Why did the prediction change? Explaining changes in predictions as time progresses

by

Wei-En Warren Wang

S.B. Electrical Engineering and Computer Science, Physics
Massachusetts Institute of Technology (2023)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by: Wei-En Warren Wang
Department of Electrical Engineering and Computer Science
January 30, 2024

Certified by: Kalyan Veeramachaneni
Principal Research Scientist
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Why did the prediction change?
Explaining changes in predictions as time progresses

by

Wei-En Warren Wang

Submitted to the Department of Electrical Engineering and Computer Science
on January 30, 2024, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Few works on machine learning (ML) explanations design explanations from the perspective of model deployment in the real-world. This work addresses the challenges of understanding ML models applied to event-based time-series data, concretizes two explanation scenarios, and proposes explanations based on changes in feature values, model predictions, and feature contributions for each deployment scenario. We study the prediction problem of turbine brake pad failures, where predictive time-series ML models were deployed in production. Our solution to help decision makers understand how the predictions are made include the development of a usable ML interface and explanations that are aware of the scenarios and contexts where the models are being used. We discuss the usage of ML explanations and the importance of the context under which the model is deployed. We showed our usable ML interface and the explanations with their corresponding scenarios built on top of the usable ML system, which consists of Pyreal, Sibyl-API, and Sibylapp.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist

Acknowledgments

The exposition of usable ML systems and the description wind turbine brake pad prediction problem are from joint works with Alexandra Zytek et al. [23, 24].

First and foremost, I would like to thank my thesis advisor, Kalyan Veeramachaneni, for guiding me and offering valuable feedback throughout the entire research process. His expertise and ideas were crucial in shaping the direction of my work. I have learned many lessons and ways of thinking about research from him, and I am very grateful to have him as an advisor.

I'm also incredibly thankful to my research mentor, Alexandra Zytek, for her support and contributions to this project. She is an exceptional mentor professionally and one of the most friendly person I have met. Her encouragement and guidance was an essential part of my research development.

I would like to express my gratitude to the Iberdrola team, especially Sofia Koukoura, for her feedback on the system and explanations we developed. This project would not have been possible without their guidance. My thanks also go to my wonderful labmates in the Data to AI Lab. Everyone is extremely welcoming and I am blessed to become a part of this family.

Lastly, I want to express my deepest thanks to my family and friends. Their unwavering love and encouragement sustains me throughout my academic journey.

The realization of this thesis wouldn't have been possible without the collective support and contributions of these amazing people. Thank you all for being an essential part of this adventure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Applications of predictive artificial intelligence (predictive AI) are everywhere in our lives. Engineers, data scientists, decision-makers, and many other people use machine learning (ML) predictions to help their work in all kinds of domains. As the popularity of complex ML models increases, ML researchers started the development of ML Explainability methods in hope of understanding the increasing number of ML models in both academia and industry. Most researchers focus on developing the "best" explanation algorithms and argue their performance through metrics that follow some formulae, but these metrics were typically designed for computer scientists to compare between different algorithms and benchmark purposes. Such evaluation does not concern how to help non-ML experts to be more comfortable using their ML models. In most real-world predictive AI application scenarios, people care about how to make better use of predictive AI. A good ML explanation must be able to present information that is easy to understand and actionable for humans [10, 13, 22]. To this end, we see the term *usable ML* more fitting than the commonly used *explainable AI (XAI)* because understanding ML models requires much more than just explanations [23]. It also requires understanding the context of the prediction problem and the scenario where the model is being used, which is nearly impossible to achieve without collaboration between computer scientists and subject matter experts.

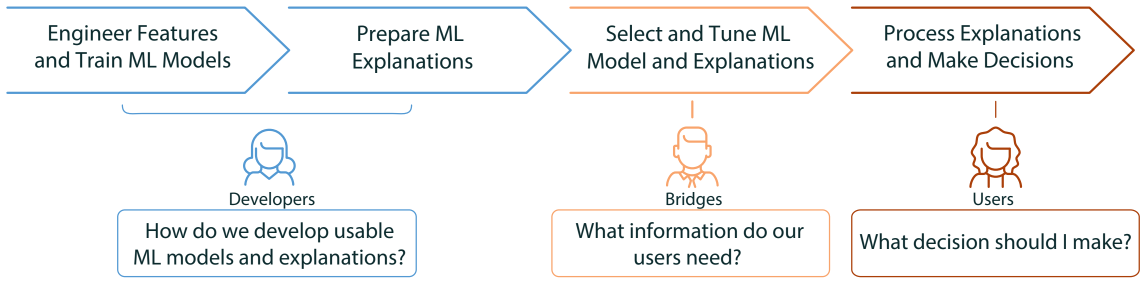
1.1 Predictive AI in Practice

In this thesis, we craft explanations considering the application scenarios of predictive AI models developed by wind turbine data engineers. Their modeling uses event-based time-series data. This type of data is pervasive in many domains, such as the viewership behavior of a streaming platform over a week or the data collected daily from various sensors in the wind turbine. Event-based time-series data often contain multiple data types and can span over different time scales.

To make use of such diverse data, people have developed systems that applied feature engineering techniques to formulate prediction problems from event-based time-series [8]. Our collaborators – data engineers – have successfully built models with their expertise and tools provided by Zephyr [8]. These models perform well in terms of common accuracy scores, but the engineers did not know much about the models beyond that. When using these models in production, the decision makers may have various questions about the predictions made by a model, such as why does this model change its prediction drastically over the weekend. These models have high performance, but the turbine engineers using them may have a hard time deciphering their logic. They may have a hard time answering questions like "why did the model's prediction change so drastically over the weekend". To answer such questions, we need input from data engineers to help frame the explanation in a scenario that makes sense to the data engineers who relay the explanation to the decision makers. Clearly, we need people with different expertise to help deploy predictive AI into production. This realization has brought us to usable ML [24].

