nn.ReLU(),
        nn.Dropout(0.1),  # Another Dropout layer
        nn.Linear(hidden_size2, output_size)
    )
```

Listing 4.2: Pytorch Lightning Probabilistic Logistic Regression Model

WeaSEL Model This model is an application and adaption of the model defined by S. Rühling Cachay [4]. In contrast to the other models, this model utilizes the

complete labeling matrix by Snorkel and does not depend on the Snorkel labeling models. Furthermore, the WeaSEL model has a mechanism to integrate the probabilistic labels into the downstream model. Hence, the model takes the feature matrix and an $n \times m$ matrix, for n time series and m labeling functions, as shown in Listing 4.3.

```
endmodel = MLPNet(dropout=0.3, net_norm='none', activation_func='ReLU',
input_dim=features.shape[1],
                hidden_dims=[10, 10, 5], output_dim=n_classes)

weasel = Weasel(
    num_LFs=labels.shape[1],
    n_classes=n_classes,
    temperature=2.0,
    accuracy_scaler='sqrt',
    use_aux_input_for_encoder=True,
    class_conditional_accuracies=True,
    encoder={"hidden_dims": [32, 10]},
    optim_encoder={"name": "Adam", "lr": 1e-4},
    optim_end_model={"name": "Adam", "lr": 1e-4},
    scheduler=None,
    end_model=endmodel,
)
```

Listing 4.3: WeaSEL Model Definition

4.2.2 Database Design

The database structure (Figure 4.4) comprises several interconnected tables each serving a specific function. The User table stores basic information about users, including ID, name, and email. Labeling Task holds details about labeling tasks, such as task ID, name, data URL, and labels. Ruleset links to User and Labeling Task tables, containing ruleset ID, name, user ID, and labeling task ID. Gold Label records the ground truth labels for tasks, referencing Labeling Task and User tables through task ID and user ID.

Rule table contains individual rules, linked to the Ruleset table via ruleset ID. Finally, Labeling Result table tracks the outcomes of labeling efforts, including links to Labeling Task, Ruleset, and User tables. These tables collectively support a comprehensive system for managing labeling tasks, rules, and results, emphasizing relationships among tasks, users, rulesets, and labeling outcomes.

The database was initially designed around SnowflakeDB with the feature of storing semi-structured data as `VARIANT` inside the labeling results. However, to make the demo available to a wider audience and guarantee reproducibility in a non-commercial environment a switch to SQLite was necessary. The `VARIANT` fields were converted to `VARCHAR`s which meant a loss of functionality from the database side.

[15] Diagram created with `https://dbdiagram.io/d`

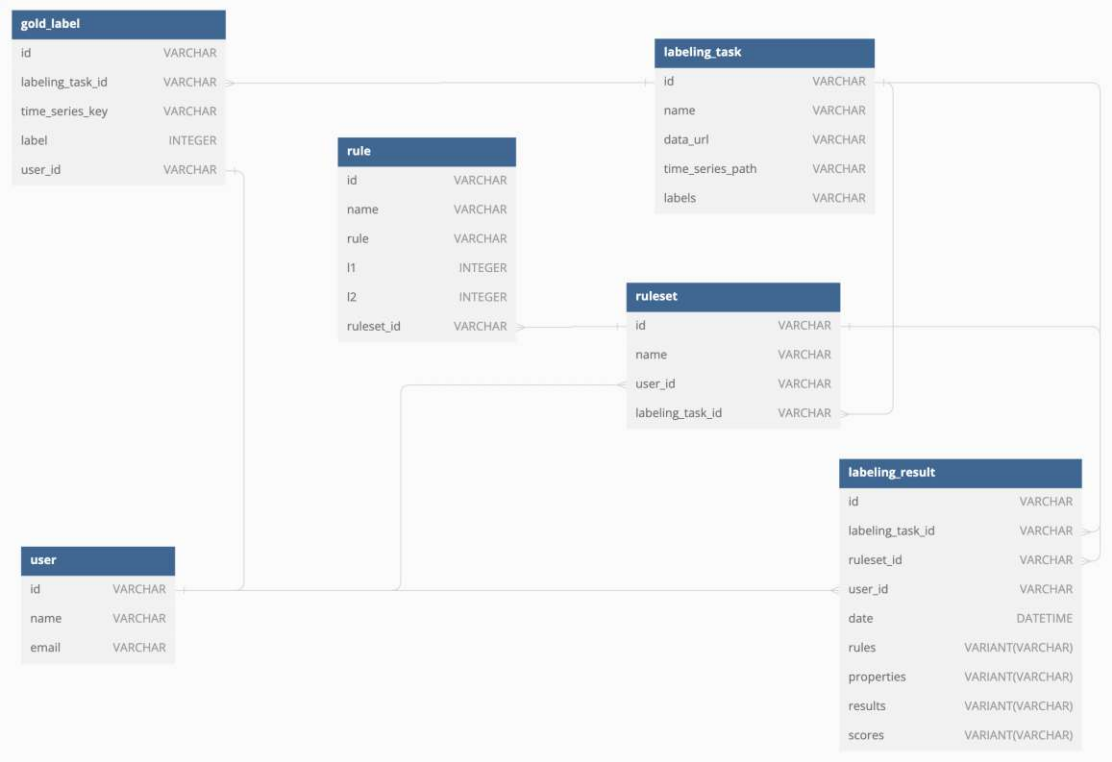


Figure 4.4: Database relations visualized[15].

In the demo application the database is created from the model entities should it not exist upon startup. A translation of the process can be done to extract an equivalent database creation file using SQLAlchemy.

4.2.3 Functionality and User Interface

Home

The Home page is the first page a user lands on when opening the application in the browser. It introduces the demo, a sample task, and the use case, as well as links to the additional information page in case there are any open issues. In case a user arrives on the Home page for the first time there is also an authentication section. If a user is not authenticated, none of the other pages are navigable and the application redirects to the Home page. The intention is less to keep intruders out but to gain insight into who worked on and labeled which part. Once logged in, everything is usable.

Sidebar

Furthermore, once a user is authenticated, the Sidebar on the left-hand side displays the name of the authenticated user, the selected labeling task, and the labeling ruleset.

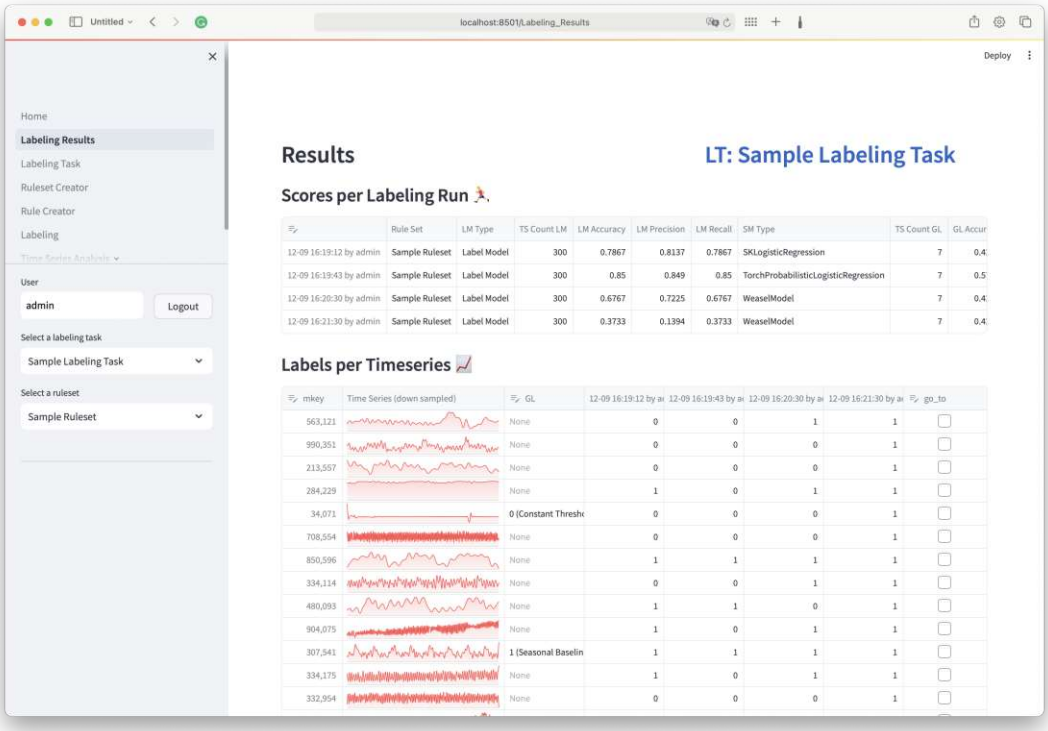


Figure 4.5: Labeling Result Overview

Labeling tasks and rulesets can be changed at any point in the application via the drop-down menus on the sidebar. Changing a labeling task can incur a waiting period because the time series data has to be loaded into the context so that the application can display the data correctly. Changing the ruleset does not incur any waiting time.

Labeling Result Overview

The Labeling Result Overview in Figure 4.5 consists of two tables. The first table, "Scores per Labeling Run", is indexed by labeling runs and shows the label and surrogate model performance per run, the used labeling ruleset, as well as the model types that were used to achieve the performance. The chosen display metrics are the model performances compared to the entered gold labels. This means that users should enter some gold labels for the results to be reliable. Further, we have to note, that these runs, and respective results, are subject to change, should additional gold labels be added. The results are sortable by each category and are used to give users an overview of past labeling performances of a specific labeling task. Additionally, there is a column with *go to* checkboxes. When ticking a checkbox a user will be forwarded to the labeling page, in Figure 4.14, with the previous parameters, allowing users to continue their efforts, as well as letting different users build on the already achieved performance.

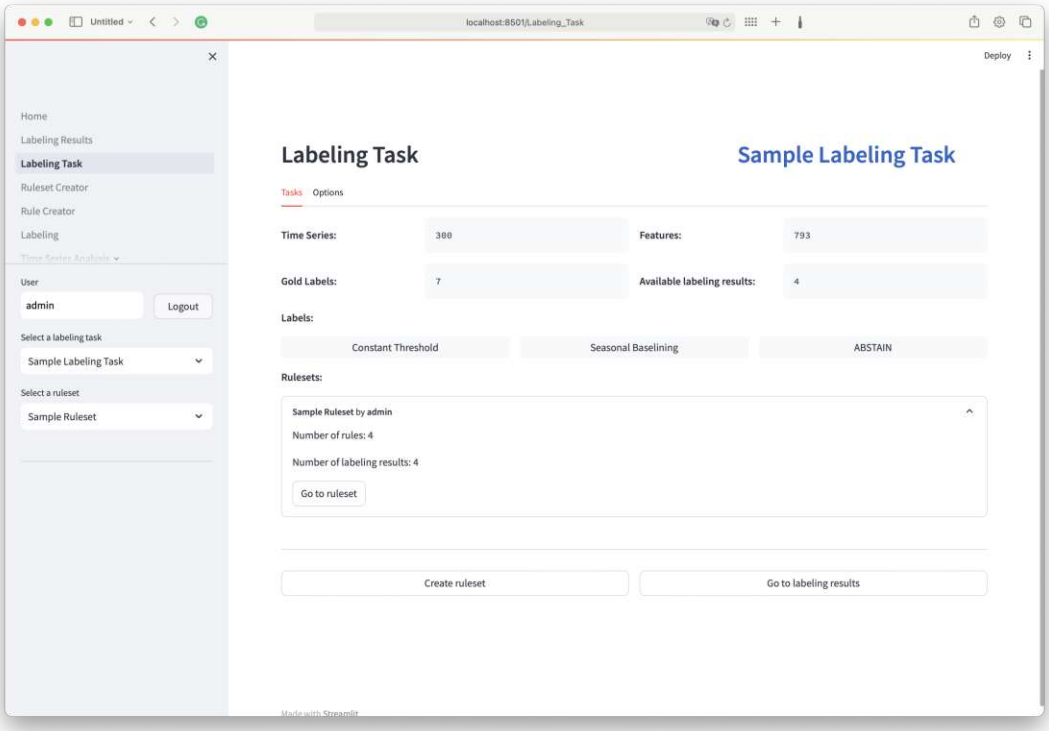


Figure 4.6: Labeling Task View

The second data frame, "Labels per Timeseries", allows users to inspect individual time series in conjunction with the labeling results. This table is indexed by time series and each labeling run is a column and the respective assigned label per time series. Users can gather additional information on how the performance of a specific run was achieved. Furthermore, users can take an even more in-depth look when clicking on the *go to* checkbox in this table, allowing them to view the time series, all features, and the labeling results on a time series analysis page, in Figure 4.19b.

Labeling Tasks

The labeling page comprises two tabs, the first tab called "Tasks", in Figure 4.6, offers insight into the selected labeling task. This tab provides a comprehensive overview of the available time series and features, the number of gold labels, and the number of existing labeling results. Additionally, the user can access information on the available labels for the given task. Lastly, there is a list of expanders for already created rulesets, specifying exactly how many labeling results per ruleset have been created. Based on this information, users can create a ruleset or view the existing labeling results.

The second tab, called "Options", in Figure 4.7, offers the possibility to add gold labels

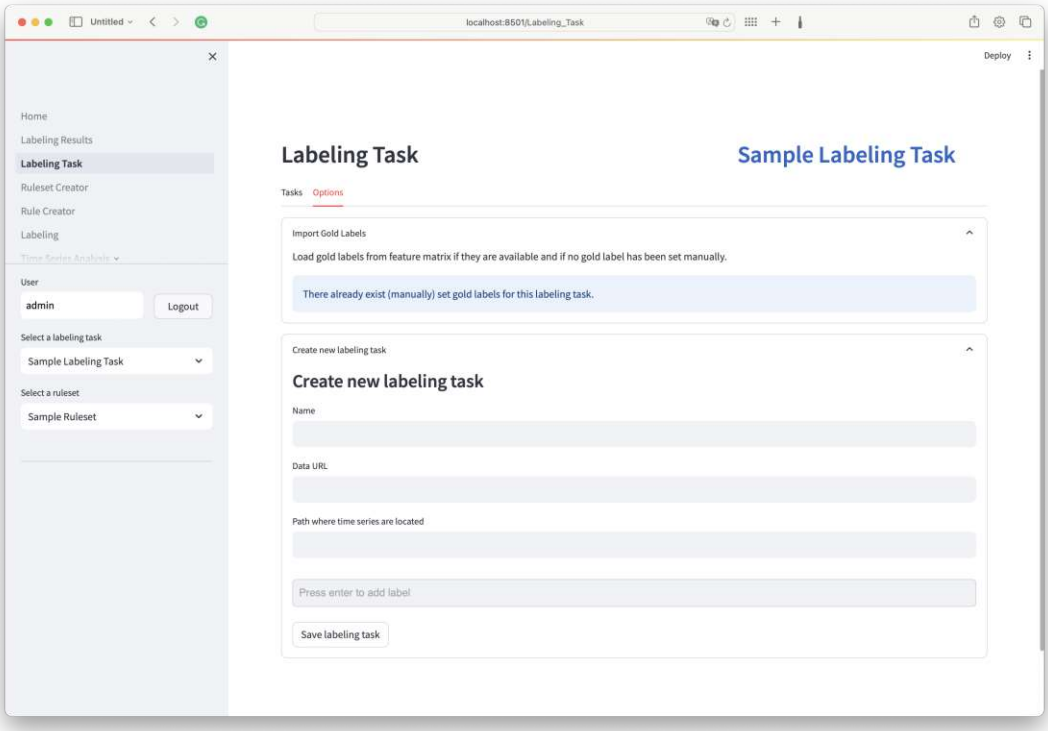


Figure 4.7: Labeling Task Options

should the feature matrix already contain some. These gold labels can only be added before any gold labels have been added by hand to avoid overwriting more recent labels. Furthermore, the labels can only be matched correctly if the label text matches exactly. Below the option for the existing task, there is another option to create a new labeling task. To create a new labeling task several properties must be specified. Every labeling task needs a name, as well as a Data URL, this is where the feature matrix is located. Moreover, the directory where the raw time series are is needed as well for displaying them and lastly, users need to provide at least two labels for the application to work. The `ABSTAIN` label will be created automatically in the background. The edit and deletion options have been hidden during the demo runs.

Rulesets

The ruleset page, in Figure 4.8, follows the pattern established on the labeling task page. There are again two tabs. The first tab, called "Rulesets", displays the created rules in expanders. Once expanded each shows the respective rule and the labels for either condition. There are two main types of rules, as will be established in the next Section 4.2.3. If rules have been created with the Free Form Rule Creator there is a leading `(*)` to indicate that these rules do not follow the basic structure. Rules with a

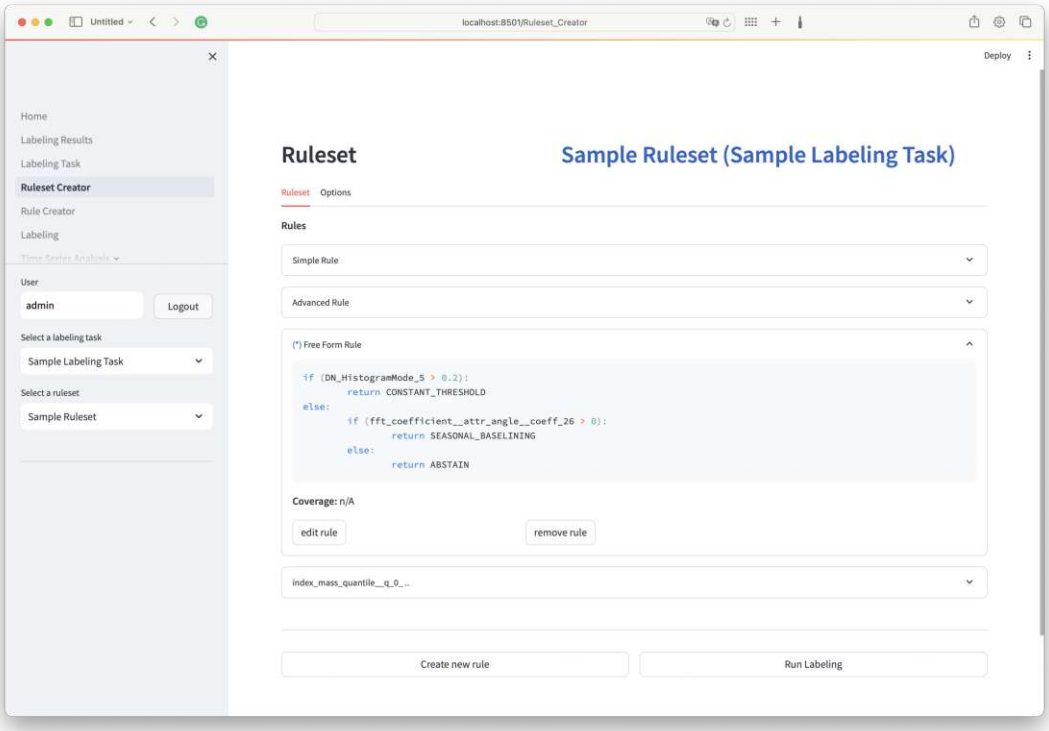


Figure 4.8: Ruleset View

leading (⋆) only show their name and the labeling code as displayed in Figure 4.8.

The options tab of the ruleset page only allows for creating more rulesets. The edit and deletion options have been removed for the use case.

Rule Creators

The application features three different rule creators, namely, the Simple Rule Creator in Figure 4.9, the Advanced Rule Creator in Figure 4.11, and the Free Form Rule Creator in Figure 4.11. These rule creators can be split into structured (Simple and Advanced) and unstructured (Free Form) rule creators. To allow for this behavior there are two mechanisms for transforming the string input into executable labeling functions.

The Simple Rule Creator, as well as the Advanced Rule Creator, follow a structure where users can provide the conditional part of an if-then-else construct, while the condition will be negated for the else part. Users can then choose the label they want to apply to the left or right part of their conditional.

For the translation of the input string to an executable labeling function we need to wrap all features contained in the rule in a lookup, as can be seen in Listing 4.4. The

`lookup` function called in the `wrap_lookup` method is a local function that is used in the local context to link the features to the function. Furthermore, the function uses a regex pattern to match all possible features excluding matching for partial feature names.

```
1     def wrap_lookup(self, rule, features):
2         for f in features:
3             def replace_feature(matchobj):
4                 return str(matchobj.group(0)).replace(f, 'lookup(x, "' + f +
      '", feature_matrix)')
5
6             rule = re.sub(r'([(<>=)&| ]|^)' + f + r'([(<>=)&| ]|$)',
      replace_feature, rule)
7         return rule
```

Listing 4.4: Method wrapping the features in a lookup function that has been put into the program context earlier.

The Free From Rule Creator allows a user to write what is close to a standard Python function, with the exception that assignments are not allowed. The allowed input can be a sequence of if-then-else statements nested as far as needed. The features are wrapped in the same war as above but the Free Form Rule Creator needs additional attention because there is a limitation regarding the Snorkel library where the outermost if-condition has to be closed. To compensate for a shortcoming of Snorkel and to avoid users having to close the condition manually there is a check that adds the outermost else clause with an `ABSTAIN` label, should it not be there, as shown in line 5 of Listing 4.5.

The full labeler then compiles and executes all functions and retrieves them from the context to be passed forward.