1.2 Usable ML for Event-Based Time-Series

Through our work in usable ML deployment, we have identified three key roles [23]: **developers**, **bridges**, and **users**. As shown in Figure 1-1, the developers are in charge of establishing the ML models, workflow, and tools used in the workflow that satisfies the needs of the users. The users have domain expertise and know what



Engineer Features and Train ML Models
Prepare ML Explanations
Select and Tune ML Model and Explanations
Process Explanations and Make Decisions
Developers
How do we develop usable ML models and explanations?
Bridges
What information do our users need?
Users
What decision should I make?

Figure 1-1: Key roles in usable ML deployment. ML developers and users define the ML system and problem. Bridges enable smoother collaborations throughout the multiple iterations of development required for a usable ML interface.

problems are worth solving in their respective fields, and most importantly, they also make decisions to address these problems. A useful ML system requires a meaningful problem and a suitable ML workflow; therefore, it requires the strengths of both developers and users. The bridges serve as a connection between the developers and users. We as developers, specifically usable ML system developers, need to establish a usable interface through close collaboration with data engineers (bridges) to provide explanations that inform decision makers (users). It is also through this collaboration that we learned the important problems in turbine engineering.

1.3 Explanation through Temporal Variations

Our conversations with bridges have informed us about the challenges in understanding event-based time-series models. Working with such data requires sophisticated processing due to their complexity. With so much complexity already present in the data itself, it is even more difficult to understand models built with such data if we attempt to explain these models holistically. Ultimately, what users care about is to know what specifically causes the predictions of a ML system to change over time. Developers and bridges answer this question from a problem-solving perspective.

Our goal is to make understandable explanations for ML models that do not overwhelm the bridges and users. We propose an algorithm that explains event-based time-series ML models by focusing on the **changes** between data and models' predictions at different time points. The hypothesis behind our design is similar to the premise of counterfactual explanations [16, 21]. From a problem solving perspective, telling people "how to change the model to make it better" is much more useful than telling people "why your model is failing". Highlighting the contrasts between instances can tell people "this is what the prediction will be if you change the data like this". In addition, focusing on the variations between instances allows us to be aware of the decisive variables without understanding every variable in the problem.

When designing explanation algorithms, the objective of the explanations must be clear. It can be finding out what are the most significant features. It can also be discovering a pattern in the data that causes certain types of outcome to occur. These objectives vary widely depending on the users' interests, so there is not a single explanation algorithm that can solve all ML explainability problems. Thus, it is important to identify the most suitable algorithm that helps people solve the specific problem at hand. We carried this philosophy as we applied our ML explanations to the domain of wind turbine monitoring. During this case study, we develop our explanations hand-in-hand with improving the task of predicting the failure of wind turbine brake pads. We realized there were several questions turbine engineers and data analysts want answered by their ML prediction models, which we contextualized into three **explanation scenarios**. These scenarios define the different ways that ML models are used in deployment. Identifying the explanation scenarios is essential to find the most suitable algorithm to generate the explanations.

1.4 Contributions

This thesis presents the first step to realizing usable ML practices for event-based time-series data. The work includes understanding the context where models are deployed in production, explanations that explain the changes at different time points, and a usable system that is developed for the wind turbine brake pad prediction model but can also be an extensible framework for all predictive AI workflows. To summarize, the contributions of this thesis are:

- ML deployment scenarios and explanation algorithms designed for predictive

AI using event-based time-series data.

- An explanation algorithm that explains event-based time-series models through the changes in feature values, model predictions, and feature contributions.
- A usable interface that focus on the seamless deployment of models for predictive AI tasks: from visualizing model predictions to learning from feature contributions.

1.5 Thesis Organization

This thesis is organized as the following. Chapter 2 lays down the foundations, relative concepts, and related works that inspired this project. Chapter 3 presents the explanation problem and our proposed explanation scenarios in an abstract context with depictions through the brake pad failure prediction problem. Chapter 4 describes the entire process of and data engineering and developing model workflows that corresponds to the scenarios in Chapter 3. Chapter 5 introduces our explanation algorithm that captures the changes in these models. Chapter 6 describes the details of the application where we implemented the explanation scenarios and algorithms proposed in previous chapters. Finally, chapter 7 summarizes the conclusions and details future directions of this project.

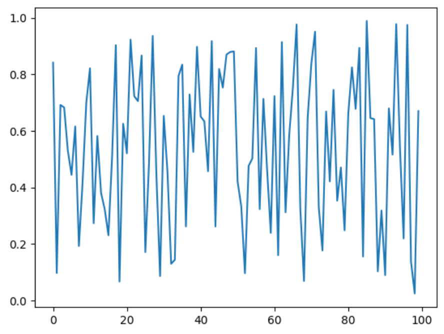
Chapter 2

Background and Related Work

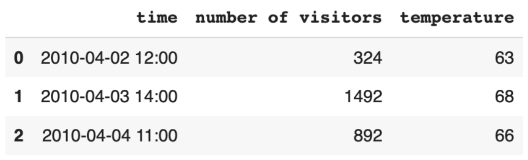
This chapter introduces the concepts that are helpful to understand the context of the problem and our contributions of this thesis. We first talk about time-series data and relevant terminology. Next, we introduce ML explanations and central concepts largely taken from [16]. Finally, we discusses related works that apply explanations to time-series modeling.