```
1     lf = "def {}(x):\n{}".format(
2     rule_name, self.wrap_lookup(rule.rule, features))
3
4     if lf.count('if') > lf.count('else'):
5         lf += '\n\treturn ABSTAIN'              % close the outermost if
      statement if it has not been closed to avoid issues with snorkel
6
7     compiled = compile(lf, filename="<string>", mode="exec")
8     exec(compiled)
9     lfexec = locals()[rule_name]
10
11    new_lf = LabelingFunction(name=rule_name, f=lfexec)
```

Listing 4.5: Creating executable labeling functions from strings

All rule creators offer the possibility to name the rules for easier communication with other developers when collaborating on the same task or problem. In case a rule has no name, the rule expanders will display the first 25 characters and internally an ID will be assigned for working with these unnamed rules.

Simple Rule Creator The Simple Rule Creator, in Figure 4.9, is a useful tool that allows you to explore your data by selecting features, setting the condition, adjusting

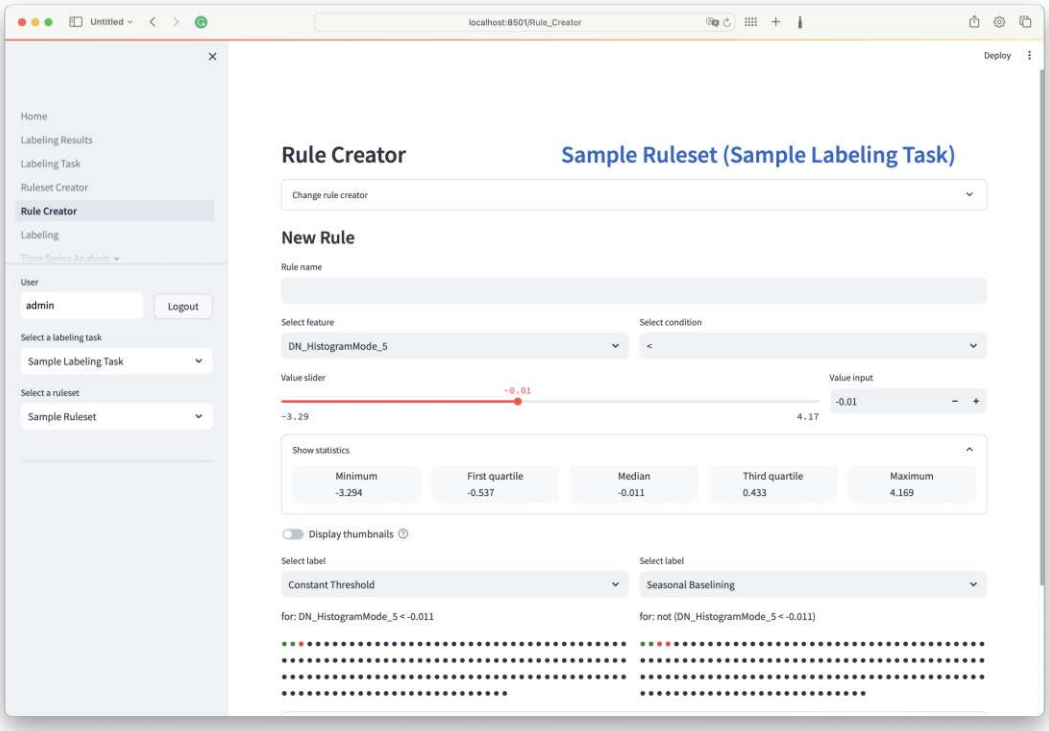


Figure 4.9: Simple Rule Creator

the value slider, or setting the value manually via the number input. When you change the feature, the slider is automatically set to the median value. This feature helps you visualize the distribution of the data, making it easier to identify patterns and trends.

In addition, the tool comes with a "Show Statistics" expander that displays important information such as the minimum value, maximum value, median, and first and third quantiles. This feature is particularly useful for identifying outliers and understanding the overall distribution of your data. By analyzing this information, you can gain valuable insights that can help you make informed decisions.

To improve the exploratory experience, each time series is represented as a dot in the time series rule list below to give users a sense of scale during exploration. This helps users easily discern the proportions of data that have been labeled a certain way. Additionally, when a time series is given a gold label, it will be highlighted in green if the gold label and the label from the label function are the same, and in red if they are different. This allows users to quickly assess how their function is performing and improve it as more time series are labeled manually.

If users desire more detailed information, they can choose to display thumbnails for the time series, in Figure 4.10. The number of thumbnails can be adjusted using a

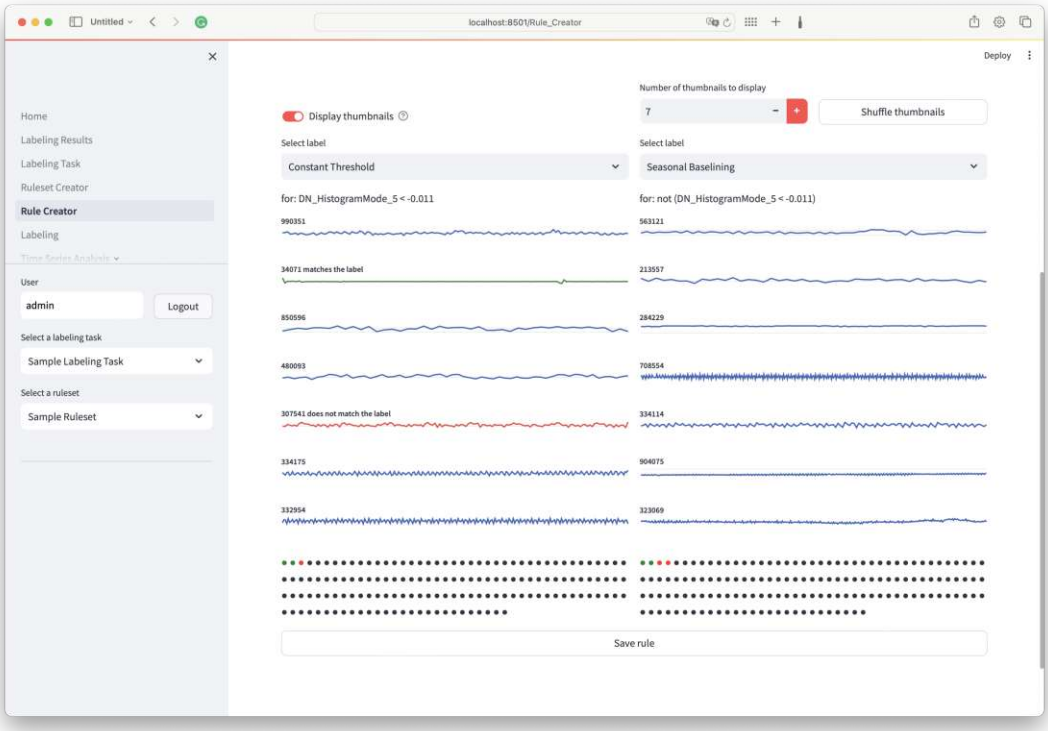


Figure 4.10: Time Series View of Simple Rule Creator

number input. To quickly browse through the data set, there is a button called "Shuffle thumbnails" changing the displayed time series on each side of the labeling condition. The time series thumbnails follow the same color pattern as the dots: green for matching labels and red for incorrect labels.

Advanced Rule Creator The Advanced Rule Creator, in Figure 4.11, operates on the same principles as the Simple Rule Creator but allows users to use logical connectives to join conditions together. A limitation is that everything is contained in one if-condition. This makes it impossible to assign more than two labels. Since the rule creator is also free-text-based, but limited to the if conditional part, there is an expander located on the bottom of the page with all available features. However, it is recommended to start exploration with the Simple Rule Creator.

Free From Rule Creator The Free Form Rule Creator, in Figure 4.12, allows users to assign multiple labels and create complex labeling functions by chaining conditionals and return statements. With this tool, it is possible to use the index of the labels or the label in all capital letters with spaces switched to underscores as a return statement. This means that users can easily create labeling functions that are tailored to their specific

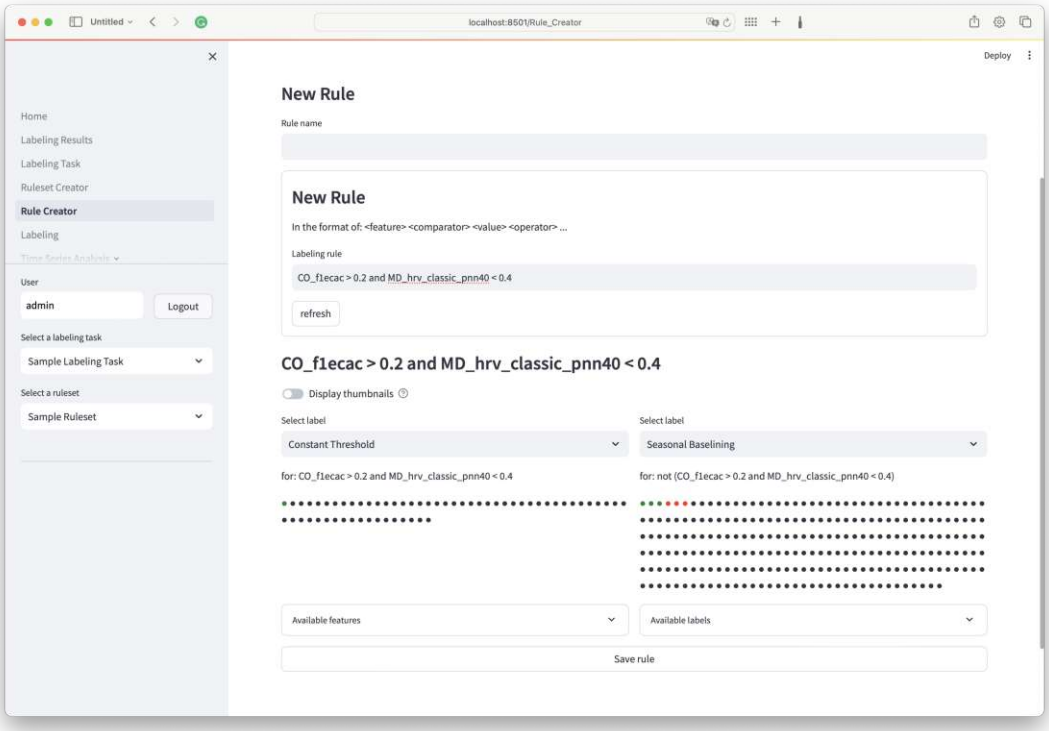


Figure 4.11: Advanced Rule Creator

needs and requirements while refreshing every change to see if the changes are what is desired.

Figure 4.13 shows the time series thumbnails and rule list for three labels for the currently selected sample labeling task and the sample labeling function of Figure 4.12. It also applies here that matching labels are green and incorrect labels are red. The `ABSTAIN` label will always take the rightmost column and can never be green, since this is not a valid label for a time series.

Labeling

The Labeling page, in Figure 4.14, is a main part of the application and the place where the models live. It consists of four parts, namely, the labeling properties, in Figure 4.15, the labeling results data frame, the labeling function analysis, in Figure 4.16, and the surrogate model analysis, in Figure 4.17. Once a user navigates to this page a first preliminary labeling step will occur creating a baseline for the user and allowing a familiarization with the result metrics.

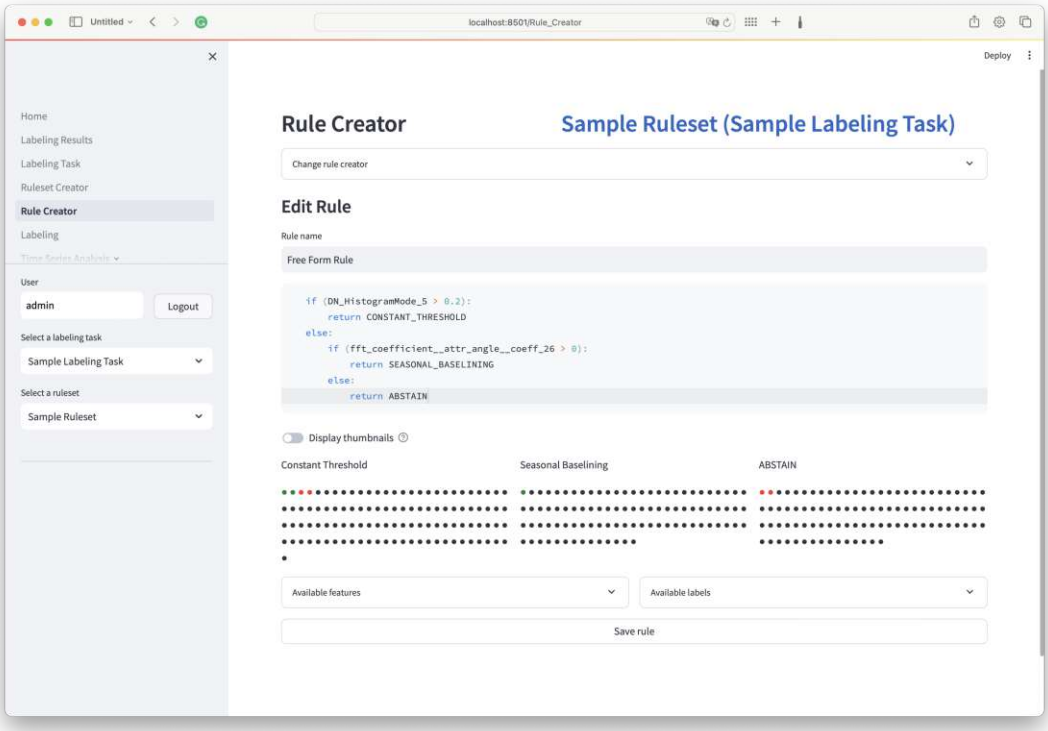


Figure 4.12: Free From Rule Creator

Labeling Properties The Labeling Properties expander, in Figure 4.15, is the main input component for the labeling workflow. This is the place where all configuration takes place. Due to the nature of Streamlit rerunning the script on every change, every change to the properties will result in a recalculation of the label model. The changes will be displayed in the three sections below. The Labeling Properties expander is split into the same sections that our proposed workflow is, namely the Snorkel Label Model, here called Snorkel Model, and the Surrogate Model. These parts change according to the input possibilities of the selected models. The possible model input parameters are listed below.

List of the model parameters for the Snorkel Label Model:

- **Snorkel Model**: The snorkel model is used for evaluating the labeling rules. There are two models available, the label model and the majority model. More information about the models can be found in Section 4.2.1.
- **Epochs**: The number of epochs to train (where each epoch is a single optimization step).
- **Seed**: A random seed to initialize the random number generator with.

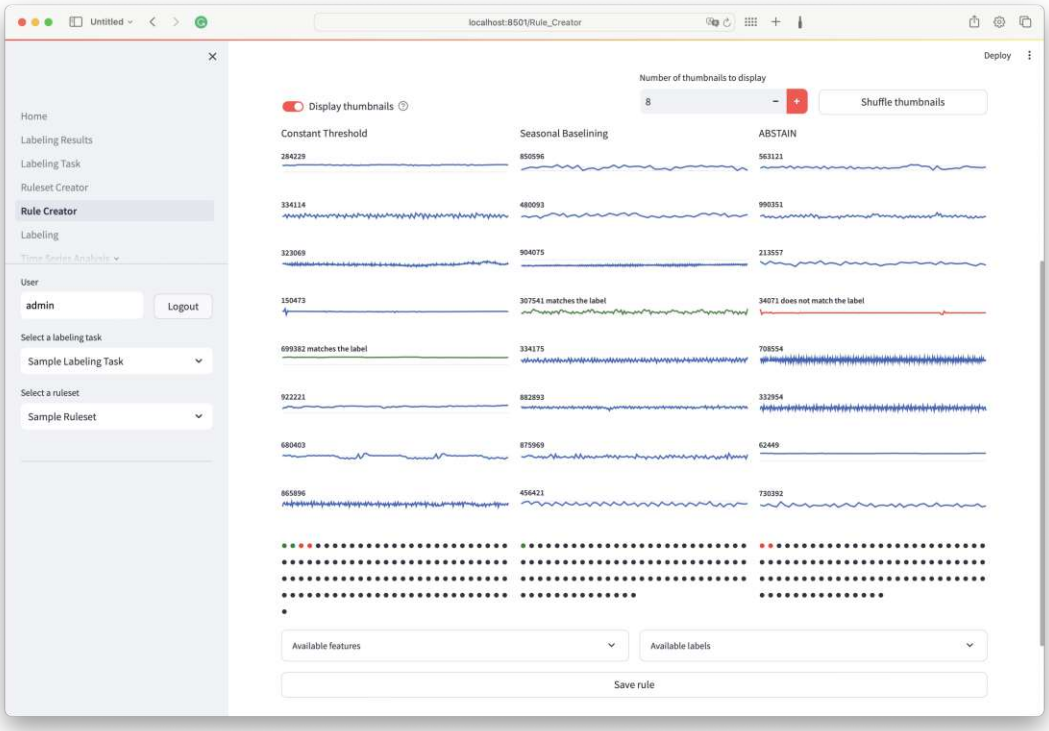


Figure 4.13: Time Series View of Free Form Rule Creator

- **Tie Break Policy**: What the model does in case of a tie. Either abstain, random, or true-random. True-random is not encouraged since it makes results less reproducible.
- **Only use selected labeling functions**: If the ruleset contains many rules, there is the option to exclude some.

List of model parameters for the Surrogate Model:

- **Surrogate Model**: Selection between the four available surrogate models from Section 4.2.1: A scikit-learn Logistic Regression, a PyTorch Lightning driven logistic regression, a PyTorch Lightning driven probabilistic logistic regression, and the WeaSEL model using a multilayer perceptron (MLP) Net as end-model.
- **Restart Surrogate Model Training**: In case the input parameters do not change the model will not be retrained. If you want to retrain for any reason, click this button.
- **Use Gold Labels (where available)**: This replaces the labels from the Label Model with the available Gold Labels for training the Surrogate Model.

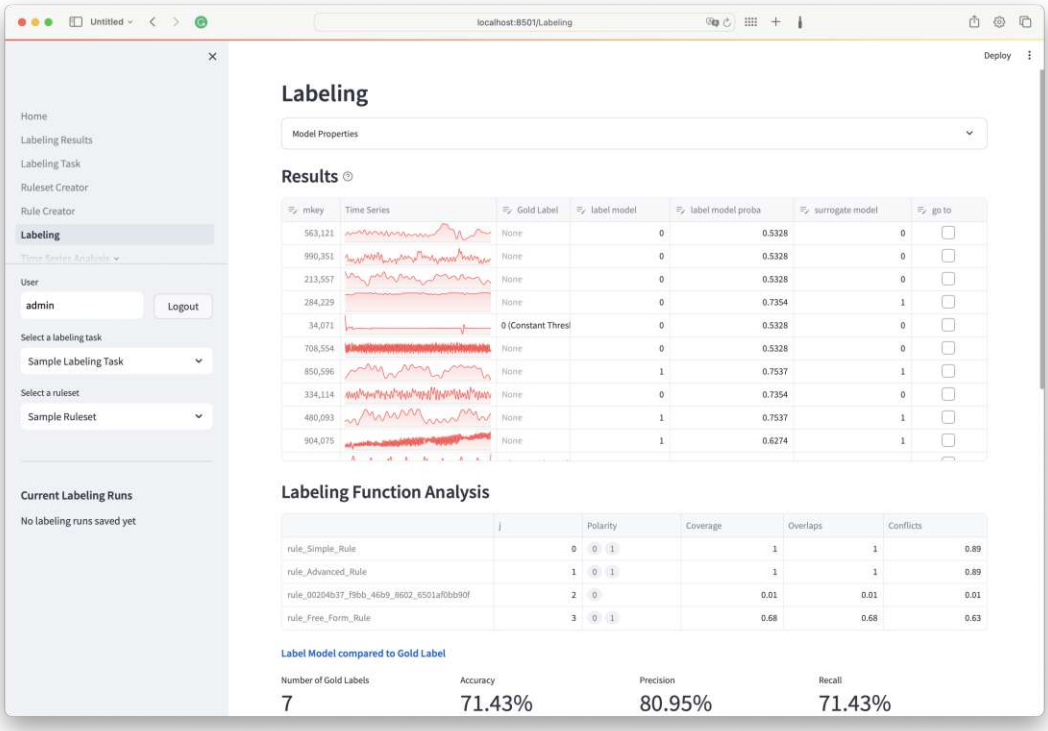


Figure 4.14: Labeling View

- **Use only the same features as the Snorkel Model**: The feature matrices contain many features. Not all of them are representative of the task you want to achieve, therefore, the pre-selection is true. If more features are needed that are not already included via the labeling functions they can be added in the additional features multi-select below.
- **C and Solver**: These are scikit-learn parameters for the logistic regression. More information can be found in the scikit-learn documentation[16].
- **Number of Stratified K-Folds**: Sets the K, the number of folds, for an evaluation metric. Has no real influence on training and solely allows to gauge the generalization capabilities.

Labeling Function Analysis As the name suggests the labeling function analysis 4.16 shows data that is related to the outcome of the labeling model. The first table is a labeling function analysis that Snorkel provides. For each rule we get the polarity,

[16] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html/

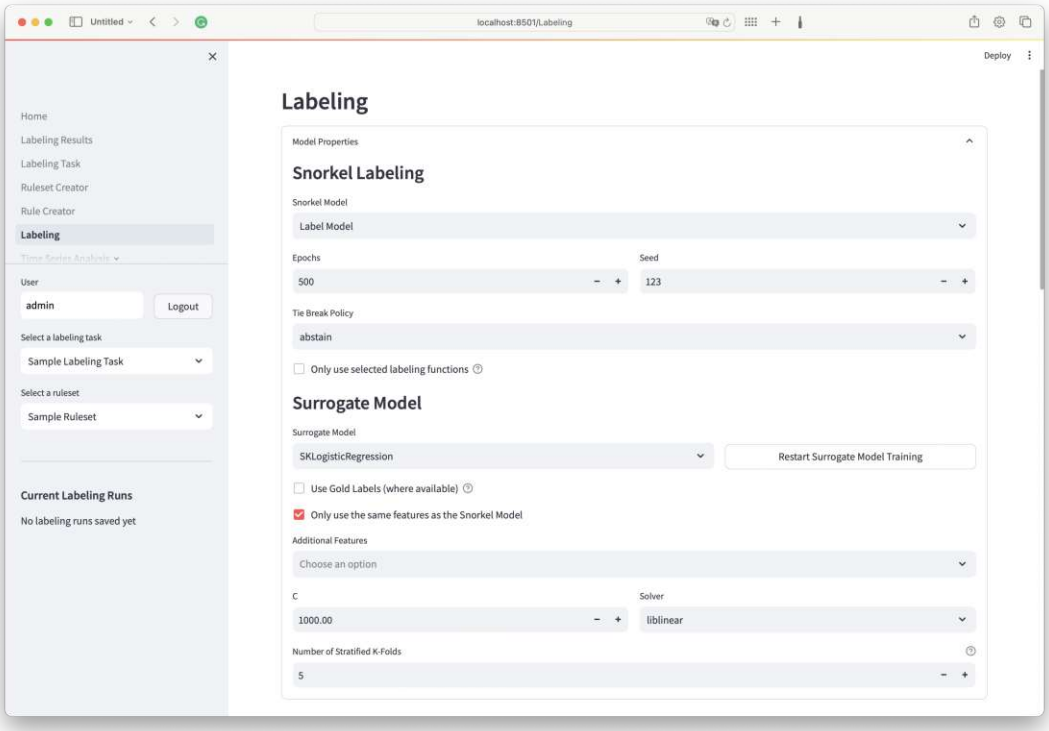


Figure 4.15: Labeling Properties

meaning which are possible labels the rule applied, based on the evidence of the created labeling matrix. Furthermore, there are also coverage, overlaps and conflicts outlined. All values are between 0 and 1, where one is 100% of all time series. The coverage describes how much of the dataset was covered by a rule. The overlaps value is a fraction of the amount of time series with more than one label and the conflicts is the fraction of data with more than one different label.

The next metric row displays the label model accuracy compared to the manually added gold labels with the previously established metrics of accuracy, precision, and recall. This should allow users to gain a first insight into the performance of their labeling functions in regard to what they have set as ground truth without the application of a surrogate model. This can be especially interesting in contrast to the gold label scores from the surrogate model analysis allowing users to quickly glance if applying the surrogate model brings an improvement.

The label model properties enable the exclusion of rules, while rules can include many different features. In the following sections, the features used for the label model are listed, followed by an expander in case the labeling rules require inspection.

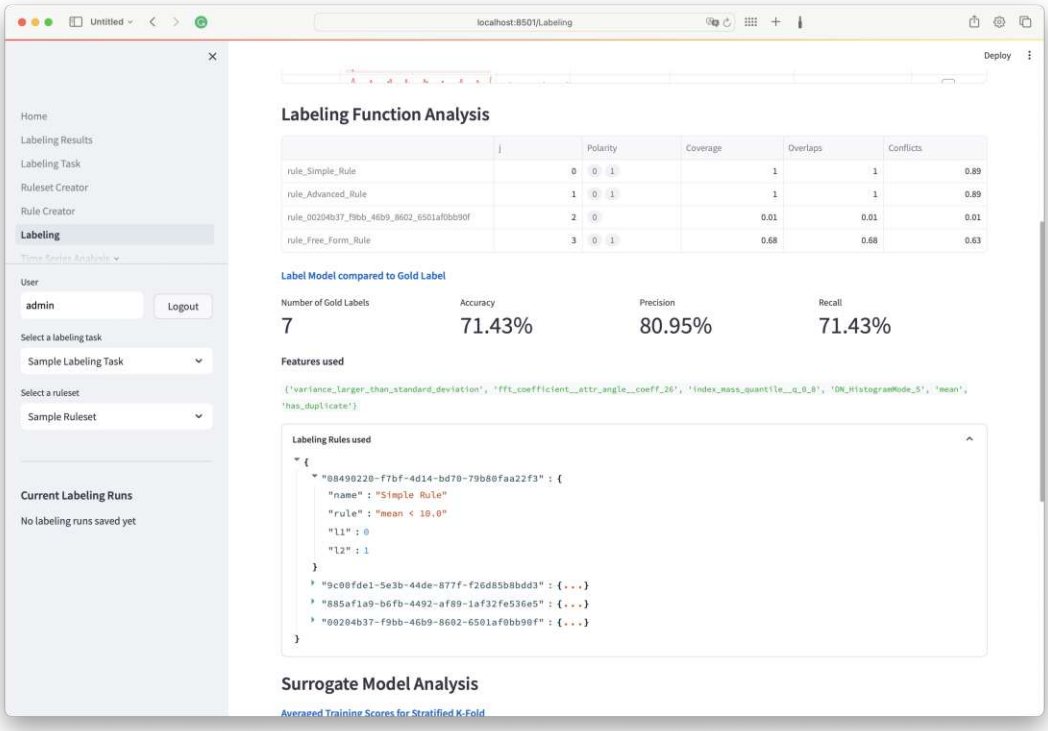


Figure 4.16: Labeling Function Analysis

Surrogate Model Analysis The Surrogate Model Analysis, in Figure 4.17, provides users with metrics in regard to the surrogate model performance. There are five sections providing in-depth model metrics. Since we are interested in the model's generalization capabilities the stratified K-fold from the properties is applied during prediction, the scores are extracted and provided to the user to compare to the other metrics. The gold label scores can be considered the most important scores, considering there is a substantial amount of labeled data already available. The users want to maximize these scores and compare them directly to the label function analysis metrics to see if the scores improved with the chosen model. Furthermore, there are the label model scores in the surrogate model analysis, these are predicted when the data is trained on the whole set as recommended by Zhou [25]. In addition to these scores, there is also the feature matrix available in an expander at the bottom of the page. Lastly, the surrogate model also displays the features used, since it is possible to apply a bigger feature matrix to the surrogate model in case there are significant features available that have not been or could not be translated to labeling functions.

Time Series Analysis The Time Series Analysis page, in Figure 4.5, is the last component of the application allowing users to view and analyze individual time series.

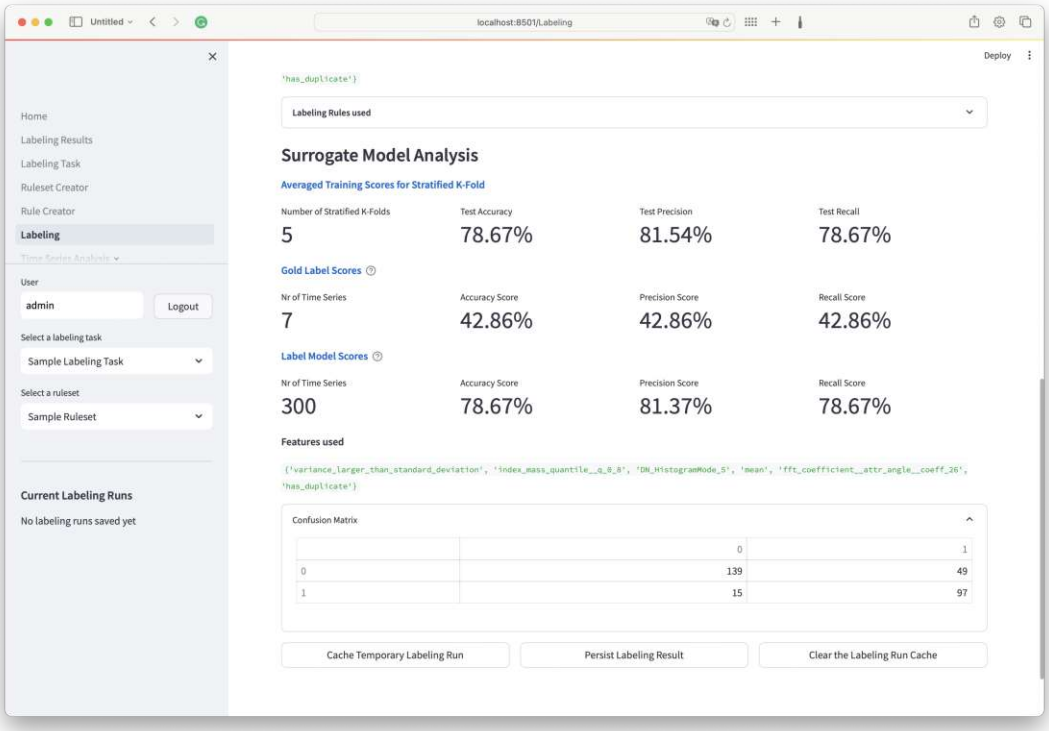


Figure 4.17: Surrogate Model Analysis

There are two main pathways to this page, each being a 'go to' button inside a data frame, either on the labeling results page or on the labeling page. Depending on the path taken, the bottom section displays different information. This is also the only page that includes a full, non-scaled-down, version of the time series graph since loading the data can cause performance issues. Above the chart, there is a select box allowing a user to set or change the gold label of a time series. Below the chart, there are all available features for the time series and the respective values. This should allow users to look up specific values for a time series. Depending on the path taken, the bottom of the page displays either the labels for the different labeling runs 4.19a or the labels for the different label models 4.19b.

4.2.4 Implementation Improvements

It has been observed that the Streamlit architecture, which was initially useful in getting the application up and running quickly, is now causing hindrances in terms of performance. Additionally, managing the cache in the background has become a challenging task in itself. There are several known issues that arise due to difficulties in keeping the data in sync, which further impacts the overall efficiency of the application. In order to optimize the performance of the system in the future, it would be advisable to separate the frontend

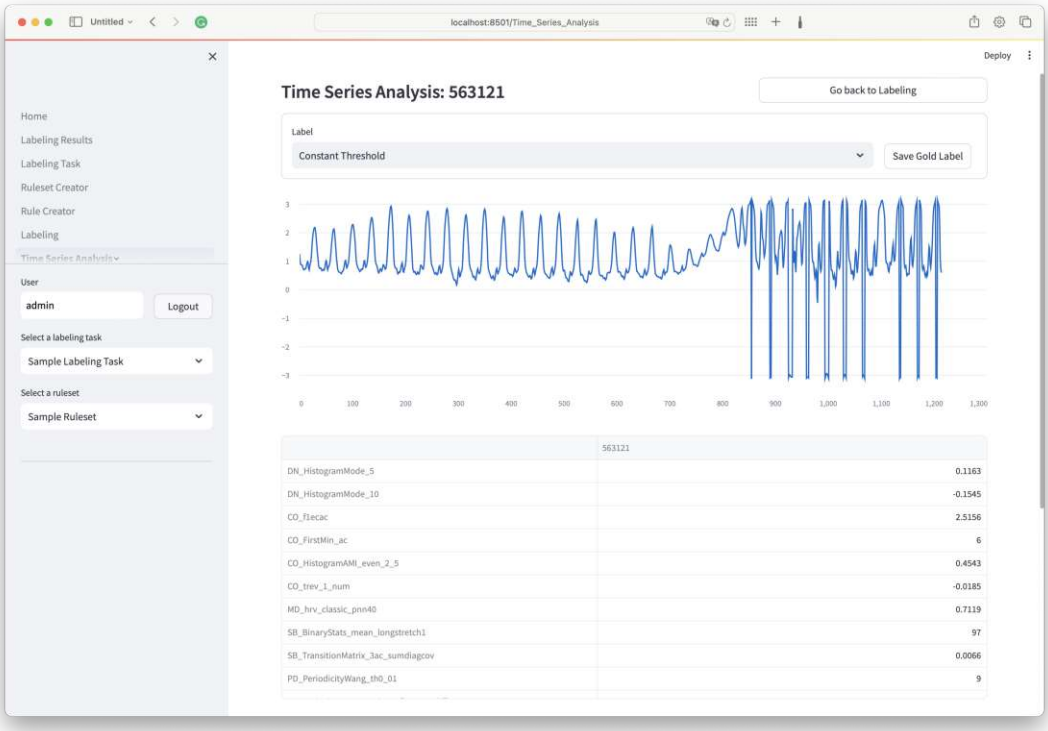
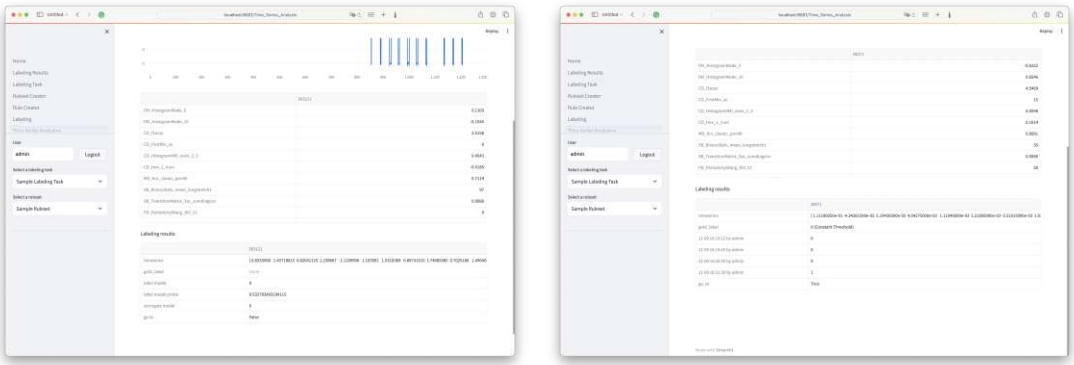


Figure 4.18: Time Series Analysis

completely from the backend. This would involve utilizing a RESTful architecture which would enable better management of communication between the frontend and backend. Additionally, this approach would allow multiple users to interact with the system



(a) Time Series Analysis from Labeling (b) Time Series Analysis from Labeling Result

Figure 4.19: Bottom section of Time Series Analysis

simultaneously, thereby enhancing the overall user experience. By implementing these changes, the system would become more efficient, flexible, and scalable.

As a result of building the database structure based on a relational database with capabilities for semi-structured data, the tables have not been optimized for a solely relational setup resulting in a performance loss.