2.1 Time-Series Data

There are several typical formulations of time-series ML problems. The first type of time-series data is signal-based. Examples include stock prices, daily temperature, and ECG signals. Signal-based time-series data refers to a single signal (univariate) or multiple signals (multivariate) that take values at time steps. We can denote time-series signal as $S[1:T]$, where the signal S consists of T time steps, or $S_i[1:T]$ for $i = 1, 2, ..., n$ if the dataset contains n signals. Signal-based data are mainly used in signal processing techniques and sequence models such as Fourier transform and LSTM. As a result, analysis of signal-based data usually focuses on the sequential nature of the signals and overlooks the semantics of time. Deep learning models such as LSTM and transformer-based models are examples of technique that ignores the information about the time steps themselves.



(a) Example of signal-based data.



	time	number of visitors	temperature
0	2010-04-02 12:00	324	63
1	2010-04-03 14:00	1492	68
2	2010-04-04 11:00	892	66

(b) Example of event-based data.

Figure 2-1: Illustration of signal-based and event-based time-series data.

2.1.1 Event-Based Time-Series

Another formulation of time-series problems is event-based. A common format for this formulation is the object notation with properties of timestamp information. Each entry of the object includes the timestamp of the event and values of each variable for that event. The timestamps are recorded as-is and are not necessarily evenly-spaced. In the case where all timestamps are evenly-spaced, data then could be represented in the signal-based form. The event-based formulation is more rich in terms of information and would be our main choice of formulation for the rest of this work. Figure 2-1 shows examples of the two time-series data format.

2.2 Importance of ML Explainability

It is important to understand when we need Explainable ML. The necessity of explanations depends on the people using the model to make decisions. In low-stakes decision environments, the predictions of a model might not require explanation if it performs well on the testing dataset or in production. However, when more risk is involved, it becomes important to understand why a model makes a prediction, especially a wrong prediction, and how can we fix the problem [16]. In addition, while complex algorithms might achieve remarkable accuracy, understanding their decision-making processes is crucial for broader acceptance and deployment [5, 19]. For instance, the inability to explain the reasoning behind an AI system's medical diagnosis could hinder its adoption. Moreover, the lack of transparency and interpretability in ML models can lead to biases and unfair outcomes, especially in high-stakes domains such as finance and healthcare.

2.3 Types of ML Explanations

To address the demand for ML explainability, researchers have developed various explanation methods. These methods can be classified into two broad categories: intrinsic and post hoc explanation methods [16]. Intrinsic ML explainability is achieved through ML models with simple or intuitive structures, such as decision trees or linear models. Users of these models can easily understand the decision-making process of the models by examining the rules or coefficients used in the models. However, many impactful, real-world problems require more complex models to achieve higher predictive accuracy, which often sacrifices intrinsic explainability.

On the other hand, post hoc explainability refers to explanation algorithms that provide explanations post-training by inspecting model predictions. Post hoc explanations can be generated while treating the ML model as a black box, which means that we have no information on the internal equations of the model. Such post hoc explanation methods are called **model-agnostic** methods. Some other post hoc methods are model-specific, meaning they could only be applied to certain types of models. An example of a model-specific explanations is Saliency Map [20], which can be only used for model architectures that support differentiation.

We are more interested in model-agnostic methods for their broad applicability since the **users** of our explanations often do not care much about what model structure is used under the hood. Another benefit of using model-agnostic methods is that the explanation framework stays the same even when the **developers** choose to use a different type of models to solve the predictive AI task.

2.3.1 Local Explanations

Regarding the scope of the explanations, ML explanations can be either global or local [16]. Global explanations describe models holistically, which means understanding the dynamics between model predictions and all of the features under all conditions. With global explanations, user can point to any prediction made by the model and say what components of the model and what features in the data instance that causes the prediction. However, it will be very difficult for humans to perceive such comprehensive information when there were a myriad of features, not to mention the high-dimensional interactions between them. This is also why an explanation algorithm that can perfectly explain any model is likely nonexistent simply because humans cannot process a perfectly detailed explanation [4].

While not perfect, local explanations are easier to develop and often much more useful than global explanations. Local explanations look at the model prediction for a single instance and analyze the behavior of the model when the input goes under perturbation [13]. Since local explanations narrow the attention down to a single prediction, it is often more feasible to describe the interaction between features and the prediction with a simple rule regardless of the model's behavior for other instances. The resulting explanation can be understandable to humans and faithful to the model and data at the same time, but only for a small subset of data. This explanation cannot represent the model's general behavior.

2.3.2 Feature Attribution methods

Feature attribution is a common type of explanation, implementations of this include algorithms such as SHAP[14] and LIME[18]. They explain machine learning models by analyzing the given model's behavior when the input data goes under perturbation. The explanation generated by SHAP and LIME assigns a value to each feature, which represents the "contribution" of that feature to the prediction task. Most such explanation methods focus on tabular data.

2.4 Related Work

There exist some works that try to tackle the problem of time-series classification explanation, but most of them focus on signal-based data [2, 6, 7]. These methods explain signal-based time-series data To our knowledge, few work has been done to systematically explain the usage of models in different temporal contexts, which is very relevant in the actual deployment of ML models. Some works have attempted to understand ML models similar through the use of changes [9, 17]. The work of [15] proposed to attribute evolution of the feature contributions to risk of hypoxaemia.

Chapter 3

Problem: Explaining Changing Predictions

ML explanations are necessary because of ML model users, who are trying to solve real-world problems. Hence, an explanation is most effective when it is tailored to the specific context and requirements of the problem at hand. This chapter introduces turbine brake pad failure prediction as a predictive AI task, and discusses the kind of context and explanations that will answer users' questions when a model is deployed. It then introduces and formulates abstract problem scenarios of explanation of changes in predictions over time.

Table 3.1: Different teams involved in the usable ML system designed for wind turbine brake pad failure prediction and their roles.

Roles	Developers	Bridges	Users
Teams	Our team	M&A team	O&M team

3.1 Example Prediction Problem: Wind Turbine Brake Pad Failure Prediction

One major failure mode of wind turbines is caused by the premature breakdown of brake pads that help control the operation of the turbines. Usually, this type of failure is prevented by sending technicians to investigate and repair the brake pads if necessary. This is a risky and costly task; however, if the brake pads do break down causing the wind turbine to fail, the repair would be an even more expensive and time-consuming operation even without considering the energy loss incurred by the failure. Ideally, users want to ensure that each visit to the wind turbines is necessary to maintain smooth operation, but how do users determine when to send people up to the turbines? Is there a better strategy than performing routine monthly checks on these turbines?

To minimize cost and ensure the smooth operation of turbines, this problem can be formulated an ML problem that predicts when the brake pads will fail. The data used for prediction is comprised of signals from sensors such as temperature and pressure sensors in the wind turbines, which come in the format of time-series signals. The ML model uses the data to generate the probability of brake pad failure at a given prediction time: for example, predicting if the brake pad will fail a day or a week from now.

So far, our team has built models that make reliable predictions, but this is not enough (see Chapter 4 and Appendix A for our models' performances) [1]. The decision makers (users) must trust these models, understand their outputs in order for them to use their output in decision making. It is important to note that many of these problems where machine learning model output/prediction is being considered, classical approaches to data analysis already exist and/or decision makers already have rule based solutions that they use.

[1]Our team here includes a broader machine learning model development team at Data to AI Lab at MIT

3.1.1 Decision making workflow and roles people play

Before designing explanation interface, it is important to consider the decision making workflow and what roles people play in it. The type of explanations that are designed (or selected from existing techniques) depends on who is seeking it. To further concretize these for our brake pad prediction case study, Table 3.1 defines the roles of the different teams involved in our usable ML system.

In prior work summarizing lessons from our collaborations with wind turbine monitors, we have detailed the roles of the teams involved in the development of our usable ML system [23]. In the wind turbine domain, the Monitoring and Analysis (M&A) team fills the bridge role, specializing in ML/data science applied to wind turbine monitoring[2]. This team kicks off the decision-making process at hand by identifying a problem in the live data with help from the ML model. They then compile a summary of relevant information and visualizations about the issue (for example, “turbine 50’s brakepad is predicted to fail because of an increase in the brake caliper temperature”).

The M&A team communicates their findings with the Operations and Maintenance (O&M) team, providing them with the compiled summary and explanations. The O&M team, the main users of the interface, look through this information and make a decision about how to proceed. If the issue poses a significant risk, the O&M team informs the site teams at the wind farm(s) in question about the issue and the suspected cause. The site teams may either fill the role of user (if they also review the model prediction and usable ML interfaces) or affected party (if they carry out the O&M team’s suggestions directly). They look into the issue on-site, potentially reaching out to a contracted party to handle repairs.

As the O&M team is not expected to have much knowledge about ML, we cannot assume that they will understand explanations that focus on models’ attributes. They care about the predictions of the models, but may not be as interested in the engineering decisions such as hyper-parameter tuning and model selections that lead to those predictions. We built prototypes for explanations and communicated with the

[2]Descriptions of wind turbine teams in this and the following two paragraphs are taken from [23].

M&A team to understand how to tune our explanations in a way that appeals to the O&M team. Through our discussions, we learned that the O&M team is interested in two main types of question:

1. Is the brake pad going to fail at time t? What is the explanation for this prediction?
2. What are the chances of the brake pad experiencing failure over the next couple of days (or weeks, or months)?
3. When predictions are made over different time horizons, and there is change in prediction, what changes in the underlying data explain this?

The first two questions require building a machine learning model and explaining it using classical techniques. The third question, is a variation, where either new explanation techniques could be designed or existing techniques could be used to explain the change in the predictions. This is the focus of this thesis.

3.2 Terminology we use in this thesis

To address the formulation of temporal prediction problems, we define relevant notations in Table 3.2. Notations of time-related properties are taken from [11]. From a user's perspective, the most meaningful variables are the cut-off time point (t_c) and the `lead` time. In deployment, the first refers to the time when the prediction is made, and the second refers to how far ahead of time the prediction is made with respect to t_c.

An `entity` is the item for which the prediction is being made. For example, this can be the data for "turbine (brake pad) 4," for which the feature values that include information and measurements for multiple time steps can be extracted.

The terminology in Table **??** can be used both for data and processing in training and deployment. It is important to note that explanations are sought in the deployment. In the next two paragraphs we delineate how this terminology is used in the two phases.

Table 3.2: Notations and definitions for the problem statement.

Notation	Definition
`entity`	The entity for which we are making the prediction. It has time series and event based data associated with it for making the prediction; e.g., “turbine 4”.
t_c	Cut-off time point. In training: No data after this time point is used to train the model. In deployment: This is the time point when a future prediction is made. (note that by definition the future data is not available at this time point)
t_{target}	Target time point. The time point for which the prediction is made.
`lead`	The time delta between t_{target} and t_c. $t_{target} = t_c + \texttt{lead}$.
X_{t_c}	The input data instance or feature vector. Subscript t_c denotes the cut-off time.
$Y_{t_{target}}$	The prediction made by the model. This is usually a singular variable unless the prediction has multiple values. Subscript t_{target} denotes the target time point.
f	The model function that produces prediction Y given input X.