Switching away from the Streamlit approach, it could have been possible to run the label models at different times in the background to provide label model insights during the rule and ruleset creation process. This would shift some of the computational workload to an earlier stage in the labeling process and would provide earlier feedback for users. Furthermore, this would allow a deferred execution in case more than one change to a parameter is desired without a need to wait for the previous calculation to finish.

In general, it would seem interesting to allow users to create their own features during the rule creation process. However, depending on the feature and time series data set this might not be feasible for complex calculations.

CHAPTER 5

Evaluation

The following chapter consists of two main parts. In the methodology section, we outline the study. The section describes the experiment, the study participants, the procedure, the datasets, and the survey. Finally, we present the results of the study and analyze them in-depth. The discussion of the results follows in Chapter 6.

5.1 Methodology

We have opted for a mixed-methods approach to facilitate answering the posed research questions in our preliminary research. Our analysis of RQs 1 and 3 is based on the results obtained from the quantitative phase of our study, whereas the insights for RQ 2 are derived from the qualitative data collected. We implemented an experimental design for the quantitative segment. We designed the experiment to assess the effectiveness of weak labels and end models compared to ground truth labels and those obtained through supervised methods. We aimed to observe, measure, and analyze how these models perform within a weak supervision context, focusing on model performance. On the qualitative side, we engaged in a comprehensive survey targeting experts in the field. Through thematic analysis of the responses collected, we aimed to find patterns, themes, and insights. These qualitative findings allow interpretation and provide additional depth to our understanding of the experimental results, thereby enriching our comprehension of the research topic.

5.1.1 Experiment

We aim to answer the posed quantitative research questions with the following real-world scenario in mind: During load forecasting for a server metric, a forecaster has to be selected. For this selection, we have to analyze the time series up to the forecast point and decide which forecasting model to choose. This is an important decision since more

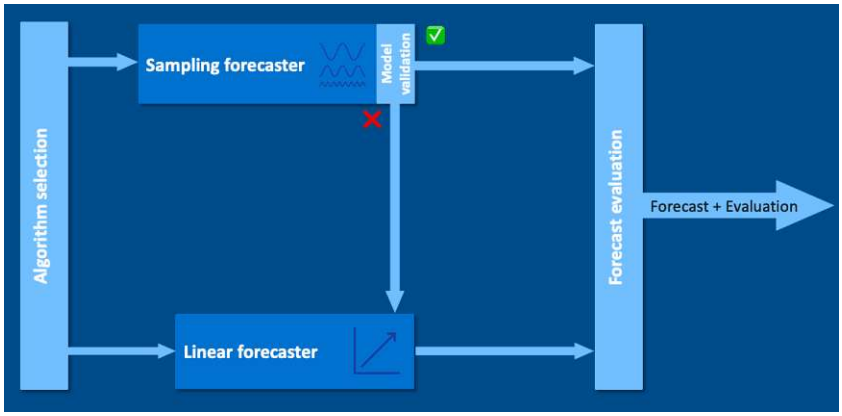


Figure 5.1: Forecasting Model Selection[17]

sophisticated models, such as seasonal forecasting models incur much higher costs than, for example, linear forecasting models. Figure 5.1 shows such a flowchart for model selection.

To aid the algorithm selection process we need a way to classify the time series data accordingly. One way to achieve this classification is by training a ML model with accurately labeled data so that the resulting model can in turn classify new unseen time series. As established in earlier sections, acquiring labeled data is expensive. To simplify and reduce the cost of this process, we have designed a labeling workflow and implemented a labeling demo application. The application is used to label time series data by applying labeling rules and using a surrogate model to further generalize the input-output mapping.

Therefore, to generate evaluation data for the scenario, labeling rules using either the *Seasonal Baselining* or *Constant Threshold* labels [94] are required to be created and applied to a set of time series data.

Participants

For our experiment, we have specifically targeted individuals with substantial domain knowledge, prioritizing professionals in data and computer science fields. This includes researchers with a solid foundation in mathematics, statistics, and computer science, who are adept at navigating the intricacies involved in our study. Our selection criteria focus on individuals with a blend of theoretical understanding and practical expertise in these disciplines. Therefore, we have specifically asked members of a data science team at Dynatrace, a company that utilizes the aforementioned forecasting systems. Furthermore, we asked software engineering researchers at TU Vienna to participate in the experiment. We have decided to use convenient sampling to decrease the amount of time until we can

[17] `https://docs.dynatrace.com/docs/platform/davis-ai/on-demand-analysis/forecast-analysis/`, accessed 05.12.2023

evaluate the results. Since this is preliminary research we have opted to choose a sample size of five participants.

Procedure

The experiment setting involves the setup of the demo outlined in Chapter 4. For local usage, participants can either use the provided Docker files or the manual installation. For an easy collaborative experience, a setup of the demo on a remote server is preferred. Once the participants launch the prototype they are confronted with additional information and a sample task to familiarize themselves with the tool, task, and scenario. During the experiment, every participant is assigned either an individual or a collaborative labeling task. Both labeling tasks require the participants to generate labels for all time series outlined in the training data set later in this section. Participants are able to do this by generating at least three labeling functions and then by refining or changing models, parameters, and labeling functions. When a participant is sufficiently happy with a labeling result the result can be persisted for the task.

The labeling performance is measured with the metrics outlined in Section 2.1. These metrics are calculated for the weak labels as well as the gold labels, using k-fold cross-validation. Initially, the participants do not have any gold labels available, but they are free to add as many as they like to improve the assessment of their model's performance. Participants can then reiterate through the process of creating and adapting their labeling rulesets until content.

Once a participant is sufficiently content with their labeling performance on the dataset the resulting database is sent back for evaluation and the survey is conducted. After receiving all labeling results, we will conduct an extended analysis of the results, the surrogate models, and their labeling performance before and after hyperparameter tuning.

Datasets

To prepare the data for evaluation, the tools catch22 [63] and tsfresh [95] were used to extract a total of 793 features. However, for tsfresh, only the `EfficientFCParameters` setting was applied, which calculates features that can be extracted without a high computational cost. This approach produced 771 usable features for evaluation, meaning that the resulting feature is not NaN. The feature calculation process took 15 hours and 47 minutes, whereas the pycatch22 [96] implementation of the catch22 features only took just under 5 minutes using an Apple M1 Max processor with 64GB of Memory.

The available datasets have been divided into two parts — the training dataset and the testing dataset. The training dataset has not been used or seen by the people involved in this particular use case. On the other hand, the testing dataset is familiar to some of the participants, but it has different features extracted from it. To prevent any assumptions based on the names of the time series, the series names have been encoded with a 6-digit metric key.

Dataset	Type	Count	Avg Observations	Avg NaNs
Condition Monitoring [97]	Training	48	3,684	0
DJIA 30 Stock [98]	Training	155	3,019	0
MotionSense Dataset [99]	Training	4,320	3,928	0
Total Training Data	**Training**	**4,523**	**3,894**	**0**
Initial	Initial	303	58,735	18,772
Eval DS1	Evaluation	303	58,735	0
Eval DS2 (Lin)	Evaluation	303	77,507	0
Eval DS3 (Split)	Evaluation	324	37,365	0
Eval DS4 (Split Lin)	Evaluation	324	40,874	0
Eval DS5 (Split Cub)	Evaluation	324	40,874	0

Table 5.1: Dataset overview

Training Datasets Upon conducting a closer inspection of various datasets, we have selected three specific sets that provide a mix of seemingly (visually) seasonal and non-seasonal data. These datasets were then split into 4514 individual time series, which represent the training time series dataset for our use case. Although the dataset may not be ideal for training, it provides a scenario that is much closer to reality than a fully balanced set. On average, each time series has 3894 observations. There were no NaN values present because these datasets were already prepared in advance. Furthermore, compared to the test set, these time series are approximately ten percent of the length.

Condition Monitoring Dataset [97, 100]: The dataset is derived from a specially configured AC induction motor that uses a motor capacitor to work on 230 V, 50 Hz, single-phase AC. Instead of its original fan, the motor uses various 3D printed fans, some similar to the original and others modified, like fans with missing blades. The 3D acceleration data is then collected from a sensor and microcontroller at an interval of 3 milliseconds. It has already been harmonized to regular time intervals at a sampling rate of 300 Hz using cubic spline interpolation, which means that no further processing was necessary. The set added 48 time series with a high amplitude and a high frequency.

DJIA 30 Stock Time Series [98]: The dataset consists of the 30 Dow Jones Industrial Average (DJIA) stock prices between the first trading day of 2006 and the last trading day of 2018. In addition to the daily closing price, it also consists of the opening price, the highest and lowest price, as well as the traded volume for each day. The dataset used for the analysis covers a period of 12 years and includes only data from weekdays since stock markets are officially open from Monday to Friday. Hence, the dataset contains only 5 data observations per week and series, resulting in fewer data points than the total number of days in the 12-year period. The missing days have not been filled to avoid prioritizing values occurring on Fridays. Apart from this, the data has no missing values. To generate 150 series trending upwards and 5 for the index, the multivariate time series

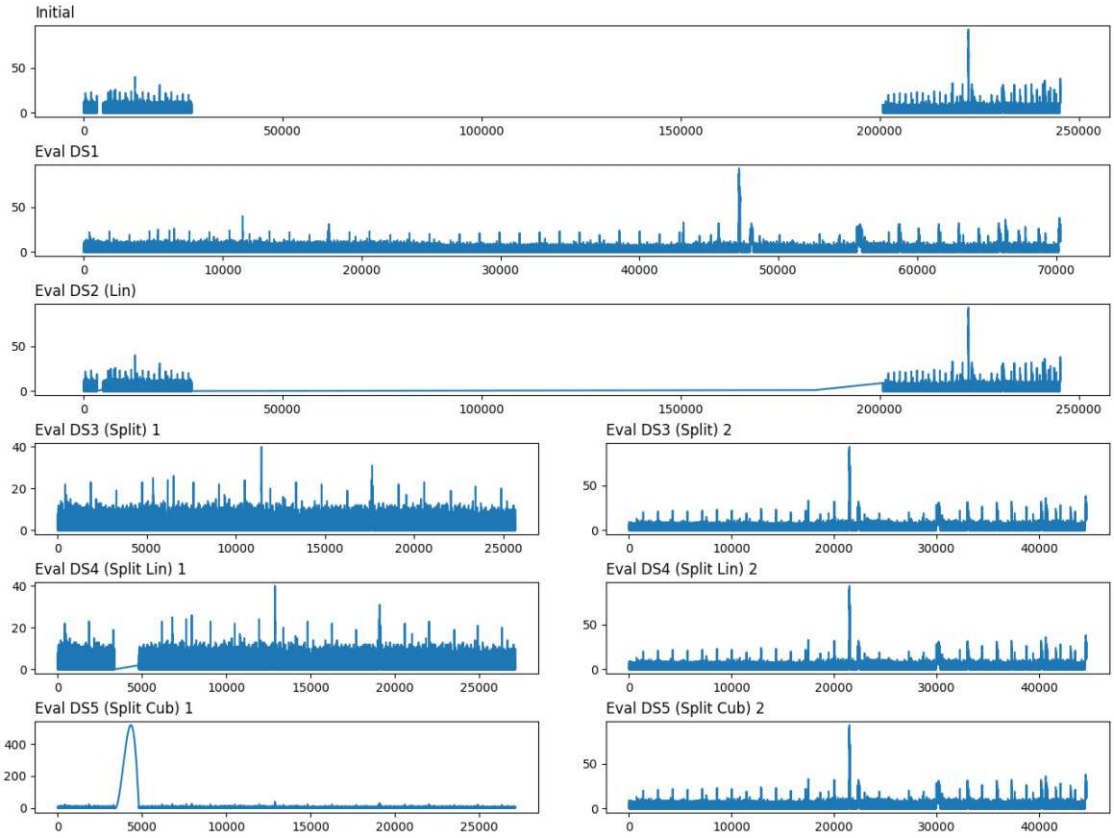


Figure 5.2: Evaluation datasets visualized

have been split into five univariate series per stock, and the index respectively.

MotionSense Dataset: Smartphone Sensor Data - HAR [99, 101]: The dataset is comprised of sensor-based time-series data, generated using the accelerometer and gyroscope sensors of an iPhone 6s. This data, capturing a range of parameters like attitude, gravity, user acceleration, and rotation rate, was gathered through an iPhone app utilizing the SensingKit framework, designed to access the Core Motion capabilities of iOS devices. The dataset involved 24 participants, each conducting a series of 15 trials that included six different activities: walking downstairs and upstairs, walking, jogging, sitting, and standing. These activities were performed under uniform conditions, with the participants carrying the iPhone in their front trouser pocket. The data is split into long sets, lasting 2-3 minutes, and short sets, lasting 30 seconds to 1 minute. Each participant's trial generated multivariate time-series data, with 12 distinct features. These features are then split, as in the above dataset, to create univariate time series. The data from the MotionSense dataset makes up the bulk of training data with a sum of 4312 usable time series.

Evaluation Datasets The initial dataset consists of 303 time series that have been manually labeled. These series are obtained by monitoring and collecting server metrics in one-minute intervals and have an average length of 58.735 data points. Out of these data points, an average of 18.772 are NaN, which means that there are 39.963 observations on average. The valid data points are found in 14-day clusters between July 2020 and the beginning of January 2021. The complete dataset is available in the project repository. The initial dataset is used to extract the evaluation sets, which are used for performance evaluation.

The main difficulty of the evaluation is the extraction of high-quality features from this dataset. There is an initial set of features available for these time series. However, the initial feature matrix used for producing the gold labels was extracted using proprietary tools and the features cannot be reproduced for the training time series sets.

To mitigate the effect of the NaN values the data has been prepared in five different ways, resulting in five evaluation data sets. The first data set (Eval DS1) ignores the NaN values and extracts the features from this set. The second data set (Eval DS2 (Lin)) linearly interpolates the NaN values of the first dataset. The third set (Eval DS3 (Split)) splits a time series if there is a day missing between observations and ignores a subseries if there are less than 100 observations in it. All other NaN values are ignored. The fourth set (Eval DS4 (Split Lin)) builds on the third set and linearly interpolates the missing values while the fifth set (Eval DS5 (Split Cub)) uses a cubic interpolation algorithm for the missing values. Splitting the time series for sets 3-5 results increases the amount of time series by 21. The split series received the same label the initial series had, resulting in an increased weight for the split series in the evaluation.