Training process. Since one `entity` contains feature values of different times, multiple data instances X can be formed from a single `entity` through feature engineering. Note that X and Y take different values at different time points. We denote such values as X_t and Y_t, where t is the time point of interest. Many time-series problems involve creating models that predict future events. In these cases, the input and target variables will be sampled at different times. The input variables are sampled at the cut-off time t_c, and the target variables are sampled at the target time t_{target}. Therefore, the model function f is trained to be a mapping from X_{t_c} to $Y_{t_{target}}$. For training a model, if multiple brake pad failures occurred for the same entity, multiple training examples can be extracted from it. We will explain this in more detail in the next chapter, along with training data generation.

In deployment. During deployment, at any time point t_c a feature vector is computed and a prediction is made for the t_{target} time point, denoted as $Y_{t_{target}}$. Note that the real value of $Y_{t_{target}}$ is not available, as it has not occurred yet.

Now, we can reframe the two questions in the previous section:

1. "Is the brake pad going to fail at time t_{target}? What is the explanation for this prediction?" becomes: What is $Y_{t_{target}}$ and the explanation for this prediction?

2. "What are the chances of the brake pad experiencing failure over the next couple of days (or weeks, or months)?" becomes: What is $Y_{t_{target}}$ for $t_{target} = t_c + \Delta_1$, $t_c + \Delta_2$,... and the explanations for these predictions, where Δ_i refers to different `lead` times of interest?

3.3 Problem Scenarios

Figure 3-1 illustrates the following two scenarios.

3.3.1 Scenario 1: Prediction at a Specific Time Point

In this scenario, the user is interested in what will happen at a specific time point t_{target} for a chosen entity. To understand the reasoning behind such predictions, the user may also be interested in the difference between predictions made at two separate time points. "*What changed between t_{c_1} and t_{c_2} that led to the change in the prediction?*"

3.3.2 Scenario 2: Predictions at Different Times in the Future

In this scenario, the user is interested in the difference in predictions at different future time points for a chosen entity. It is for fixed t_c but with different `lead`. This is interesting because it provides insights about the evolution of the prediction problem overtime. "*How did you use the same data differently to make predictions for different* `lead| times`?"

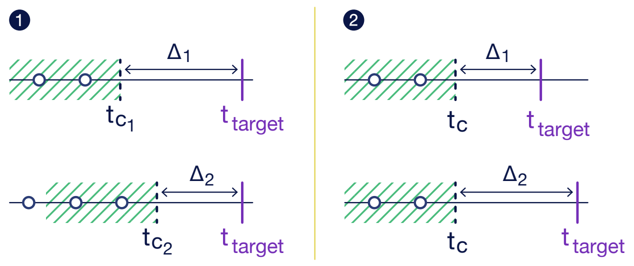


Figure 3-1: Illustration of the two deployment scenarios for our prediction task our users are interested in. The first scenario concerns the predictions made at different target times (t_{target}). The second scenario concerns the predictions made for different lead times but with the same cut-off time (t_c).

Chapter 4

Training the Machine Models to Support the two Scenarios

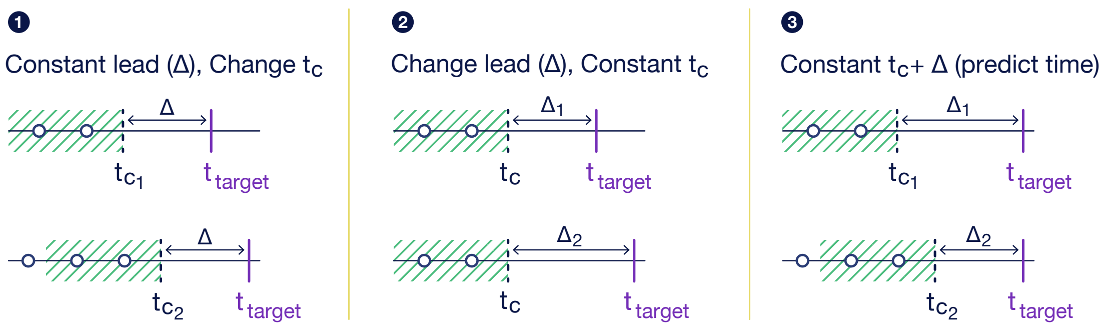
To address the two scenarios presented in the previous chapter, we need to train one or more ML models that can generate the predictions the users are looking for. The models and accompanying data are then provided as inputs to explanation algorithms and interfaces we intend to design to explain to the users "*why did the predictions change?*". In this chapter, we describe this data and model preparation, using the "brake pad failure prediction" use-case as an example.

4.1 Modeling Context for Problem Scenarios

	cut-off time (t_c)	`lead` (Δ)	prediction time (t_{target})
Context A	Dynamic	Fixed	Dynamic
Context B	Fixed	Dynamic	Dynamic
Context C	Dynamic	Dynamic	Fixed

Table 4.1: Configuration of adopted contexts.