The demo application does not contain the evaluation data sets and labels to avoid influencing the participants.

Hyperparameter Tuning

In order to possibly enhance the model performance we will conduct hyperparameter tuning runs [102]. Hyperparameters are the external configurations of the model and cannot be learned from data directly. They influence the learning process and the structure of the model itself [103]. Unlike model parameters learned during training, hyperparameters include the regularization parameter C or the selected kernel, considering SVMs. Tuning these parameters is critical because they have a profound impact on the behavior of the trained model. Optimal hyperparameter settings can help avoid overfitting while also improving the model's ability to generalize from training data to unseen data. This process often involves searching through a predefined space of possible hyperparameter values, evaluating the performance of the model for each combination, and selecting the set that yields the best results, measured against a validation set. To facilitate this process we have decided to utilize the `ParameterGrid` functionality provided by scikit-learn (SKL). Hyperparameters can vary drastically between datasets

and problems, making hyperparameter a valid but costly way to improve the model performance.

5.1.2 Survey

To answer the qualitative RQ about the labeling workflow design we have designed the survey to be a case-control study according to the six-part series "Principles of Survey Design" by Kitchenham and Pfleeger [104, 105, 106, 107, 108, 109] and follows the principles provided. Furthermore, a focus was put on avoiding common survey pitfalls outlined in the article of Ghazi et al [110]. In regards to this study, it is important to note that there may be certain problems that cannot be prevented, namely the limitation of having a relatively small sample size. Nevertheless, the survey is intended to provide context for the evaluation results, and as such, it remains a crucial component of this study.

The target population of this survey is the same as that of the experiments. Reiterating, the survey and experiment target experts in the domain of time series data labeling, particularly those who possess a strong understanding of categorizing seasonal data, and who are adept at software engineering.

Given the preliminary nature and the small scale of this study, every experiment participant will be asked to answer the survey questions, meaning that the same convenient sample of five people will be used. In other words, every experiment participant will be given an opportunity to provide feedback and insights.

Since this is a small-scale experiment, we want to ensure a high response rate. To achieve this the experiment participants will be reminded of the survey in multiple locations, and especially personally after completing the experiment and submitting their experiment data.

The survey is composed of 49 questions that are divided into eight sections. These questions cover a wide range of topics, from general demographic information to more specific inquiries about the tool and its use case, and provide valuable insights for evaluation purposes.

The following are the main sections of the survey:

1. **Demographic Information:** This section gathers information about the survey participants and their backgrounds allowing for better interpretation of the results.

2. **Data Labeling:** This section aims to evaluate the participants' data labeling expertise, to provide some context about the outcome of their labeling functions regarding the experiment.

3. **Time Series Data:** This section assesses the participants' experience in dealing with time series data and their general understanding of it.

4. **First Impressions:** This section focuses on the look and feel of the demo tool and provides valuable insight into the initial exposure and complexity.

5. **Labeling Scenario:** This section evaluates the participants' domain knowledge and ability before focusing on the outcome of their labeling results.

6. **Usage:** This section delves deeper into how the label maker was used, its performance, and possible improvements.

7. **Rule Creation:** This section specifically evaluates which of the rule creators was preferred and why, providing insight into the tool usage.

8. **Final Section:** This section is used to gain an overall understanding of the participants' expectations and future outlook.

5.2 Results

Five individuals participated in the experiment. The aim of the experiment is to assess the performance of the labeling approach across different labeling tasks and to analyze the effectiveness of the models.

We have received three completed labeling tasks, the Individual Labeling Task 1 (ILT1), Individual Labeling Task 2 (ILT2), and Collaborative Labeling Task (CLT). The ILT1 and ILT2 are individual labeling tasks and were performed by one individual data scientist respectively. The CLT is a collaborative task and was completed by three collaborating data scientists. The participants conducted varying amounts of labeling runs per task.

Additionally, one individual labeling task was taken on by a computer science researcher who did not have prior experience in time series data labeling. Unfortunately, the results of this task were not submitted for evaluation due to a non-satisfactory performance.

In this section, we are differentiating between four types of labels:

- **Weak labels**, or weakly generated Labels, are labels that are generated by a Snorkel label model and used to train our surrogate models.
- **Predicted labels**, or in some cases called surrogate labels, are labels that are predicted by a ML model trained on any labels, or weak labels respectively.
- **Gold labels**, or ground truth labels, are labels that have been manually labeled during the experiment by the participants.
- **Evaluation labels** are the ground truth labels from our evaluation set. We distinguish specifically between evaluation labels and gold labels to emphasize the different datasets. Furthermore, the evaluation labels have been verified and error-corrected using a majority vote.

Overall, eight labeling runs in three different labeling tasks were submitted by the participants. All of the labeling runs were performed on the Snorkel Label Model and the SKL Log Reg model with the standard parameters. Table 5.2 displays the performance of the surrogate model compared to the weak labels generated by the label model (LM), as well as compared to the manually set gold labels. The metrics are always calculated in relation to their count. From the results, we can see that the gold label account increased in the CLT, meaning that additional gold labels were added between labeling runs. The average performance of the metrics precision, recall, and F1 does not look very promising. However, in many cases this is a result of an inability of the surrogate model's ability to model the input. Thus, it returned only the *0*, or *Constant Threshold*, label. More precisely:

- The ILT1 surrogate model could not predict anything.
- The ILT2 surrogate model could only predict something in run 2.
- The CLT surrogate model could only predict something in runs 4 and 5.

In total, 229 time series have been hand-labeled. 28 have been labeled by two and 10 by three users. Five, respectively two, have received ambiguous labels, resulting in 174 unambiguous labels. Furthermore, this results in 7 time series being labeled differently in various labeling tasks by different users. This means that 18% of time series that have been labeled more than once are labeled differently. Taking all labeled data points into consideration the participants agree in 96% of cases. An example of 2 time series that have received both labels can be seen in Figure 5.3. Examples for the labels *Seasonal Baselining* and *Constant Threshold* can be seen in Figure 5.4 and Figure 5.5 respectively.

During all labeling runs, a total of 26 different labeling rules have been applied to the data. These rules can be inspected in Listing 8.1. A full overview of their polarity, coverage, and conflicts on the labeling task can be found in the Appendix 8 in the Table 8.1, 8.2, and 8.3.

Run	LT	TS Count	LM Precision	LM Recall	LM F1	GL Count	GL Precision	GL Recall	GL F1
1	CLT	4523	0.2604	0.5103	0.3448	24	0.25	0.5	0.3333
2	CLT	4523	0.2604	0.5103	0.3448	44	0.3492	0.5909	0.4390
3	CLT	4523	0.2604	0.5103	0.3448	60	0.3403	**0.5833**	0.4298
4	CLT	4523	0.5185	0.5178	0.5181	87	**0.5853**	0.5747	**0.5800**
5	CLT	4523	0.5185	0.5178	0.5181	128	0.5458	0.5391	0.5424
1	ILT1	4523	0.1302	0.3608	0.1913	50	0.3364	0.58	0.4258
1	ILT2	4523	0.5661	0.7524	0.6461	51	0.2599	0.5098	0.3443
2	ILT2	4523	**0.7801**	**0.7813**	**0.7807**	51	0.2369	0.3137	0.2699
	Mean		0.4118	0.5576	0.4611		0.3630	0.5239	0.4206
	Min		0.1302	0.3608	0.1913		0.2369	0.3137	0.2699
	Max		0.7801	0.7813	0.7807		0.5853	0.5909	0.5800

Table 5.2: Labeling results table with the maximum results highlighted.

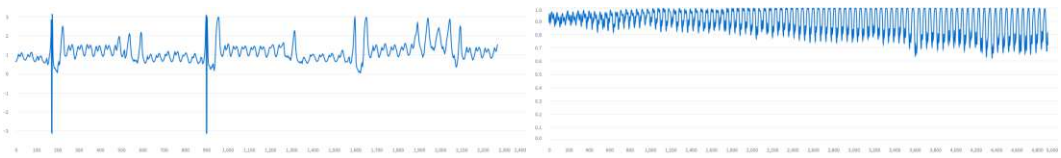


Figure 5.3: Time series with ambiguous labels

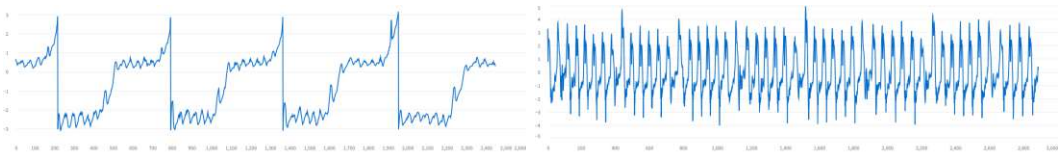


Figure 5.4: Time series labeled *Seasonal Baselining*

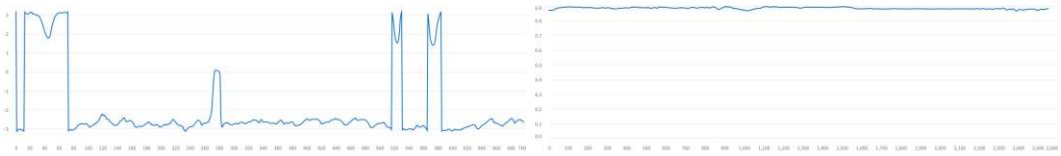


Figure 5.5: Time series labeled *Constant Threshold*

The labeling rules made use of 23 of the 793 features. A description of the features can also be found in the Appendix 8. They used a total of 23 different features. 13 of those were extracted from tsfresh and 10 from pycatch22.

5.2.1 Analysis

Upon the initial evaluation of the labeling results, it can be observed that their performance falls short of the specified target of 70% compared to the gold labels. In order to carry out a more in-depth analysis, the following measures have been taken. Firstly, additional machine learning models are introduced. Furthermore, we will evaluate which labeling functions are the most significant and have the biggest influence on the scores.

Then, an extra benchmark is added by implementing standard supervised learning on the gold labels to compare the resulting outcomes with the existing models. All incorporated models are trained and their results are evaluated against the gold label sets and the evaluation sets. These steps have been taken to ensure that the analysis is thorough and complete and that the best possible outcome is achieved in this stage of the analysis.

Evaluation Models

To aid the comprehensive evaluation, a wider set of models is trained and evaluated for their performance on the training data. Following is a list of models and a short description.

- SKL Log Reg[18]: scikit-learn implementation of the logistic regression model. Allows for changes of the solver in regard to the optimization problem.
- PTL Log Reg: Logistic regression implementation using Pytorch Lightning as outlined in a previous section.
- PTL Prob Log Reg: Probabilistic logistic regression implementation using Pytorch Lightning as outlined in a previous section.
- WeaSEL[19]: WeaSEL model implementation using a Pytorch multi-layer perceptron network as end-model.
- scikit-learn Support Vector Classifier (SKL SVC)[20]: scikit-learn implementation of the support vector classifier. Has multiple kernels to preselect for optimal performance.
- scikit-learn Gaussian Naive Bayes (SKL GNB)[21]: This is a Gaussian implementation of the Naive Bayes classifier by scikit-learn, allowing for classification of continuous data.
- scikit-learn Random Forest Classifier (SKL RFC)[22]: A random forest classifier implementation by scikit-learn allowing for changes in the measurement of the split quality.
- scikit-learn Gradient Boosting Classifier (SKL GBC)[23]: Tree-based sci-kit learn implementation of a gradient boosting classifier.
- scikit-learn K Neighbors Classifier (SKL KNN)[24]: scikit-learn implementation of the K-nearest neighbor classification model offering various algorithms for distance calculation.

Labeling Function Performance

We conducted an extensive analysis of labeling function performance by executing an exhaustive search of the search space. To achieve this, we applied every possible combination of labeling functions that had at least three rules per labeling task to our label model. Then the labeling function performance was evaluated based on their

[18] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html/
[19] https://github.com/autonlab/weasel
[20] https://scikit-learn.org/stable/modules/svm.html/
[21] https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes/
[22] https://scikit-learn.org/stable/modules/ensemble.html#forest/
[23] https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosted-trees/
[24] https://scikit-learn.org/stable/modules/neighbors.html#classification/

performance metrics on both the use case feature matrix and the evaluation set feature matrices.

$$combinations = \sum_{k=3}^{n} \binom{n}{k} \tag{5.1}$$

Equation 5.1 models the count of combinations per labeling run for a set of n labeling functions and a subset of k labeling functions.

For the CLT task, 9 rules produce 466 combinations, for ILT1, 10 rules result in 968 combinations, and for ILT2, 7 rules generate 99 possible combinations. Moreover, evaluating the combinations on both training and evaluation matrices, in total, 3066 models were trained and evaluated. As a result, we were able to extract six superior rule combinations, five of which were unique and can be viewed in Listing 5.1.

```
fs1 = ['fourier_entropy__bins_2',
       'percentage_of_reoccurring_values_to_all_values',
       'time_reversal_asymmetry_statistic__lag_1', 'number_peaks__n_50']

fs2 = ['SP_Summaries_welch_rect_centroid', 'cid_ce__normalize_True',
       'FC_LocalSimple_mean3_stderr', 'PD_PeriodicityWang_th0_01',
       'fft_aggregated__aggtype_variance']

fs3 = ['standard_deviation', 'PD_PeriodicityWang_th0_01',
       'SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1', 'CO_f1ecac']

fs4 = ['fourier_entropy__bins_3', 'time_reversal_asymmetry_statistic__lag_1',
       'percentage_of_reoccurring_values_to_all_values']

fs5 = ['PD_PeriodicityWang_th0_01', 'SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1',
       'CO_f1ecac', 'variance']
```

Listing 5.1: Feature sets extracted from best performing labeling rules.

These rule subsets per labeling task are then used to extract more representative features. A histogram of the features used in the labeling rules can be seen in Figure 5.6

Supervised Benchmark

To generate a practical and real-world benchmark as a comparison, a supervised approach was also implemented for the labeling tasks. Unlike the regular approach taken in this project that involves applying labeling functions and training the label model before using the surrogate model, this approach directly trained the ML models with the gold label inputs for each labeling task.

Furthermore, to provide a comprehensive comparison, another set of runs was included that utilized all 174 unambiguous gold labels from all labeling tasks for training with all features in each run. Moreover, a series of runs (*Best FT*) were conducted using the highest-performing labeling rule combination features.

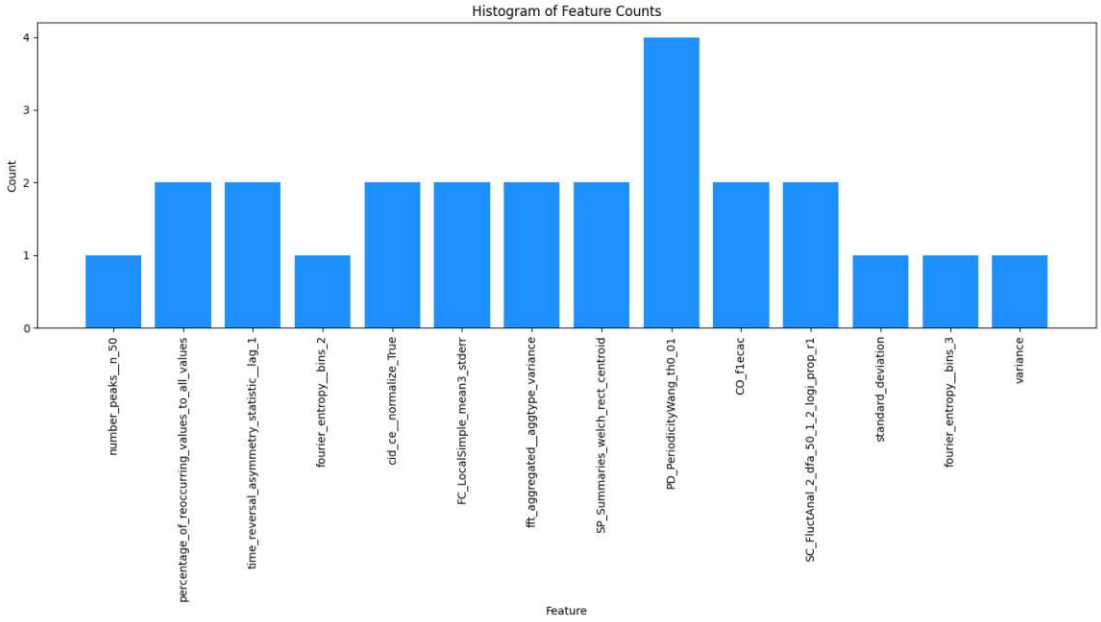


Figure 5.6: Features by occurrence in the best labeling rulesets

LT	Evaluation Set	Gold Label Count	Model	Precision	Recall	F1
CLT	Eval DS2 (Lin)	128	SKL Log Reg	0.6130	0.6105	0.6117
ILT1	Eval DS3 (Split)	50	SKL KNN	0.5930	0.5833	0.5881
ILT2	Eval DS3 (Split)	51	SKL GBC	**0.7105**	**0.6944**	**0.7024**
CLT	Eval DS3 (Split)	174	SKL KNN	0.6667	0.6450	0.6557
Best FS[25]	Eval DS1	174	SKL Log Reg	0.5902	0.5907	0.5904

Table 5.3: Supervised training results considering the training features of each labeling run

For the supervised benchmark, only the six scikit-learn models were used, resulting in a total of 240 training and evaluation runs. The peak performances for each labeling task and the peak performances of the additional runs are shown in Table 5.3. Interestingly, the ILT2 runs performed best with only 51 gold labels. However, considering all unambiguous labels, the CLT had the best result and improved its performance by >3% all metrics. Surprisingly, when using the feature sets from Section 5.2.1, the best feature set (Best FS) could only surpass the performance of the worst labeling run. These results provide a thorough baseline of the performance of the models and labeling tasks.

[25]Best FS are features extracted from the best combination of labeling rules.

	Precision	Recall	F1
count	805	805	805
mean	0.4777	0.5158	0.4873
std	0.1533	0.0718	0.118
min	0.2248	0.4191	0.3025
25%	0.4159	0.4653	0.4175
50%	0.4782	0.4884	0.4801
75%	0.5775	0.5617	0.5651
max	**0.7482**	**0.6944**	**0.7050**

Table 5.4: Resulting metrics of the evaluation of nine models per labeling result averaged over five runs.

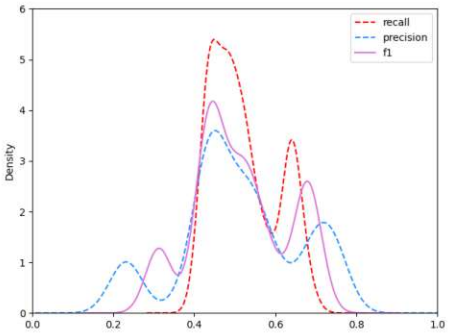


Figure 5.7: Metrics density distribution

Model Evaluations

The next step in our analysis is evaluating the labels we obtained from the three labeling tasks and comparing the results to our benchmark. To evaluate which of the used models exhibits the highest initial potential for further hyperparameter tuning another series of training and evaluation steps was conducted. Each of the machine learning models was trained on the weakly generated labels and the surrogate model labels. As stated in the first part of this section only in three of eight cases the surrogate model predicted anything but zero resulting in 490 runs. As we have seen a high variance during these runs, the models were trained five times each and the metrics were averaged over the runs. The resulting data frame can be described as seen in Table 5.4 and the metrics density distribution is shown in Figure 5.7. The maximum result on any untuned model, with basic parameters, is a confidence boost that the benchmark of 70% can be reached.

Models Trained on Surrogate Model Labels Now, we are diving deeper into the models that were trained with labels from the surrogate model, as was initially conceptualized in the idea. The result data is split by the training label type. The models considered in this case are the six scikit-learn models because we only have a 1-d label array as input. The results of the runs show that the top-performing models can come close to the specified 70% and are among the top results overall.

Model	Evaluation Set	LT	Precision	Recall	F1
SKL GNB	**Eval DS1**	CLT	0.7482	0.6666	**0.7050**
SKL GBC	**Eval DS2 (Lin)**	ILT2	0.7013	**0.6897**	0.6955
SKL SVC	Eval DS2 (Lin)	CLT	0.7428	0.6369	0.6858
SKL Log Reg	Eval DS1	ILT2	0.5577	0.5478	0.5527
SKL KNN	Eval DS3 (Split)	CLT	0.5533	0.5462	0.5497
SKL RFC	Eval DS5 (Split Cub)	ILT2	0.5532	0.5327	0.5427

Table 5.5: Model training results with surrogate labels as inputs.

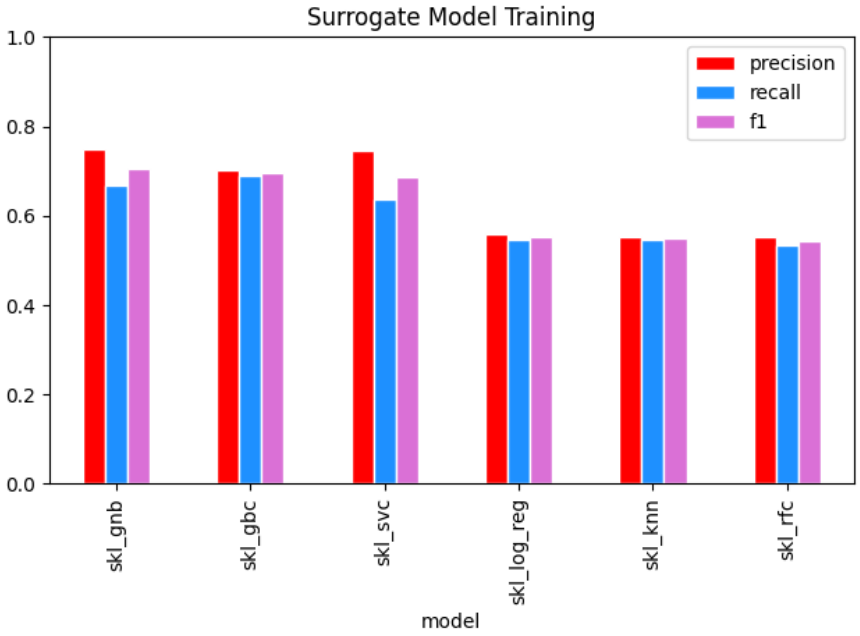


Figure 5.8: Model performances trained on *surrogate model*, sorted by the F1-score.

The key findings from Table 5.5 are:

- The SKL GBC has the highest recall.
- The SKL GNB has the highest F1-score.
- Eval DS1 and Eval DS2 are the best-performing evaluation sets.

Models Trained on Weak Labels The question we are focusing on in this part is if the performance can be improved right away without the need to fit another model before. This means that we want to see if these models perform better when used directly as surrogate models. All nine evaluation models have been trained and evaluated against

Model	Evaluation Set	LT	Precision	Recall	F1
SKL GNB	**Eval DS1**	**CLT**	**0.7482**	0.6666	**0.7050**
SKL GBC	**Eval DS2 (Lin)**	ILT2	0.7055	**0.6930**	0.6991
SKL SVC	Eval DS4 (Split Lin)	CLT	0.7455	0.6358	0.6862
PTL Log Reg	Eval DS1	CLT	0.6269	0.6145	0.6206
PTL Prob Log Reg	Eval DS1	CLT	0.6173	0.6026	0.6098
SKL Log Reg	Eval DS1	ILT2	0.5577	0.5478	0.5527
SKL KNN	Eval DS3 (Split)	CLT	0.5533	0.5462	0.5497
WeaSEL Model	Eval DS3 (Split)	CLT	0.5568	0.5290	0.5425
SKL RFC	Eval DS5 (Split Cub)	ILT2	0.5322	0.5148	0.5233

Table 5.6: Model training results with the weak labels as inputs.

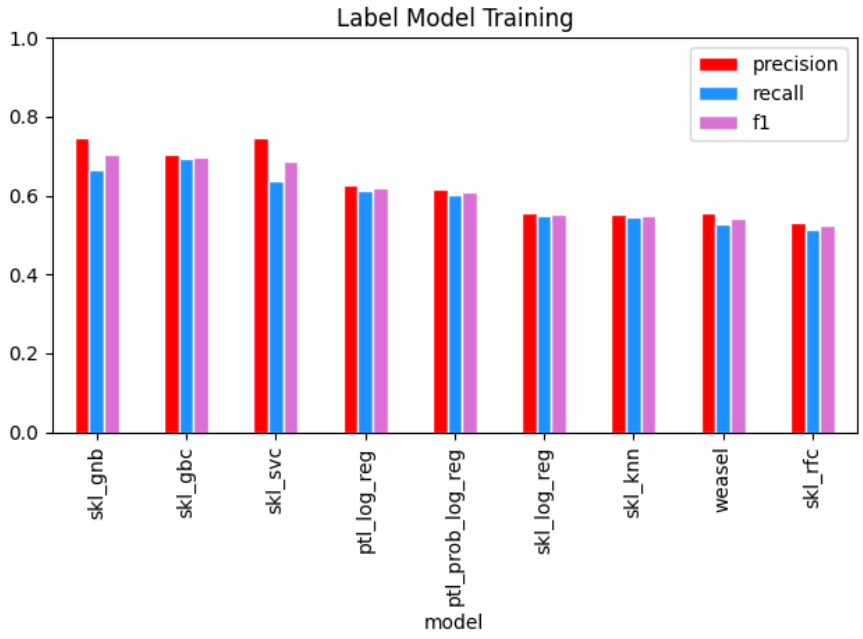


Figure 5.9: Model performances trained on *label model*, sorted by the F1-score.

the five evaluation datasets. As we can see these results look similar to the results of the previously evaluated models trained on surrogate labels, making a strong case for continuing with the weak label approach for model tuning. Considering the performance metrics overall the three most promising models are the SKL GBC, SKL GNB, and SKL SVC. Depending on the model different evaluation sets perform better. However, the Eval DS1 set performs best in most cases. The best performances have been achieved with the full feature set from ILT2.

Model	Evaluation Set	LT	Features	Precision	Recall	F1	F1 Change
SKL RFC	Eval DS2 (Lin)	CLT	fs5	0.7149	0.7030	0.7089	+0.1662
SKL GNB	Eval DS1	CLT	fs1	0.7482	0.6667	0.7051	+0.0001
SKL KNN	Eval DS1	CLT	fs3	0.7042	0.7030	0.7036	+0.1539
SKL GBC	Eval DS1	CLT	fs5	0.7206	0.6832	0.7014	+0.0059
SKL SVC	Eval DS4 (Split Lin)	CLT	fs1	0.7455	0.6358	0.6863	+0.0005
SKL Log Reg	Eval DS1	CLT	fs3	0.6465	0.6469	0.6467	+0.0940

Table 5.7: Surrogate models trained with better features.

The key findings from Table 5.6 are:

- The SKL GNB has the highest precision and F1-score.
- The CLT labeling results perform best as input.
- Models other than SKL GNB, SKL SVC, and SKL GBC perform significantly worse across the board.
- The custom models are not worth the effort in this case.
- Eval DS1, Eval DS2 (Lin), and Eval DS4 (Split Lin) have the highest chance of evaluating positively.

Models Trained on Improved Features Since we have established previously that the Pytorch Lightning models are not worth the additional effort because their performance has not been higher, we have focused on the scikit-learn models from now on. These have been trained in this round with the features extracted from Listing 5.1, to see if this improves the performance and if there is a set of features that performs considerably better than any other.

After implementing the better feature sets, we observed a substantial improvement in the performance of the models that can be seen in Table 5.7. Among all the models, we noted the most substantial increase in the accuracy of SKL RFC and SKL KNN algorithms. On the other hand, the accuracy scores of SKL GNB and SKL GBC models remained unchanged. Out of all the models, four have surpassed the benchmark of 70% accuracy, while one model is very close to reaching this mark, and its performance could further improve with some tuning. However, the logistic regression model is still lagging, with a noticeable difference of more than 5% in each metric compared to the other models.

Through our experimentation with enhanced feature sets and weak labels, we have identified notable improvements in model performance in select cases, as can be seen in Table 5.8. The SKL KNN model exhibited the most substantial improvement, experiencing an increase of more than 17%. Additionally, the SKL RFC model demonstrated a 16% improvement, while the SKL Log Reg model exhibited a 14% improvement, re-establishing its place in the competition. It's crucial to acknowledge that while some models showcased

Model	Evaluation Set	LT	Features	Precision	Recall	F1	F1 Change
SKL KNN	Eval DS2 (Lin)	CLT	fs5	0.7221	0.7195	0.7208	+0.1711
SKL GNB	Eval DS1	CLT	fs3	0.7138	0.6997	0.7067	+0.0017
SKL Log Reg	Eval DS3 (Split Lin)	CLT	fs2	0.7046	0.6852	0.6948	+0.1421
SKL GBC	Eval DS1	CLT	fs4	0.6851	0.6832	0.6841	- 0.0150
SKL RFC	Eval DS1	CLT	fs4	0.6851	0.6832	0.6841	+0.1608
SKL SVC	Eval DS2 (Lin)	ILT1	fs3	0.6576	0.5314	0.5878	- 0.0984

Table 5.8: Label models trained with better features.

Features	Evaluation Set	LT	Model	Precision	Recall	F1
fs5	Eval DS2 (Lin)	CLT	SKL KNN	0.7221	0.7195	0.7208
fs3	Eval DS2 (Lin)	CLT	SKL KNN	0.7129	0.7129	0.7129
fs4	Eval DS1	ILT2	SKL KNN	0.6970	0.6931	0.6950
fs2	Eval DS2 (Lin)	ILT2	SKL GBC	0.6973	0.6865	0.6918
fs1	Eval DS3 (Split)	CLT	SKL RFC	0.6953	0.6698	0.6823

Table 5.9: Feature sets ranked by F1-score

a significant performance boost, others experienced a decline in performance following feature selection. Nonetheless, we attained the highest overall performance in this analysis.

In a final attempt, we try to discern performances between the feature sets, shown in Table 5.9. The best-performing feature set is *fs5*. However, there does not seem to be a sizable gap between the different sets, as most features overlap anyway.

5.2.2 Hyperparameter Tuning

Based on the results of the previous subsections, we have decided not to exclude any more models from the hyperparameter tuning. We are using the weak labels from each task, as well as weak labels generated by applying the most significant labeling rules. Furthermore, we are using the improved feature sets (*fs1-5*), as well as the features that occur in the labeling rules per labeling task for training. The parameter search space is illustrated in Table 5.10.

Maximizing the training metrics during the hyperparameter tuning did not yield a better result than choosing the default parameters for the models. On the contrary, fitting the models better to the training data meant that they had worse outcomes on the evaluation data, as shown in Table 5.11.

Nevertheless, when evaluating the whole model and parameter grid against the evaluation set we can find a configuration where the results can be improved even further, as seen in Table 5.12. This is not practical in a real-world scenario, since the labels for an evaluation set might not be known, thus not allowing for model tuning in this direction.

Model	SKL Log Reg
Parameter	Values
C	1e-1, 1, 1e3, 1e5
Penalty	l1, l2, elasticnet, None
Solver	liblinear, lbfgs, newton-cg, saga
L1 ratio	0.1, 0.25, 0.5, 0.75, 0.9

Model	SKL SVC
Parameter	Values
C	1e-1, 1, 10, 1e3
Kernel	rbf, poly
Poly Degree	3, 5

Model	SKL KNN
Parameter	Values
Neighbors	range(1,50)

Model	SKL GBC
Parameter	Values
Estimators	1, 10, 100, 1000
Learning Rate	1, 1e-1, 1e-2, 1e-3
Max Depth	1, 5, 10, 25, 50, Inf

Model	SKL RFC
Parameter	Values
Estimators	1, 10, 100, 1000
Criterion	gini, entropy, log_loss
Max Depth	1, 5, 10, 25, 50, Inf

Model	SKL GNB
Parameter	Values
Smoothing	1e-11, 1e-9, 1e-7, 1e-5, 1e-3, 1e-1

Table 5.10: Parameter space for hyperparameter tuning

Model	Evaluation Set	LT	Labels	FS	Eval Prec	Eval Rec	Eval F1
SKL KNN	Eval DS1	CLT	Best	fs3	0.7100	0.7063	0.7081
SKL GBC	Eval DS3 (Split)	CLT	Best	fs3	0.7064	0.6852	0.6956
SKL RFC	Eval DS1	CLT	Full	fs4	0.6851	0.6832	0.6841
SKL Log Reg	Eval DS3 (Split)	CLT	Best	fs1	0.6975	0.6698	0.6833
SKL SVC	Eval DS1	ILT1	Best	Full	0.3642	0.3630	0.3636
SKL GNB	Eval DS5 (Split Cub)	ILT1	Best	Full	0.3443	0.3488	0.3465

Table 5.11: Model results achieved by maximizing the training metrics. The column Labels describes which set of labeling rules was used to extract the weak labels from the label model.

Model	LT	Labels	FS	Precision	Recall	F1
SKL KNN	ILT2	Full	fs3	0.7398	0.7360	0.7379
SKL RFC	ILT2	Full	Full	0.7441	0.7191	0.7314
SKL GBC	ILT1	Full	fs2	0.7471	0.6964	0.7209
SKL GNB	ILT1	Best	fs3	0.7275	0.7006	0.7138
SKL Log Reg	ILT2	Full	fs5	0.7121	0.6944	0.7031
SKL SVC	CLT	Best	Full	0.7406	0.6337	0.6830

Table 5.12: Model results by maximizing the evaluation set metrics. The column Labels describes which set of labeling rules was used to extract the weak labels from the label model.

			Precision		Recall		F1	
LT	LR	Model	LM	SM	LM	SM	LM	SM
CLT	Best	SKL RFC	0.6364	0.7967	0.5920	0.6667	0.6134	0.7259
CLT	Full	SKL GNB	0.5881	0.7974	0.5747	0.6264	0.5813	0.7017
ILT1	Best	SKL GBC	0.7846	0.8378	0.5805	0.7644	0.6673	0.7994
ILT1	Full	SKL RFC	0.8222	0.7842	0.7989	0.7701	0.8104	0.7771
ILT2	Best	SKL GBC	0.7731	0.8351	0.5345	0.7586	0.6320	0.7950
ILT2	Full	SKL KNN	0.5311	0.5176	0.5402	0.5287	0.5356	0.5231

Table 5.13: Hyperparameter training results (SM) when compared to the gold labels per labeling task (LM).

Performance improvements are, however, attainable if the evaluation set is a real subset of the training set, meaning that the data source is the same. In the case of our models, we could achieve an increase in most labeling tasks compared to only applying the label model, as shown in Table 5.13. This directly loops back to the initial results in Table 5.2 and shows the generalization and tuning capabilities of the models.

LT	LR	Model	Precision LM	Precision SM	Recall LM	Recall SM	F1 LM	F1 SM
CLT	Best	SKL RFC	0.6364	0.7967	0.5920	0.6667	0.6134	0.7259
CLT	Full	SKL GNB	0.5881	0.7974	0.5747	0.6264	0.5813	0.7017
ILT1	Best	SKL GBC	0.7846	0.8378	0.5805	0.7644	0.6673	0.7994
ILT1	Full	SKL RFC	0.8222	0.8898	0.7989	0.8621	0.8104	0.8757
ILT2	Best	SKL GBC	0.7731	0.8351	0.5345	0.7586	0.6320	0.7950
ILT2	Full	SKL RFC	0.5311	0.6179	0.5402	0.6149	0.5356	0.6164

Table 5.14: Hyperparameter training results (SM) when compared to all unambiguous gold labels (LM).

We saw an even bigger increase in performance across all tasks, when we evaluated the model results against the unambiguous gold labels (Table 5.14). The minimum increase exceeds 5% in all metrics and tops out at an increase of above 20% in some cases.

Evaluation across the various optimization stages Figure 5.10 shows the evaluation performance across the various stages during our evaluation and illustrates that in most cases at least a similar performance is attainable by utilizing weak labels and a classification model compared to using supervision. Across the board, we have achieved F1-scores that are similar if not improved over a supervised approach by using weak labels and one of the tuned ML models.

5.2.3 Feature Importance

In an attempt to find features with a big impact across the different models and evaluation results we have calculated and compared the feature importance between models trained on gold labels and models trained on the predicted labels. Feature importance is in general not comparable across multiple models because the definition varies based on the interpretation. Regarding the interpretations of features in models, linear models determine it by analyzing the model's coefficients, with the feature's importance indicated by the coefficient's magnitude and its direction by the sign. Meanwhile, tree-based models measure feature importance by considering how much it reduces impurity or how frequently it is used to split nodes. To overcome these difficulties we have scaled the importance with a min-max scaler, to get the relative importance for a model.

For our evaluation, we have decided to focus on Spearman's rank correlation coefficient [111]. Spearman's rank correlation is a method to measure the rank correlation between variables without making any assumptions about their underlying probability distribution. It evaluates the degree to which the relationship between two variables can be explained by a monotonic function. This method is particularly useful when the data doesn't follow a normal distribution or when the relationship between the variables is not linear.

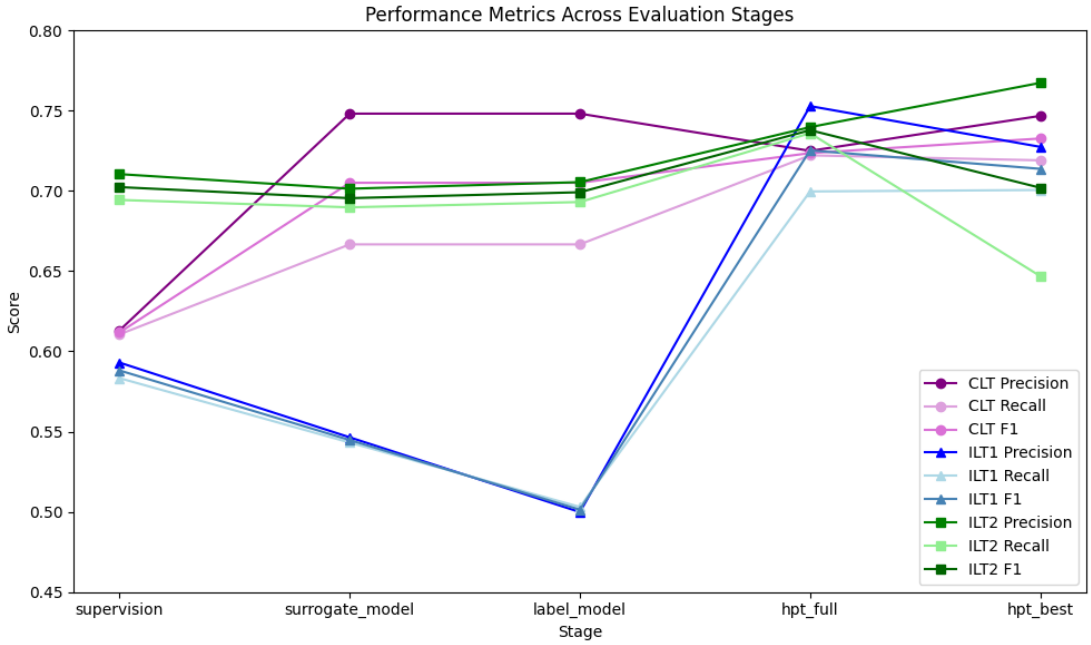


Figure 5.10: Movement of performance metrics for each labeling task across the evaluation stages.

Not surprisingly, the correlation coefficient is mostly positive between models trained on the ground truth and models trained on the best predictions from our hyperparameter tuning section. This at least indicates that the models capture the underlying data structure well and it emphasizes the consistency between the models. Also not surprisingly, we have seen that when we correlate the feature importance between the ground truth models there is a negative correlation. This emphasizes what we have seen in the previous section, that models that are fitted better to the evaluation sets perform worse on the ground truth sets and vice versa. One step further, comparing the prediction model performances we get low positive correlation coefficients, further strengthening the hypothesis that there is not one model that fits all if the data set is highly variable.

We have further used the importance coefficients to create a bar chart, shown in Figure 5.11, showcasing the relative scaled importance across all models when used for classifying seasonal and constant data. The three relatively most important features in our case are the *fft_aggregated___aggtype_variance*, the *time_reversal_asymmetry_statistic_lag_1*, and the *variance*.

5.2.4 Evaluation Sets

While we also set out to evaluate if there is an evaluation set that performs best, hence also a superior interpolation algorithm (at least for the problem at hand), we have not

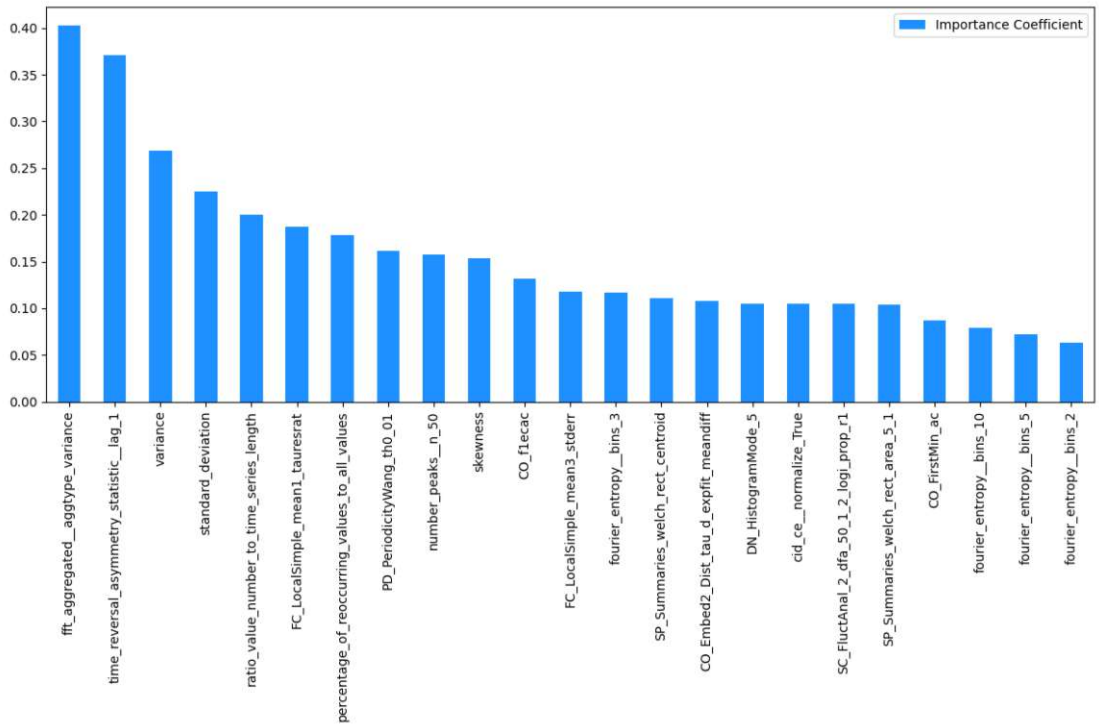


Figure 5.11: Feature importance coefficients scaled and averaged across all models.

seen a drastic change in the performance evaluations. The difference in the evaluation metric is per evaluation set below 2%. The reasoning for continuing to evaluate against all sets anyway is the low cost of prediction, compared to training.

5.2.5 Survey Results