In the previous section, we introduced two common scenarios featuring temporal predictions that the users are interested in. We as developers build ML workflows and models to generate predictions for those scenarios. These two scenarios require multi-



1
Constant lead (Δ), Change tc
Δ
tc1
ttarget
Δ
tc2
ttarget
2
Change lead (Δ), Constant tc
Δ1
tc
ttarget
Δ2
tc
ttarget
3
Constant tc+ Δ (predict time)
Δ1
tc1
ttarget
Δ2
tc2
ttarget

Figure 4-1: This figure shows the relevant time parameters for each of the three modeling contexts. In context a, the lead time is constant. In context b, the cut-off time point is constant. In context c, the prediction time we are interested in is constant.

ple models in order to generate predictions. We discovered three different "contexts" for which machine learning models need to be trained. (We use "context," which comes from the developer's perspective, to distinguish from "scenario", which comes from the users' perspective.) Table 4.1 refers to the configuration of each parameter in a given context and Figure 4-1 depicts an illustration of the three contexts.

To train these models from historic data, one would need to extract training examples, segment data (using only data prior to the cut-off time to create features) and train a ML model. To do this, we adopted the "*label, segment, featurize*" framework proposed by [11] to formalize the problem scenario.

4.1.1 Context A: Explanation of a Single Model's Behavior for Data at Different Time Points (t_c)

Context A is a straightforward formulation. This context explains a **single** given model, with a fixed lead time, and considers the change in prediction for an entity at different time points. For example, we have an XGBoost model trained with a data set, with labels set such that the `lead` is 10 days. We can then use this model to

- make a prediction on January 10th, 2024 of whether the break pad will fail on January 20th, 2024 (10 days ahead).
- make a prediction again on Janaury 17th, 2024, of whether the break pad will

fail on January 27^{th}, 2024 (10 days ahead).

We are interested in how the model's prediction changes when more data has been collected, and at different time points. The underlying time-series data used to generate explanations for each prediction will have a different t_{start} and t_{stop}. Studying with this model allows us to pinpoint when and how a single time-series model changes its prediction with respect to changes in the input data.

Most implementations of explanations for machine learning models fall into this category. For example, a local feature attribution method can be used to produce a contribution each time a prediction is made. In our case, the input to the brake-pad failure model goes through feature engineering pipelines that transform multiple signals into tabular features. Therefore, each different time point within the input data can be viewed as a distinct data instance. This concept is not limited to time-series problems since this formulation only requires the usage of different data instances, to which we then append the time points, each as a "property" of its instance. Within this context, the explanation would be the comparison of the model's predictions and its corresponding feature contributions for the selected time points.

4.1.2 Context B: Explanation of Predictions over Different Lead Times with the Same Cut-off Time

Context B implements Scenario 2 in Chapter 3 by creating a model for each prediction with different `lead` times. In many temporal prediction cases, people are interested in how a model's prediction varies when it is asked to make predictions for different time points in the future.

In Context B, users are interested in viewing how the confidence of the model changes as it starts to predict events that are farther away in the horizon. A major difference between this context and Context A is that here we are explaining multiple different models instead of a single model. This is due to our design that the prediction of a single model only includes a single value. The predictions at different `lead` times are generated by different models. More specifically, these models are of the same type but are trained for a different `lead` time.

As an example, we could train two XGBoost models, one with a `lead` time of 10 days and one with a `lead` time of 20 days (denoted as $f_{10}(\cdot)$ and $f_{20}(\cdot)$ respectively). We could then use these two models to make two predictions. Let's say these predictions are made on January 10^{th}, 2024:

- What is the prediction for January 20^{th}, 2024? This would use model f_{10}.
- What is the prediction for January 30^{th}, 2024? This would use model f_{20}.

Unlike for the previous context, in this scenario, these different models are given the **same input data** to make their predictions. We can then observe these models' series of predictions, and the corresponding feature contributions, to see how they evolve. The goal is to understand what changes when the same data is used to make predictions for different times in the future.

4.1.3 Context C: Explanation of Predictions for a Specific Time Point in the Future

Context C also works with Scenario 1, but it takes a completely different approach from Context A. Here we also use multiple models with different `lead` times, similar to Context B.

The difference is that the models used in Context B are making predictions from one t_c for different t_{target}, while the models used in Context C are making a prediction for the same t_{target} from different values of t_c. Consider the same approach, where we trained two XGBoost models, one with a `lead` time of 10 days and one with a `lead` time of 20 days (denoted as $f_{10}(\cdot)$ and $f_{20}(\cdot)$ respectively). With these two models, we are able to make the following predictions:

- On January 1^{st}, 2024, make a prediction for January 21^{st}, 2024. This uses the XGBoost model with a `lead` time of 20 days (f_{10}).
- On January 11^{st}, 2024, make a prediction for January 21^{st}, 2024. This uses the XGBoost model with a `lead` time of 10 days (f_{20}).

Note that in the example above, the date for which the prediction is made remains the same; that is, January 21st, 2024.

In this case, each model gets different input data instances. To produce a prediction at the same t_{target} but from different values of t_c, the models must be fed with different input data.

4.2 Training ML models using Zephyr

To train multiple machine learning models with different lead times for our brake pad failure prediction problem, we rely on a system built for developing prediction models for data from wind turbines, called Zephyr [8]. Zephyr is machine learning framework for predictive maintenance of wind energy data. The framework consists of a set of modules to: (1) process raw signal-data and format it into a machine readable representation; (2) apply a series of signal processing operations to extract valuable information from signals data; (3) construct prediction problem and automatically generate labels; (4) featurize data and produce a feature matrix; (5) build and train machine learning models. For the purpose of this thesis, we will be mainly focusing on modules 3-5. Training models involves generating labels, feature engineering, and ultimately training an ML model. To do this, we use the libraries `Featuretools`[1] and `Zephyr`[2]. In this chapter, we give a very brief overview of the different steps involved in training the models that support the different scenarios using `Zephyr`. For a more detailed understanding of how ML models are developed in `Zephyr`, we refer the reader to [8].