The survey was completed by five participants, four of whom provided labeling tasks for the labeling task evaluation. We gathered feedback from these individuals on their usage of a time series data labeling tool. Due to the small sample size of only five respondents, the conclusions drawn lack significance. Generalizing the survey results of the time series data labeling tool is impossible with such a limited group. The potentially valuable insights may not accurately reflect the views, experiences, and needs of a larger user base due to this lack of representativeness. The diversity and variability inherent in a larger sample are missing, which leads to skewed perceptions of user satisfaction and tool effectiveness. Averages, proportions, and correlations derived from such a small group are subject to high variability and can be overly influenced by outliers. Drawing firm conclusions or making significant decisions based on the data is difficult due to this instability. However, the qualitative feedback and the feedback that is directly related to labeling tasks become an important source of insight under these circumstances. The open-ended responses offer a deeper understanding of experiences and perceptions,

highlighting areas of appreciation as well as suggestions for improvement.

The form contained various fields regarding the participants' background, work experience, education, knowledge of data labeling concepts, participation in labeling tasks, overall satisfaction with the tool, and their likelihood of recommending it to others. The data collected was a mix of categorical data, such as job title, experience level, and educational qualifications, ordinal data, including familiarity and contentment ratings, and comments left in a free-text format.

The main results from the survey are:

- Roles Distribution: The majority of respondents are Data Scientists (80%), with the remainder being Researchers (20%).
- Experience Levels Distribution: Most respondents have 3 to 5 years of experience in their current role (60%), followed by those with 1 to 2 years (20%) and 6 to 10 years (20%).
- Educational Backgrounds Distribution: 3 of 5 hold a Mathematics/Statistics Degree and 2 of 5 hold a Computer Science/Software Engineering degree.
- Familiarity with Data Labeling: Each of the participants selected a different level of familiarity from a scale of 1 to 5, resulting in a big spread.
- Familiarity with Time Series Data: All participants are familiar with time series data and have at least some experience in analyzing and interpreting the data, through various applications.
- Data Labeling Experience: 60% of respondents have been involved in data labeling tasks in their professional or academic work.
- Seasonality: All participants have a general understanding of seasonality and seasonal patterns.
- Labeling Rule Creation: Four participants preferred the Simple Rule Creator. One user preferred both, the Simple and the Advanced Rule Creator.
- Overall Satisfaction and Tool Recommendation: The average satisfaction rating with the time series data labeling tool is 7.8 out of 10. Furthermore, all users would recommend utilizing a tool with a similar suite of functionality to their colleagues and peers.
- Potential: Most participants (4 out of 5) anticipate that the tool can potentially save time during data labeling processes.
- Collaboration: Collaboration has been rated as important by the participants. Additionally, the tool has been rated to facilitate collaboration. Collaboration was also considered most crucial for the work by one participant.

- Satisfaction with the Labeling Performance: 0 participants were fully satisfied with their labeling performance. One remark was that it is hard to find good surrogate models.

Taking a closer look at the analysis, there is a positive correlation coefficient between data labeling familiarity and years of experience, as well as a negative correlation coefficient between the years of experience and overall satisfaction with the time series data labeling tool. This negative correlation suggests that as the years of experience and the data labeling expertise increase, the overall satisfaction with the tool slightly decreases. It might indicate that more experienced users have higher expectations or different needs when it comes to data labeling tools.

The survey overall paints a positive picture in regard to the labeling demo and the participants offered valuable insight into the results.

CHAPTER 6

Discussion

The discussion follows the initially posed research questions structurally and aims to answer them by interpreting the results in Section 5 before finally discussing threats to validity.

Research Question 1 *How effective are simple heuristics and weak supervision in labeling time series data compared to manual labeling? How effective is it compared to a supervised approach with less data?*

In our experiment, we have obtained a small set of hand-labeled data, details of which are given in Section 5.2. Out of the entire dataset, 38 time series received two or more labels, of which 7 were labeled differently by the participants. This implies that 18% of data points were erroneously labeled by humans. The error rate per labeling task may even be higher since overall only about 17% of time series had two labels. Therefore, the human benchmark performance for label agreement is approximately 82%. Interestingly, one of the participants also noted that the scale at which a time series is analyzed plays a crucial role in its classification and can be an issue between labelers. This further underscores the necessity for automated label generation.

Using these labels to train supervised machine learning models, we achieved a best-case performance of 70%, with most performances ranging from 58-61% when compared to our evaluation set. While maximizing the metrics of the models, it was found that we could push all labeling task metrics up to 74%, but not beyond that. This could be attributed to the different sampling rates, the vastly different amount of observations per time series, as well as the nature of the datasets. The evaluation set is solely based on server metrics, whereas the other sets were obtained through different means, as outlined in Section 5.1.1. However, attaining a widely available training set equal in structure and type to the evaluation set was not possible.

In a real-world scenario, there might also not be an evaluation set available and the only validation can be obtained by firstly labeling parts of the data. This can be the case because there is newly generated data that needs to be labeled, the labels of existing data changed, or some of the characteristics of the labels evolved. This case has been thoroughly evaluated by considering the labels that we attained through the experiment instead of the evaluation set. This might arguably be a better indicator for real-world performance, than any external evaluation set. By considering these gold labels, we get a much more uniform set of time series, where we were able to improve the f1-score to a maximum of 87% in one case, and in general, to an average of 73.64%. This is significantly higher than the evaluation set average and also exceeds our goal of 70%. Considering the best-case scenario we were able to outperform our human benchmark of 82%. Furthermore, we have to take into consideration that there might be room for improvements in regard to the labeling rules, based on the medium satisfaction of the users with their labeling results.

How simple are the applied heuristics?

While many of the features cannot be considered simple heuristics by themselves, such as Fourier transform or autocorrelation features, the burden of extracting them has been moved away from the demo application and hence the user. Furthermore, even though the demo application allowed for nested conditions and an arbitrary amount of boolean arithmetic, 95% of the rules applied through the demo were based on one extracted feature, evaluated against a condition to label a time series. Therefore, we consider the heuristics to be simple.

Can the results be generalized in the space of classifying seasonal and non-seasonal data?

As shown in Section 5.2.2, the results depend heavily on the data. This claim is further supported by the negative Spearman's rank correlation coefficients between the baseline evaluation and gold label sets, as outlined in Section 5.2.3. Therefore, the generalization capabilities are limited within our non-homogeneous set of time series.

Research Question 2 *How can the labeling workflow be designed for time series data? How can users be aided in the process of writing labeling functions?*

We have gone through multiple iterations of the labeling workflow to improve upon the process where users label time series manually and collaborate to get consensus on their results. The entire approach is detailed in Section 4.1, and it is divided into two main parts: rule creation and labeling.

When creating rules, users can select the level of flexibility they desire and need to design the rules. The level of flexibility affects the amount of information available to users, as higher flexibility comes with less information. The rule creator with the least flexibility works with drop-down fields, sliders, and a statistics expander to provide a broad overview of the feature distribution. In contrast, the most flexible creator is a free-form text field. Rule creation is aided by a simple distribution of time series per

label, colored green, red, or black, depending on whether they are classified correctly, incorrectly, or not at all, respectively. This feature has been found to be particularly beneficial for rule creation. However, the color choices have been deemed inappropriate for individuals with red-green blindness. In addition, a set of thumbnails can be displayed to provide another visual cue about the classification per rule. Overall, these features have been found to be especially useful during rule creation. The statistics expander could have been improved by using metric distributions instead of raw numbers.

In the second part of the process, users label the dataset using a table that displays a thumbnail of the time series, the label per model (label model, surrogate model), and the gold label. Users can modify and set gold labels directly from the table. Suppose the thumbnail and labeling results are insufficient for a user. In that case, they can navigate to the time series analysis, where the entire graph, along with all features and labeling results, is displayed. Furthermore, users can set the gold label immediately from there.

To enable users to adjust their surrogate model, we have provided them with four different models and model parameters and metrics to track the changes in metrics.

Considering the input format for labeling functions, which formats are preferred by users?

During the survey, participants provided valuable feedback about their experience using the application. Additionally, a qualitative feedback session was conducted to gather more in-depth feedback. The results indicated that in general, participants preferred a simple approach to the application. It was observed that the more advanced rule creators were considered too complicated for the users who all had less than two hours of experience with the application. However, these tools could prove to be beneficial for users who need to dive deeper into labeling and refining the labels of a dataset. One user expressed a desire for more flexibility during label design. They suggested that full calculations should be possible. These calculations could also alleviate the need to extract all the features beforehand, at the cost of an increased computation time during the labeling.

Despite this, the user feedback was generally positive in regard to the overall navigation, user-friendliness, and workflow of the application.

Research Question 3 *Which classification models are superior in generalizing beyond weakly labeled time series data and what are the potential trade-offs?*

After extensive hyperparameter tuning, the SKL RFC model was found to be the top performer in most testing scenarios. It demonstrated robustness against variations in the feature set and outperformed other models by a margin of over 10%. The SKL GBC and the SKL KNN models also performed well, ranking closely behind the SKL RFC. Notably, both the SKL KNN and SKL RFC models showed a significant improvement in performance after careful tuning, highlighting the importance of hyperparameter optimization in maximizing model effectiveness.

Our evaluation showed that simplifying the model by using fewer features led to a negligible performance difference between the different models. This allowed all models to achieve high levels of training accuracy and precision. However, this approach highlighted certain trade-offs, especially with tree-based classifiers like SKL RFC and SKL GBC. These models delivered superior performance but can be challenging in terms of interpretability and computational efficiency when dealing with larger models. It becomes more difficult to trace how input features influence model predictions, while also increasing the computational resources needed.

Furthermore, the SKL KNN model, while effective, was noted for its high computational cost and memory requirements, which could be problematic with larger models and might not be feasible at all. Meanwhile, the SKL GNB model's assumption of feature independence can be a significant limitation, given the evident interdependence of features in our dataset. Moreover, while the SKL SVC is generally praised for its performance in high-dimensional spaces, this advantage was less relevant in our scenario, which involved a reduced feature set. Ultimately, our exploration confirmed that tree-based models, despite their interpretability and computational drawbacks, outperformed other models.

Are there any features that influenced the model performance substantially?

After our hyperparameter tuning process, we established a comparative basis for our models through the calculation and scaling of Spearman's correlation coefficients. The visualization provided in Figure 5.11 highlights that among the various features evaluated, two stand out as more effective predictors within our models, underscoring their pivotal role in influencing model performance. However, these coefficients are highly dependent on the subset of features chosen by the labeling participants, and even though they occur consistently with high values in our results, might not be the best predictors for seasonality.

Threats to Validity The field of empirical software engineering research faces multiple challenges, as outlined in the work of Feldt et al. [112]. These challenges can be broadly categorized into three types: Internal Validity, External Validity, and Construct Validity. While the mixed-methods evaluation employed in this study has been designed with these threats in mind, it is impossible to avoid them altogether. Some of the specific threats encountered in this study have been previously discussed, but we will elaborate on them here.

The *Internal Validity* threat concerns the accuracy of the measurement. This means ensuring that the results obtained are what we intend to measure and that we are not misinterpreting any data. To address this threat, we have utilized various models to try to map linear and non-linear relationships and ensure that the measurements are accurate. However, there is always a possibility of misinterpretation.

The threat of *External Validity* concerns the transferability of the results beyond the study's context. This means there is no guarantee that the results we obtained in this study are generalizable beyond what we have shown. The highly dynamic nature of

time series data, such as differences in sampling rates, different lengths, and growing or changing data, make it difficult to generalize the results. We have used non-related data to mitigate this threat, but our evaluation indicates that the results may still not be fully generalizable. Additionally, the small sample size of participants further limits the impact of the results.

The *Construct Validity* threat pertains to the suitability of the methodology used to address the research question. In this study, defining, implementing, and subsequently ranking input formats according to a survey was particularly challenging. Furthermore, it cannot be guaranteed that a construct has not been clearly defined and, therefore, misrepresented, making the drawn conclusions inadequate.

CHAPTER 7

Conclusion

In this thesis, we aimed to address the feasibility of automating the labeling process for time series data by applying simple heuristics, so-called labeling functions, in conjunction with machine learning models. An evaluation framework based on a real-world case was developed to approach this goal. Furthermore, to aid the evaluation, a complete labeling workflow was developed. The evaluation data generation was facilitated via a demo application. The demo application was developed to enable participants to interactively label time series data by applying labeling rules to the datasets. The application development was supported by applying state-of-the-art frameworks and tools.

The labeling results were then dissected. During the initial evaluation, we found evidence that study participants were not in total agreement on time series labels, the so-called gold labels. This supports the claims of subjectivity in labeling data and hence proves the importance of a standardized labeling workflow. The labeling results were further used to extract supervised benchmark performances. These benchmarks were used as a comparison for the evaluation models trained on weak labels and surrogate labels. Performance improvements were shown during hyperparameter tuning, underscoring the necessity for well-tuned models. We found that achieving high levels of performance comparable to manual labeling and supervised labeling is possible under certain conditions.

Our investigation, therefore, highlighted some promising aspects of automated labeling, including identifying superior models and the necessity of hyperparameter tuning to achieve optimal performance. However, it also revealed a significant limitation: the results of our study are not broadly generalizable. This limitation points to the nuanced challenges inherent in applying machine learning techniques to diverse datasets and underscores the complexity of the task. While the concrete results are not generalizable, the survey responses suggest a general positive impact of such a system.

The findings of this research suggest a path forward that necessitates additional investigation. Specifically, there is a clear need for further research to explore the applicability in regard to different labels and datasets. Such efforts are crucial for enhancing the utility and effectiveness of automated time series labeling processes in a broader array of contexts.

In sum, this thesis contributes to the ongoing conversation about the automation of data labeling, particularly within the context of time series data. While we have demonstrated that it is feasible to develop a programmatic labeling workflow that can perform comparably to manual efforts, the limited generalizability of our results underscores the challenges that remain. Future research in this area is essential for extending the reach and relevance of these findings and for continuing to refine the processes and technologies involved in automated labeling.

An artifact for reproduction can be found on Zenodo using the following URL: `https://zenodo.org/records/10871162`.

CHAPTER 8

Appendix

```python
import pandas as pd

feature_matrix = pd.DataFrame()

def lookup(x, feature, feature_matrix):
    return feature_matrix.at[x.name, feature]

def rule_Small_variance_indicates_Constant_Threshold(x):
    return 0 if lookup(x, "variance", feature_matrix) < 0.1 else -1

def rule_Large_skewness_indicates_Constant_Threshold(x):
    return 0 if lookup(x, "skewness", feature_matrix) > 1 else -1

def rule_Many_duplicate_values_indicates_Constant_Threshold(x):
    return 0 if lookup(x, "ratio_value_number_to_time_series_length",
    feature_matrix) < 0.9 else -1

def rule_Small_fourier_entropy__bins_5_indicates_Seasonal_Baselining(x):
    return 1 if lookup(x, "fourier_entropy__bins_5", feature_matrix) < 0.2
    else 0

def rule_Small_fourier_entropy__bins_3_indicates_Seasonal_Baselining(x):
    return 1 if lookup(x, "fourier_entropy__bins_3", feature_matrix) < 0.1
    else 0

def rule_Small_fourier_entropy__bins_2_indicates_Seasonal_Baselining(x):
    return 1 if lookup(x, "fourier_entropy__bins_2", feature_matrix) < 0.08
    else 0

def rule_Time_reversal_asymmetry_lag_1_indicates_constant_threshold(x):
    return -1 if lookup(x, "time_reversal_asymmetry_statistic__lag_1",
    feature_matrix) > -1.8314717772846557e+21 else 0

def rule_number_peaks__n_50_indicates_seasonal_baseline(x):
```

```
30      return -1 if lookup(x, "number_peaks__n_50", feature_matrix) < 85.915
        else 1
31
32  def rule_More_than_20_percentage_of_reoccuring_values_to_all
33      _values_indicate_constant_threshold(x):
34      return 0 if lookup(x, "percentage_of_reoccurring_values_to_all_values",
        feature_matrix) > 0.20139 else -1
35
36  def rule_embedding_dist(x):
37      return 0 if lookup(x, "CO_Embed2_Dist_tau_d_expfit_meandiff",
        feature_matrix) < 0.12 else 1
38
39  def rule_Minimal_Variance(x):
40      return 0 if lookup(x, "variance", feature_matrix) < 0.010 else 1
41
42  def rule_Min_AC(x):
43      return 0 if lookup(x, "CO_FirstMin_ac", feature_matrix) < 7.0 else 1
44
45  def rule_Min_standard_deviation(x):
46      return 0 if lookup(x, "standard_deviation", feature_matrix) < 0.10 else 1
47
48  def rule_ForecastError1(x):
49      return 0 if lookup(x, "FC_LocalSimple_mean3_stderr", feature_matrix) <
        0.17488 else 1
50
51  def rule_Change_in_AC_after_diff(x):
52      return 0 if lookup(x, "FC_LocalSimple_mean1_tauresrat", feature_matrix) <
         0.773398 else 1
53
54  def rule_Periodicity_Metric(x):
55      return 0 if lookup(x, "PD_PeriodicityWang_th0_01", feature_matrix) < 10.0
         else 1
56
57  def rule_Logi_Scaling(x):
58      return 0 if lookup(x, "SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1",
        feature_matrix) < 0.17016 else 1
59
60  def rule_Sum_20_low(x):
61      return 0 if lookup(x, "SP_Summaries_welch_rect_area_5_1", feature_matrix)
         >= 0.999 else 1
62
63  def rule_F1AC(x):
64      return 0 if lookup(x, "CO_f1ecac", feature_matrix) > 128.62096999936 else
         1
65
66  def rule_4dd90fb2_22a4_40dc_b1b9_5fb4c2c4c11d(x):
67      return 0 if lookup(x, "cid_ce__normalize_True", feature_matrix) < 1.0
        else 1
68
69  def rule_8e7e0683_9824_475b_a3f6_7449e3281c7c(x):
70      return 1 if lookup(x, "fft_aggregated__aggtype_variance", feature_matrix)
         > 400000.0 else 0
71
```