4.2.1 Raw Data

Our raw data sets contain six `csv` files. `Zephyr` provides a function that converts csv files into an `Entityset` [12]. An `Entityset` is a collection of tables (entities) and the relationships between them. An overview of the `Entityset` used for generating

[1]https://github.com/alteryx/featuretools
[2]https://github.com/sintel-dev/zephyr

```
Entityset: SCADA data
  DataFrames:
    turbines [Rows: 70, Columns: 10]
    alarms [Rows: 0, Columns: 5]
    notifications [Rows: 127, Columns: 15]
    stoppages [Rows: 16898, Columns: 18]
    work_orders [Rows: 5735, Columns: 21]
    scada [Rows: 7914960, Columns: 167]
  Relationships:
    alarms.COD_ELEMENT -> turbines.COD_ELEMENT
    stoppages.COD_ELEMENT -> turbines.COD_ELEMENT
    work_orders.COD_ELEMENT -> turbines.COD_ELEMENT
    scada.COD_ELEMENT -> turbines.COD_ELEMENT
    notifications.COD_ORDER -> work_orders.COD_ORDER
```

Figure 4-2: Content of the Entityset. Zephyr processes this data to generate a featurized data set (the input to the machine learning model) and labels based on the selection of the labeling function and specified time parameters.

training data is shown in Figure 4-2.

This `Entityset` comprises multiple tables, including:

- `turbines` which has turbine metadata;
- `alarms` which records alarm occurrences in the diagnostics system;
- `notifications` that contains the maintenance history for each turbine;
- `stoppages` which records the start– and end-time of every time the turbine stopped as well as the cause for stopping;
- `work_orders` which have information about maintainence work done on the turbines;
- `scada` which includes time-series signals.

Most of these tables are connected via an `ID` column called `COD_ELEMENT`, which uniquely identifies specific turbines.

4.2.2 Generate Labels

After the `Entityset` is created, we use the labeling function provided by `Zephyr`, namely `brake_pad_presence`, to create labels from our data. The labeling function follows the method set out in [11]. In our use cases, we will generate training sets with varying `lead` times. Different settings of this parameter lead to different problem formulations; labels generated with `lead` $= 0$ days are used for training a model that makes a prediction for the present, while those generated with `lead` $= 30$ days are used for training a model that predicts the result one month from now. For our application example, we generate data sets with the following range of `lead`: $0, 5, 10, 15, 20, 25, 30$ days. How the labeling function generates the labels given a machine learning task is detailed in [8].

It is critical to note that to train models with different lead times, the framework first finds historical examples of brake pad failure cases, then divides the historical data into two parts based on the cut-off time (t_c): the former part is used to train the models, and the latter part is used to compute the prediction label.

Using the default settings of the labeling function, we obtain 943 labels where a turbine's brake pad has failed and 1,498 labels where it has not failed. This results in a total of 2,441 labels that will be used in the subsequent parts including model training.

4.2.3 Feature engineering

`Zephyr` uses the Deep Feature Synthesis (DFS) algorithm from `Featuretools` to generate the feature matrix X_{t_c} given the `Entityset` and a list of cut-off times. In the DFS algorithm, we use the default transformation operations, and we limit the aggregation operations to include: `count`, `sum`, `percent_true`, and `max`. The most indicative table in our data for the brake pad problem is the `stoppages` entity, since it illustrates any issue that led to needing to stop the turbine from operating. Therefore, certain entities were deemed irrelevant to the prediction problem at hand by subject matter experts, and thus were excluded from the feature generation process, these

Table 4.2: Hyperparameter settings for the XGBoost models.

hyperparameter	**value**
tree_method	“approx”
enable_categorical	True
subsample	0.2
sampling_method	“uniform”
random_state	35

entities include: `notifications`, `alarms`, and `work orders`. The resulting feature matrix is completely numeric and has 318 features after those not relevant to the prediction task are removed. The first five features of X_{t_c} are `COUNT(stoppages)`, `MAX(stoppages.COD_WO)`, `MAX(stoppages.IND_DURATION)`, `MAX(stoppages.IND_LOST_GEN)`, and `SUM(stoppages.COD_WO)`.

4.2.4 Model Training

All the ML models used in our system are XGBoost classifiers [1] that predict whether the turbine brake pad is going to fail at t_c+`lead` given input X_{t_c}. We train the models until they converge. In this paper, we restricted our applications to explain models that have converged, but explanation algorithms can also be applied to models that have not fully converged. Explanations for models that are “incomplete” may still be valuable; for example, we can use them to learn about which correlations were more difficult for models. However, this is beyond the scope of this paper.

On average, the training time for each model takes 25 minutes. The train/test split for the data used to train and evaluate all models are set to 7:3. The list of parameters we set manaully for the XGBoost classifiers are listed in Table 4.2. The fitting of the models are performed with the default settings of XGBoost.

Table 4.3: Performance scores for models trained for different `lead` times. We capture `precision`, `recall`, `f1`, `fbeta`, `AUROC`, and `average precision`.

lead (days)	precision	recall	f1	fbeta	AUROC	average precision
0	0.989	0.961	0.975	0.983	0.977	0.966
5	0.978	0.961	0.970	0.975	0.974	0.955
10	0.955	0.979	0.967	0.960	0.975	0.943
15	0.978	0.940	0.959	0.970	0.963	0.942
20	0.964	0.933	0.948	0.957	0.955	0.925
25	0.949	0.982	0.965	0.955	0.974	0.939
30	0.943	0.989	0.966	0.952	0.976	0.937

4.2.5 Model Performance Summaries

In Table 4.3, we present the performance metrics for the models trained for different lead times for the brake pad prediction problem. The high `f1` indicates that these models can be deployed in decision-making.