```
72 def rule_c6e90c15_c4d9_4564_820c_af71ba4eb025(x):
73     return 0 if lookup(x, "fourier_entropy__bins_10", feature_matrix) >
       0.211776 else 1
74
75 def rule_99ce81a2_8c98_4ed4_ac97_b79efac28bcb(x):
76     return 0 if lookup(x, "DN_HistogramMode_5", feature_matrix) > 0.5 or
       lookup(x, "DN_HistogramMode_5",
77
          feature_matrix) < -0.5 else 1
78
79 def rule_588654e0_5822_43c0_b2c5_ac54fe684839(x):
80     return 0 if lookup(x, "PD_PeriodicityWang_th0_01", feature_matrix) < 23.0
        else -1
81
82 def rule_56c3bc47_3043_4750_8020_e180e98ed299(x):
83     return 0 if lookup(x, "SP_Summaries_welch_rect_centroid", feature_matrix)
        > 1.0 else -1
84
85 def rule_f83f8771_f75c_445d_a782_b60c9839ddf3(x):
86     return 1 if lookup(x, "FC_LocalSimple_mean3_stderr", feature_matrix) >
       0.1 else -1
```

Listing 8.1: Labeling rules extracted from experiment labeling tasks

Rule Name	Polarity	Coverage	Overlaps	Conflicts
rule_embedding_dist	0,1	1	1	0.9708
rule_Minimal_Variance	0,1	1	1	0.9708
rule_Min_AC	0,1	1	1	0.9708
rule_Min_standard_deviation	0,1	1	1	0.9708
rule_ForecastError1	0,1	1	1	0.9708
rule_Change_in_AC_after_diff	0,1	1	1	0.9708
rule_Periodicity_Metric	0,1	1	1	0.9708
rule_Logi_Scaling	0,1	1	1	0.9708
rule_Sum_20_low	0,1	1	1	0.9708
rule_F1AC	0,1	1	1	0.9708

Table 8.1: ILT1 Labeling Function Analysis

Rule Name	Polarity	Coverage	Overlaps	Conflicts
rule_4dd90fb2_22a4_40dc_b1b9_5fb4c2c4c11d	0,1	1	1	0.9896
rule_8e7e0683_9824_475b_a3f6_7449e3281c7c	0,1	1	1	0.9896
rule_c6e90c15_c4d9_4564_820c_af71ba4eb025	0,1	1	1	0.9896
rule_99ce81a2_8c98_4ed4_ac97_b79efac28bcb	0,1	1	1	0.9896
rule_588654e0_5822_43c0_b2c5_ac54fe684839	0	0.3420	0.3420	0.3413
rule_56c3bc47_3043_4750_8020_e180e98ed299	0	0.0170	0.0170	0.0170
rule_f83f8771_f75c_445d_a782_b60c9839ddf3	1	0.8832	0.8832	0.8766

Table 8.2: ILT2 Labeling Function Analysis

Rule Name	Polarity	Coverage	Overlaps	Conflicts
rule_Small_variance_indicates_Constant_Threshold	0	0.5275	0.5275	0.3515
rule_Large_skewness_indicates_Constant_Threshold	0	0.1291	0.1291	0.0720
rule_Many_duplicate_values_indicates_Constant_Threshold	0	0.1337	0.1337	0.0944
rule_Small_fourier_entropy___bins_5_indicates_Seasonal_Baselining	0,1	1	1	0.5381
rule_Small_fourier_entropy___bins_3_indicates_Seasonal_Baselining	0,1	1	1	0.5381
rule_Small_fourier_entropy___bins_2_indicates_Seasonal_Baselining	0,1	1	1	0.5381
rule_Time_reversal_asymmetry_lag_1_indicates_constant_threshold	0	0.0015	0.0015	0
rule_number_peaks___n_50_indicates_seasonal_baseline	1	0.0740	0.0740	0.0638
rule_More_than_20_percentage_of_reoccuring_values_to_all_values_indicate_constant_threshold	0	0.0605	0.0605	0.0486

Table 8.3: CLT Labeling Function Analysis

Feature	Type	Description
Catch22		
CO_Embed2_Dist _tau_d_expfit_meandiff	Other	Goodness of exponential fit to embedding distance distribution
CO_FirstMin_ac	Linear autocorrelation structure	First minimum of the ACF
CO_f1ecac	Linear autocorrelation structure	First 1/e crossing of the ACF
DN_HistogramMode_5	Distribution shape	5-bin histogram mode
FC_LocalSimple _mean1_tauresrat	Incremental differences	Change in autocorrelation timescale after incremental differencing
FC_LocalSimple _mean3_stderr	Simple forecasting	Error of 3-point rolling mean forecast
PD_PeriodicityWang _th0_01	Linear autocorrelation structure	Wang's periodicity metric
SC_FluctAnal_2_dfa _50_1_2_logi_prop_r1	Self-affine scaling	Detrended fluctuation analysis (low-scale scaling)
SP_Summaries_welch _rect_area_5_1	Linear autocorrelation structure	Power in lowest 20% frequencies
SP_Summaries_welch _rect_centroid	Linear autocorrelation structure	Centroid frequency

Table 8.4: Description of catch22 features applied in the experiment

Feature	Type	Description
TSFresh		
cid_ce__normalize _True	Simple	Calculator for time series complexity
fft_aggregated__aggtype _variance	Combiner	Spectral variance of the absolute fourier transform spectrum
fourier_entropy__bins_10	Simple	Binned entropy of the power spectral density (using welch method)
fourier_entropy__bins_2	Simple	Binned entropy of the power spectral density (using welch method)
fourier_entropy__bins_3	Simple	Binned entropy of the power spectral density (using welch method)
fourier_entropy__bins_5	Simple	Binned entropy of the power spectral density (using welch method)
number_peaks__n_50	Simple	The number of peaks of at least support n in the time series x
percentage_of_reoccurring _values_to_all_values	Simple	The percentage of values that are present in the time series more than once
ratio_value_number_to _time_series_length	Simple	A factor which is 1 if all values in the time series occur only once, and below one if this is not the case
skewness	Simple	Skewness of x
standard_deviation	Simple	Standard deviation of x
time_reversal_asymmetry _statistic__lag_1	Simple	Time reversal asymmetry statistic
variance	Simple	Variance of x

Table 8.5: Description of tsfresh features applied in the experiment

Model	Evaluation Set	LT	Accuracy	Precision	Recall	Features	Trained on
skl_knn	Eval DS2 (Lin)	CLT	0.7195	0.7221	0.7195	fs5	LM
skl_gnb	Eval DS1	CLT	0.6997	0.7138	0.6997	fs3	LM
skl_gbc	Eval DS2 (Lin)	ILT2	0.6931	0.7055	0.6931	full	LM
skl_gbc	Eval DS2 (Lin)	ILT2	0.6898	0.7014	0.6898	full	SM
skl_log_reg	Eval DS4 (Split Lin)	CLT	0.6852	0.7046	0.6852	fs2	LM
skl_gbc	Eval DS1	CLT	0.6832	0.6851	0.6832	fs4	LM
skl_rfc	Eval DS1	CLT	0.6799	0.6820	0.6799	fs4	LM
skl_gnb	Eval DS1	CLT	0.6667	0.7482	0.6667	full	SM
skl_gnb	Eval DS1	CLT	0.6667	0.7482	0.6667	full	LM
skl_svc	Eval DS2 (Lin)	CLT	0.6370	0.7429	0.6370	full	SM
skl_svc	Eval DS4 (Split Lin)	CLT	0.6358	0.7455	0.6358	full	LM
ptl_log_reg	Eval DS1	CLT	0.6145	0.6269	0.6145	full	LM
ptl_prob_log_reg	Eval DS1	CLT	0.6026	0.6174	0.6026	full	LM
skl_log_reg	Eval DS1	ILT2	0.5479	0.5578	0.5479	full	SM
skl_log_reg	Eval DS1	ILT2	0.5479	0.5578	0.5479	full	LM
skl_knn	Eval DS3 (Split)	CLT	0.5463	0.5533	0.5463	full	SM
skl_knn	Eval DS3 (Split)	CLT	0.5463	0.5533	0.5463	full	LM
skl_rfc	Eval DS5 (Split Cub)	ILT2	0.5327	0.5532	0.5327	full	SM
skl_svc	Eval DS2 (Lin)	ILT1	0.5314	0.6576	0.5314	fs3	LM
weasel	Eval DS3 (Split)	CLT	0.5290	0.5569	0.5290	full	LM
skl_rfc	Eval DS5 (Split Cub)	ILT2	0.5148	0.5323	0.5148	full	LM

Table 8.6: Full results table for model comparison

Model	Evaluation Set	LT	Accuracy	Precision	Recall	Features	Trained on
skl_knn	Eval DS2 (Lin)	CLT	0.7195	0.7221	0.7195	fs5	LM
skl_gnb	Eval DS1	CLT	0.6997	0.7138	0.6997	fs3	LM
skl_gbc	Eval DS2 (Lin)	ILT2	0.6931	0.7055	0.6931	full	LM
skl_log_reg	Eval DS4 (Split Lin)	CLT	0.6852	0.7046	0.6852	fs2	LM
skl_rfc	Eval DS1	CLT	0.6799	0.6820	0.6799	fs4	LM
skl_svc	Eval DS2 (Lin)	CLT	0.6370	0.7429	0.6370	full	SM
ptl_log_reg	Eval DS1	CLT	0.6145	0.6269	0.6145	full	LM
ptl_prob_log_reg	Eval DS1	CLT	0.6026	0.6174	0.6026	full	LM
weasel	Eval DS3 (Split)	CLT	0.5290	0.5569	0.5290	full	LM

Table 8.7: Cleaned rankings for ML models

List of Figures

List of Tables

Listings

Glossary

evaluation label A label for a time series that is part of the evaluation set and labeled by hand. 60

gold label A gold label, also called ground truth label, is the label that should be achieved in supervised machine learning scenarios. 20, 25, 29, 37–39, 55, 58, 60, 61

logit The **log**istic un**it** function is the quantile function associated with the standard logistic distribution [35]. 12

NaN is an abbreviation for Not a Number, which occurs when a value is undefined or unrepresentable. 55, 56, 58

predicted label A label that is generated by a ML model. 60

type I error Type I errors are FP instances of a classification problem, meaning that the instances are falsely classified as positive. 6

type II error Type II errors are FN instances of a classification problem, meaning that the instances are falsely classified as negative. 6

weak label A label that is generated by a Snorkel label model. 55, 60, 73

Acronyms

1-NN 1 nearest neighbor. 23

Best FS best feature set. 65

CLT Collaborative Labeling Task. 60, 61, 64, 65, 69, 90, 95

CNN convolutional neural network. 21

DJIA Dow Jones Industrial Average. 56

FM feature matrix. 29

FN false negative. 6, 99

FP false positive. 6, 99

GUI graphical user interface. 26

ILT1 Individual Labeling Task 1. 60, 61, 64, 90, 95

ILT2 Individual Labeling Task 2. 60, 61, 64, 65, 68, 90, 95

LF Labeling Function. 33

LM label model. 61

LOOCV Leave-One-Out Cross-Validation. 8

ML Machine Learning. 1–3, 32, 54, 60, 64, 73, 92, 96, 99

MLE Maximum Likelihood Estimation. 12, 13

MLP multilayer perceptron. 46

NLP Natural Language Processing. 24, 25

OvA one-versus-all. 15

OvO one-versus-one. 15

PoC Proof of Concept. 23

PTL Log Reg PyTorch Lightning Logistic Regression. 34, 63, 68

PTL Prob Log Reg PyTorch Lightning Probabilistic Logistic Regression. 34, 63, 68

RNN recurrent neural network. 22

RQ research question. 3, 53, 59

SKL scikit-learn. 58

SKL GBC scikit-learn Gradient Boosting Classifier. 63, 65, 67–73, 81, 82

SKL GNB scikit-learn Gaussian Naive Bayes. 63, 67–73, 82

SKL KNN scikit-learn K Neighbors Classifier. 63, 65, 67–72, 81, 82

SKL Log Reg scikit-learn Logistic Regression. 34, 61, 63, 65, 67–72

SKL RFC scikit-learn Random Forest Classifier. 63, 67–73, 81, 82

SKL SVC scikit-learn Support Vector Classifier. 63, 67–72, 82

SVM Support Vector Machine. 14, 15, 58

TN true negative. 6

TP true positive. 6

Bibliography

[1] D. Karimi, H. Dou, S. K. Warfield, and A. Gholipour, "Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis," *Medical Image Analysis*, vol. 65, p. 101759, Oct. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841520301237

[2] P. Liang, "Semi-supervised learning for natural language," Thesis, Massachusetts Institute of Technology, 2005, accepted: 2006-07-13T15:13:19Z. [Online]. Available: https://dspace.mit.edu/handle/1721.1/33296

[3] J. Zhang, Y. Yu, Y. Li, Y. Wang, Y. Yang, M. Yang, and A. Ratner, "WRENCH: A Comprehensive Benchmark for Weak Supervision," Oct. 2021, arXiv:2109.11377 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2109.11377

[4] S. Rühling Cachay, B. Boecking, and A. Dubrawski, "End-to-End Weak Supervision," Nov. 2021, arXiv:2107.02233 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2107.02233

[5] S. Galhotra, B. Golshan, and W.-C. Tan, "Adaptive Rule Discovery for Labeling Text Data," May 2020, arXiv:2005.06133 [cs]. [Online]. Available: http://arxiv.org/abs/2005.06133

[6] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, "Data Programming: Creating Large Training Sets, Quickly," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/6709e8d64a5f47269ed5cea9f625f7ab-Abstract.html

[7] P. Varma and C. Ré, "Snuba: automating weak supervision to label training data," *Proceedings of the VLDB Endowment*, vol. 12, no. 3, pp. 223–236, Nov. 2018. [Online]. Available: https://dl.acm.org/doi/10.14778/3291264.3291268

[8] S. Bandyopadhyay, A. Datta, and A. Pal, "Automated Label Generation for Time Series Classification with Representation Learning: Reduction of Label Cost for Training," Jul. 2021, issue: arXiv:2107.05458 arXiv:2107.05458 [cs]. [Online]. Available: http://arxiv.org/abs/2107.05458

[9] A. Law and A. Ghosh, "Multi-label classification using a cascade of stacked autoencoder and extreme learning machines," *Neurocomputing*, vol. 358, pp. 222–234, Sep. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523121930757X

[10] D. Mandal, S. Bharadwaj, and S. Biswas, "A Novel Self-Supervised Re-labeling Approach for Training with Noisy Labels," 2020, pp. 1381–1390. [Online]. Available: https://openaccess.thecvf.com/content_WACV_2020/html/Mandal_A_Novel_Self-Supervised_Re-labeling_Approach_for_Training_with_Noisy_Labels_WACV_2020_paper.html

[11] J. Z. Bengar, J. van de Weijer, B. Twardowski, and B. Raducanu, "Reducing Label Effort: Self-Supervised Meets Active Learning," 2021, pp. 1631–1639. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2021W/ILDAV/html/Bengar_Reducing_Label_Effort_Self-Supervised_Meets_Active_Learning_ICCVW_2021_paper.html

[12] Y. Wang, D. Song, W. Wang, S. Rao, X. Wang, and M. Wang, "Self-supervised learning and semi-supervised learning for multi-sequence medical image classification," *Neurocomputing*, vol. 513, pp. 383–394, Nov. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222011900

[13] H. Bai, M. Cao, P. Huang, and J. Shan, "Self-supervised Semi-supervised Learning for Data Labeling and Quality Evaluation," Nov. 2021, arXiv:2111.10932 [cs]. [Online]. Available: http://arxiv.org/abs/2111.10932

[14] R. Wirth and J. Hipp, "CRISP-DM: Towards a Standard Process Model for Data Mining," *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, Jan. 2000.

[15] Z.-H. Zhou, "Introduction," in *Machine Learning*, Z.-H. Zhou, Ed. Singapore: Springer, 2021, pp. 1–24. [Online]. Available: https://doi.org/10.1007/978-981-15-1967-3_1

[16] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433–460, 1950.

[17] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, place: US Publisher: American Psychological Association. [Online]. Available: https://doi.apa.org/doi/10.1037/h0042519

[18] B. Widrow, "An Adaptive "ADALINE" Neuron Using Chemical "MEMISTORS"," Stanford Electronics Laboratories, Technical Report 1553-2, Oct. 1960. [Online]. Available: https://www-isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf

[19] A. Newell and H. Simon, "The logic theory machine–A complex information processing system," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 61–79, Sep. 1956, conference Name: IRE Transactions on Information Theory. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1056797

[20] B. Russel and A. N. Whitehead, *Principia Mathematica*, 1st ed. University of Michigan, 1910. [Online]. Available: https://quod.lib.umich.edu/u/umhistmath/aat3201.0001.001

[21] E. A. Feigenbaum, "Expert systems in the 1980s," *State of the art report on machine intelligence. Maidenhead: Pergamon-Infotech*, vol. 23, 1981.

[22] R. Michalski, J. Carbonell, and T. Mitchell, *Machine learning. an Artificial Intelligence approach. Volume 2*, Jan. 1983, vol. Vol. II.

[23] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote Sensing of Environment*, vol. 62, no. 1, pp. 77–89, Oct. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425797000837

[24] A. Kulkarni, D. Chong, and F. A. Batarseh, "5 - Foundations of data imbalance and solutions for a data democracy," in *Data Democracy*, F. A. Batarseh and R. Yang, Eds. Academic Press, Jan. 2020, pp. 83–106. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128183663000058

[25] Z.-H. Zhou, "Model Selection and Evaluation," in *Machine Learning*, Z.-H. Zhou, Ed. Singapore: Springer, 2021, pp. 25–55. [Online]. Available: https://doi.org/10.1007/978-981-15-1967-3_2

[26] D. Berrar, "Cross-Validation," in *Encyclopedia of Bioinformatics and Computational Biology*. Elsevier, Jan. 2018, vol. 1, pp. 542–545.

[27] C. Schaffer, "Selecting a classification method by cross-validation," *Machine Learning*, vol. 13, no. 1, pp. 135–143, Oct. 1993. [Online]. Available: https://doi.org/10.1007/BF00993106

[28] I. Tougui, A. Jilbab, and J. E. Mhamdi, "Impact of the Choice of Cross-Validation Techniques on the Results of Machine Learning-Based Diagnostic Applications," *Healthcare Informatics Research*, vol. 27, no. 3, pp. 189–199, Jul. 2021, publisher: Korean Society of Medical Informatics. [Online]. Available: https://synapse.koreamed.org/articles/1147752

[29] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, vol. 4, no. none, pp. 40–79, Jan. 2010, publisher: Amer. Statist. Assoc., the Bernoulli Soc., the Inst. Math. Statist., and the Statist. Soc. Canada. [Online]. Available: https://projecteuclid.org/journals/statistics-surveys/volume-4/issue-none/A-survey-of-cross-validation-procedures-for-model-selection/10.1214/09-SS054.full

[30] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 1995, pp. 1137–1143.

[31] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, second edition ed. MIT Press, 2018. [Online]. Available: https://cs.nyu.edu/~mohri/mlbook/

[32] A. Ratner, S. Bach, P. Varma, and C. Ré, "An Overview of Weak Supervision." [Online]. Available: https://www.snorkel.org/blog/weak-supervision

[33] D. W. Hosmer and S. Lemeshow, *Applied Logistic Regression*, second edition ed. John Wiley & Sons, Ltd, 2000.

[34] C. Fan, Z. Xie, Y. Liu, C. Li, and H. Liu, "Adaptive Controller Based on Spatial Disturbance Observer in a Microgravity Environment," *Sensors*, vol. 19, p. 4759, Nov. 2019.

[35] J. Cramer, "The Origins and Development of the Logit Model," Aug. 2003.

[36] S. W. Menard, *Applied Logistic Regression Analysis*. Thousand Oaks, Calif. : Sage Publications, 2002. [Online]. Available: http://archive.org/details/appliedlogisticr0000mena_d2d4

[37] A. Bashir, "Logistic Regression Classification for Uncertain Data," *Research Journal of Mathematical and Statistical Sciences*, vol. 2, pp. 1–6, Feb. 2014.

[38] N. Gray and S. Ferson, "Logistic Regression Through the Veil of Imprecise Data," Jun. 2022, arXiv:2106.00492 [stat]. [Online]. Available: http://arxiv.org/abs/2106.00492

[39] P. L. Harrington Jr., A. Zaas, C. W. Woods, and G. S. Ginsburg, "Robust Logistic Regression with Bounded Data Uncertainties."

[40] T. S. Jaakkola and M. I. Jordan, "A Variational Approach to Bayesian Logistic Regression Models and their Extensions," in *Sixth International Workshop on Artificial Intelligence and Statistics*. PMLR, Jan. 1997, pp. 283–294, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/r1/jaakkola97a.html

[41] A. A. Johnson, M. Q. Ott, and M. Dogucu, *Bayes Rules! An Introduction to Applied Bayesian Modeling, Chapter 13 Logistic Regression*, Dec. 2021. [Online]. Available: https://www.bayesrulesbook.com/chapter-13

[42] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, ser. COLT '92. New York, NY, USA: Association for Computing Machinery, Jul. 1992, pp. 144–152. [Online]. Available: https://dl.acm.org/doi/10.1145/130385.130401

[43] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[44] S. Winters-Hilt, A. Yelundur, C. McChesney, and M. Landry, "Support Vector Machine Implementations for Classification & Clustering," *BMC Bioinformatics*, vol. 7, no. Suppl 2, p. S4, Sep. 2006. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1683575/

[45] "Support Vector Machines," in *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, J. Shawe-Taylor and N. Cristianini, Eds. Cambridge: Cambridge University Press, 2000, pp. 93–124. [Online]. Available: https://www.cambridge.org/core/books/an-introduction-to-support-vector-machines-and-other-kernelbased-learning-methods/support-vector-machines/DD4EA48AA6C383944EA67BF8A7BEC6CC

[46] O. Oyebode and D. Ighravwe, "Urban Water Demand Forecasting: A Comparative Evaluation of Conventional and Soft Computing Techniques," *Resources*, vol. 8, p. 156, Sep. 2019.

[47] T. G. Dietterich, "Ensemble Methods in Machine Learning," in *Multiple Classifier Systems*. Berlin, Heidelberg: Springer, 2000, pp. 1–15.

[48] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, Aug. 1995, pp. 278–282 vol.1. [Online]. Available: https://ieeexplore.ieee.org/document/598994

[49] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, publisher: Institute of Mathematical Statistics. [Online]. Available: https://www.jstor.org/stable/2699986

[50] R. Cowie, E. Douglas-Cowie, M. McRorie, I. Sneddon, L. Devillers, and N. Amir, "Issues in Data Collection," in *Emotion-Oriented Systems: The Humaine Handbook*, R. Cowie, C. Pelachaud, and P. Petta, Eds. Berlin, Heidelberg: Springer, 2011, pp. 197–212. [Online]. Available: https://doi.org/10.1007/978-3-642-15184-2_12

[51] N. Y. Yen, J.-W. Chang, J.-Y. Liao, and Y.-M. Yong, "Analysis of interpolation algorithms for the missing values in IoT time series: a case of air quality in Taiwan," *The Journal of Supercomputing*, vol. 76, no. 8, pp. 6475–6500, Aug. 2020. [Online]. Available: https://doi.org/10.1007/s11227-019-02991-7

[52] I. Pratama, A. E. Permanasari, I. Ardiyanto, and R. Indrayani, "A review of missing values handling methods on time-series data," in *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, Oct. 2016, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7858189?casa_token=PTEvFsb5DjIAAAAA:

icYz2skqxSzsynAN9N1rBJ9iVJqSpul9o9z5b05YU4C8b2VBSGomGmIiRnNTQ2ElkoJ_BwVlwuw

[53] M. N. Noor, A. S. Yahaya, N. A. Ramli, and A. M. M. A. Bakri, "Filling Missing Data Using Interpolation Methods: Study on the Effect of Fitting Distribution," *Key Engineering Materials*, vol. 594-595, pp. 889–895, 2014, publisher: Trans Tech Publications Ltd. [Online]. Available: https://www.scientific.net/KEM.594-595.889

[54] S. Moritz and T. Bartz-Beielstein, "imputeTS: Time Series Missing Value Imputation in R," *The R Journal*, vol. 9, no. 1, p. 207, 2017. [Online]. Available: https://journal.r-project.org/archive/2017/RJ-2017-009/index.html

[55] Z. Ding, G. Mei, S. Cuomo, Y. Li, and N. Xu, "Comparison of Estimating Missing Values in IoT Time Series Data Using Different Interpolation Algorithms," *International Journal of Parallel Programming*, vol. 48, no. 3, pp. 534–548, Jun. 2020. [Online]. Available: https://doi.org/10.1007/s10766-018-0595-5

[56] H. Ahn, K. Sun, and K. Pio Kim, "Comparison of Missing Data Imputation Methods in Time Series Forecasting," *Computers, Materials & Continua*, vol. 70, no. 1, pp. 767–779, 2022. [Online]. Available: https://www.techscience.com/cmc/v70n1/44403

[57] A. Zhang, S. Song, J. Wang, and P. S. Yu, "Time series data cleaning: from anomaly detection to anomaly repairing," *Proceedings of the VLDB Endowment*, vol. 10, no. 10, pp. 1046–1057, Jun. 2017. [Online]. Available: https://dl.acm.org/doi/10.14778/3115404.3115410

[58] X. Wang and C. Wang, "Time Series Data Cleaning: A Survey," *IEEE Access*, vol. 8, pp. 1866–1881, 2020, conference Name: IEEE Access. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8943205

[59] B. D. Fulcher, "Feature-Based Time-Series Analysis," in *Feature Engineering for Machine Learning and Data Analytics*. CRC Press, 2018, num Pages: 30.

[60] B. D. Fulcher and N. S. Jones, "hctsa: A Computational Framework for Automated Time-Series Phenotyping Using Massive Feature Extraction," *Cell Systems*, vol. 5, no. 5, pp. 527–531.e3, Nov. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405471217304386

[61] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)," *Neurocomputing*, vol. 307, pp. 72–77, Sep. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218304843

[62] J. Van Der Donckt, J. Van Der Donckt, E. Deprost, and S. Van Hoecke, "tsflex: flexible time series processing & feature extraction," Dec. 2021, arXiv:2111.12429 [cs, eess, stat]. [Online]. Available: http://arxiv.org/abs/2111.12429

[63] C. H. Lubba, S. S. Sethi, P. Knaute, S. R. Schultz, B. D. Fulcher, and N. S. Jones, "catch22: CAnonical Time-series CHaracteristics," Jan. 2019, arXiv:1901.10200 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1901.10200

[64] T. Henderson and B. D. Fulcher, "An Empirical Evaluation of Time-Series Feature Sets," Oct. 2021, arXiv:2110.10914 [cs]. [Online]. Available: http://arxiv.org/abs/2110.10914

[65] A. Renault, A. Bondu, V. Lemaire, and D. Gay, "Automatic Feature Engineering for Time Series Classification: Evaluation and Discussion," in *2023 International Joint Conference on Neural Networks (IJCNN)*, Jun. 2023, pp. 1–10, arXiv:2308.01071 [cs]. [Online]. Available: http://arxiv.org/abs/2308.01071

[66] L. Von Ahn and L. Dabbish, "Labeling images with a computer game," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Vienna Austria: ACM, Apr. 2004, pp. 319–326. [Online]. Available: https://dl.acm.org/doi/10.1145/985692.985733

[67] P. G. Ipeirotis, "Analyzing the Amazon Mechanical Turk marketplace," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 17, no. 2, pp. 16–21, Dec. 2010. [Online]. Available: https://doi.org/10.1145/1869086.1869094

[68] A. Sorokin and D. Forsyth, "Utility data annotation with Amazon Mechanical Turk | IEEE Conference Publication | IEEE Xplore," Jul. 2008. [Online]. Available: https://ieeexplore.ieee.org/document/4562953

[69] B. Plank, "The "Problem" of Human Label Variation: On Ground Truth in Data, Modeling and Evaluation," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 10 671–10 682. [Online]. Available: https://aclanthology.org/2022.emnlp-main.731

[70] C. Zhou, M. Prabhushankar, and G. AlRegib, "On the Ramifications of Human Label Uncertainty," Nov. 2022, arXiv:2211.05871 [cs]. [Online]. Available: http://arxiv.org/abs/2211.05871

[71] D. Q. Sun, H. Kotek, C. Klein, M. Gupta, W. Li, and J. D. Williams, "Improving Human-Labeled Data through Dynamic Automatic Conflict Resolution," Dec. 2020. [Online]. Available: https : / / machinelearning . apple . com / research / improving-human-labeled-data

[72] E. Eldele, M. Ragab, Z. Chen, M. Wu, C.-K. Kwoh, and X. Li, "Label-efficient Time Series Representation Learning: A Review," Aug. 2023, arXiv:2302.06433 [cs]. [Online]. Available: http://arxiv.org/abs/2302.06433

[73] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, Feb. 2017, conference Name: Journal of Systems Engineering and Electronics. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7870510

[74] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, Jul. 2019. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1

[75] W. Yu, I. Y. Kim, and C. Mechefske, "Analysis of different RNN autoencoder variants for time series classification and machine prognostics," *Mechanical Systems and Signal Processing*, vol. 149, p. 107322, Feb. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327020307081

[76] A. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré, "Training Complex Models with Multi-Task Weak Supervision," Dec. 2018, arXiv:1810.02840 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1810.02840

[77] S. Khattar, H. ODay, P. Varma, J. Fries, J. Hicks, S. Delp, H. Bronte-Stewart, and C. Re, "Multi-frame Weak Supervision to Label Wearable Sensor Data," *Proceedings of the 36 th International Conference on Machine Learning, Long Beach, California, PMLR 97*, 2019.

[78] Y. Wang, H. Wang, Y. Shen, J. Fei, W. Li, G. Jin, L. Wu, R. Zhao, and X. Le, "Semi-Supervised Semantic Segmentation Using Unreliable Pseudo-Labels," 2022, pp. 4248–4257. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2022/html/Wang_Semi-Supervised_Semantic_Segmentation_Using_Unreliable_Pseudo-Labels_CVPR_2022_paper.html

[79] M. González, C. Bergmeir, I. Triguero, Y. Rodríguez, and J. M. Benítez, "Self-labeling techniques for semi-supervised time series classification: an empirical study," *Knowledge and Information Systems*, vol. 55, no. 2, pp. 493–528, May 2018. [Online]. Available: https://doi.org/10.1007/s10115-017-1090-9

[80] A. Abanda, U. Mori, and J. A. Lozano, "A review on distance based time series classification," *Data Mining and Knowledge Discovery*, vol. 33, no. 2, pp. 378–412, Mar. 2019. [Online]. Available: https://doi.org/10.1007/s10618-018-0596-4

[81] D. Spathis, I. Perez-Pozuelo, L. Marques-Fernandez, and C. Mascolo, "Breaking away from labels: The promise of self-supervised machine learning in intelligent health," *Patterns*, vol. 3, no. 2, p. 100410, Feb. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2666389921002841

[82] V. Rani, S. T. Nabi, M. Kumar, A. Mittal, and K. Kumar, "Self-supervised Learning: A Succinct Review," *Archives of Computational Methods in*

Engineering, vol. 30, no. 4, pp. 2761–2775, May 2023. [Online]. Available: https://doi.org/10.1007/s11831-023-09884-2

[83] "Machine Learning Labeling - Amazon SageMaker Ground Truth - AWS." [Online]. Available: https://aws.amazon.com/sagemaker/groundtruth/

[84] "Label Studio Open Source Data Labeling." [Online]. Available: https://labelstud.io/

[85] "Prodigy · Prodigy · An annotation tool for AI, Machine Learning & NLP." [Online]. Available: https://prodi.gy/

[86] "Annotate | Labelbox." [Online]. Available: https://labelbox.com/product/annotate/

[87] "Rubrix: Labeling." [Online]. Available: https://rubrix.readthedocs.io/en/stable/reference/python/python_labeling.html

[88] "Visplore – Software for Visual Time Series Analysis." [Online]. Available: https://visplore.com/home-11-2022/

[89] "TRAINSET." [Online]. Available: https://trainset.geocene.com/

[90] P. Varma, "Reef : Automating Weak Supervision to Label Training Data," 2018. [Online]. Available: https://www.semanticscholar.org/paper/Reef-%3A-Automating-Weak-Supervision-to-Label-Data-Varma/cf9e68cc5c37d9fe151e0e7ebc98bbe1067297a9

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[92] L. Roque, "Probabilistic Logistic Regression with TensorFlow," Jan. 2023. [Online]. Available: https://towardsdatascience.com/probabilistic-logistic-regression-with-tensorflow-73e18f0ddc48

[93] M. Kana, "Introduction to Bayesian Logistic Regression," Feb. 2020. [Online]. Available: https://towardsdatascience.com/introduction-to-bayesian-logistic-regression-7e39a0bae691

[94] "Davis® AI," Dec. 2023. [Online]. Available: https://docs.dynatrace.com/docs/platform/davis-ai

[95] "tsfresh — tsfresh 0.20.1.post0.dev7+ge2dfc6f documentation." [Online]. Available: https://tsfresh.readthedocs.io/en/latest/

[96] "pycatch22 - CAnonical Time-series CHaracteristics in python," Dec. 2023, original-date: 2022-07-05T03:58:34Z. [Online]. Available: https://github.com/DynamicsAndNeuralSystems/pycatch22

[97] S. Matzka, "Condition Monitoring Dataset (AI4I 2021)." [Online]. Available: https://www.kaggle.com/datasets/stephanmatzka/condition-monitoring-dataset-ai4i-2021

[98] "DJIA 30 Stock Time Series." [Online]. Available: https://www.kaggle.com/datasets/szrlee/stock-time-series-20050101-to-20171231

[99] M. Malekzadeh, "MotionSense Dataset : Smartphone Sensor Data - HAR." [Online]. Available: https://www.kaggle.com/datasets/malekzadeh/motionsense-dataset

[100] S. Matzka, J. Pilz, and A. Franke, "Structure-borne and Air-borne Sound Data for Condition Monitoring Applications," 2021, pp. 1–4. [Online]. Available: https://www.kaggle.com/datasets/stephanmatzka/condition-monitoring-dataset-ai4i-2021

[101] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi, "Mobile sensor data anonymization," in *Proceedings of the international conference on internet of things design and implementation*, ser. IoTDI '19. New York, NY, USA: ACM, 2019, pp. 49–58, number of pages: 10 Place: Montreal, Quebec, Canada tex.acmid: 3310068 cites: MotionSenseDatasetSmartphonea. [Online]. Available: http://doi.acm.org/10.1145/3302505.3310068

[102] H. J. P. Weerts, A. C. Mueller, and J. Vanschoren, "Importance of Tuning Hyperparameters of Machine Learning Algorithms," Jul. 2020, arXiv:2007.07588 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2007.07588

[103] P. Probst, A.-L. Boulesteix, and B. Bischl, "Tunability: Importance of Hyperparameters of Machine Learning Algorithms," *Journal of Machine Learning Research*, vol. 20, no. 53, pp. 1–32, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-444.html

[104] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 1: turning lemons into lemonade," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 6, pp. 16–18, Nov. 2001. [Online]. Available: https://dl.acm.org/doi/10.1145/505532.505535

[105] ——, "Principles of survey research: part 2: designing a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 1, pp. 18–20, Jan. 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/566493.566495

[106] ——, "Principles of survey research: part 3: constructing a survey instrument," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 2, pp. 20–24, Mar. 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/511152.511155

[107] ——, "Principles of survey research: part 4: questionnaire evaluation," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 3, pp. 20–23, May 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/638574.638580

[108] ——, "Principles of survey research: part 5: populations and samples," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 5, pp. 17–20, Sep. 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/571681.571686

[109] ——, "Principles of survey research: part 6: data analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 2, pp. 24–27, Mar. 2003. [Online]. Available: https://dl.acm.org/doi/10.1145/638750.638758

[110] A. N. Ghazi, K. Petersen, S. S. V. R. Reddy, and H. Nekkanti, "Survey Research in Software Engineering: Problems and Mitigation Strategies," *IEEE Access*, vol. 7, pp. 24 703–24 718, 2019, conference Name: IEEE Access. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8533319

[111] C. Spearman, "The Proof and Measurement of Association between Two Things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904, publisher: University of Illinois Press. [Online]. Available: https://www.jstor.org/stable/1412159

[112] R. Feldt and A. Magazinius, "Validity Threats in Empirical Software Engineering Research - An Initial Survey." Redwood City, San Francisco Bay, CA, USA, Jan. 2010, pp. 374–379.