The plots of the performance metrics of the prediction models are listed in Appendix A. In general, all measures of model accuracy listed in Table 4.3 are higher when the `lead` time is closer to 0 days, and they decrease as the `lead` time increase.

Chapter 5

Designing explanations for changes in predictions

In this chapter, we discuss the explanations we developed to help users understand the behavior of brake pad prediction models at different time points. Our algorithm is derivative of **local feature contribution** algorithms. Local feature contribution explanations are feature attribution methods that explain a model by calculating the contribution of each feature to the prediction for a specific instance.

An example of such an explanation is shown in Figure 5-1. In this example, the ML task being explained is a regression problem that predicts the median price of a block of houses based on various information about the location and demographics. For local feature contribution explanations, the feature and feature value are both integral to the explanation; it does not make sense to state that a feature is important to some prediction regardless of the value that the feature takes. In Figure 5-1, the "ocean proximity" feature has a high positive contribution to the predicted price for this data instance when it takes a value of "near bay." This feature's contribution will change, and may even become negative, if we change its value to another value, such as "inland".

On the other hand, two data instances with the exact same value for a feature may have very different feature contributions. This is because the features of an ML model are usually not independent. Taking the same housing data set as an example, "ocean proximity" being "near bay" contributes positively to the predicted price in this selected instance (block of houses). However, if another instance (block of houses) has the same value for "ocean proximity", but is located in a flood-prone area, then the contribution of "ocean proximity" for this specific instance (block of houses) might be much lower than it is for the block shown in Figure 5-1.

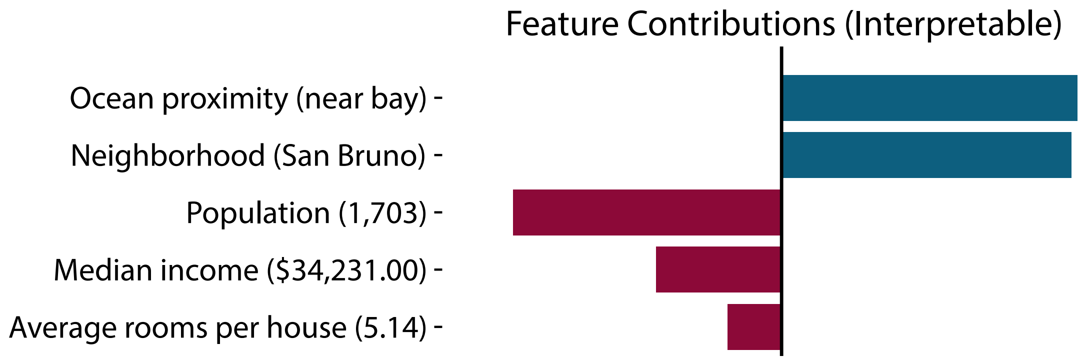


Figure 5-1: An example of an explanation generated by local feature contributions. The left side of the figure shows the features and feature values for an explained instance (with feature values in parentheses). The color blue indicates a positive contribution to the predicted median price, while the color red indicates a negative contribution. The length of the bar is equal to the absolute value of the contribution for the corresponding feature and feature value. Figure taken from Pyreal [24].

We select "local feature contribution" as the basis for our algorithm because of its popularity and intuitiveness. There are many implementations for local feature contribution algorithms. The most popular example is SHAP [14], which we used to calculate all the feature contributions in this work.

5.1 Explaining Differences in Predictions

Traditionally, when explaining ML models, the data instances, X, that need to be explained are generally assumed to be for separate entities. For the house pricing model, each instance is a single block of houses.

When explaining temporal ML models, data instances can be from separate entities, but they can also be from the same entity but with feature values taken at different time intervals. This possibility of extracting multiple data instances can be illustrated by showing how multiple training examples are extracted from the same entity. In our turbine brake pad case study, this is evident in how the training data is generated in the label-segment-featurize workflow [11]. Figure 5-2 is used in [11] to illustrate the training data instances from the original time-series data set. The first row, "Time series of events" in Figure 5-2, shows all the data points in a given data entity. The last row, "Select non-overlapping examples," shows that two data instances are generated from this single entity. From the perspective of the ML model, these two data instances are standalone instances, and the model does not take the fact they are from the same entity into account when making predictions for them. However, this relationship is interesting for human developers and users, as the variations between the feature values and model predictions in these two instances tell us about how the data changes as time progresses for this specific entity. In practice, the sequence of events can be very long, and a single entity can be used to generate more than two instances. The differences between such instances can tell us about the evolution of the problem, while the same cannot be said for normal data sets where data instances do not have special relationships.

We have learned from the success of counterfactual explanations that humans favor explanations that characterize alternative situations [16, 21]. It is often easier for humans to understand a concept through comparison with another concepts. Thus, it is reasonable to think that the notion of **change** would be beneficial for understanding a model's behavior; specifically, picking two or more instances and observing how they differ. In this work, we consider possible three types of difference: difference in feature values, difference in model predictions, and difference in feature contributions.

1. **Changes in feature values**. Changes in feature values are straightforward. They may contain the most information, but are often less useful for explaining the model's prediction alone. It is easy to point out features that changed a lot in value, but model predictions may not reflect such changes. Nevertheless, highlighting changes in feature value can be helpful to bridges and users, since they have knowledge about the subject matter and can draw correlations between feature values and model predictions without any information about the

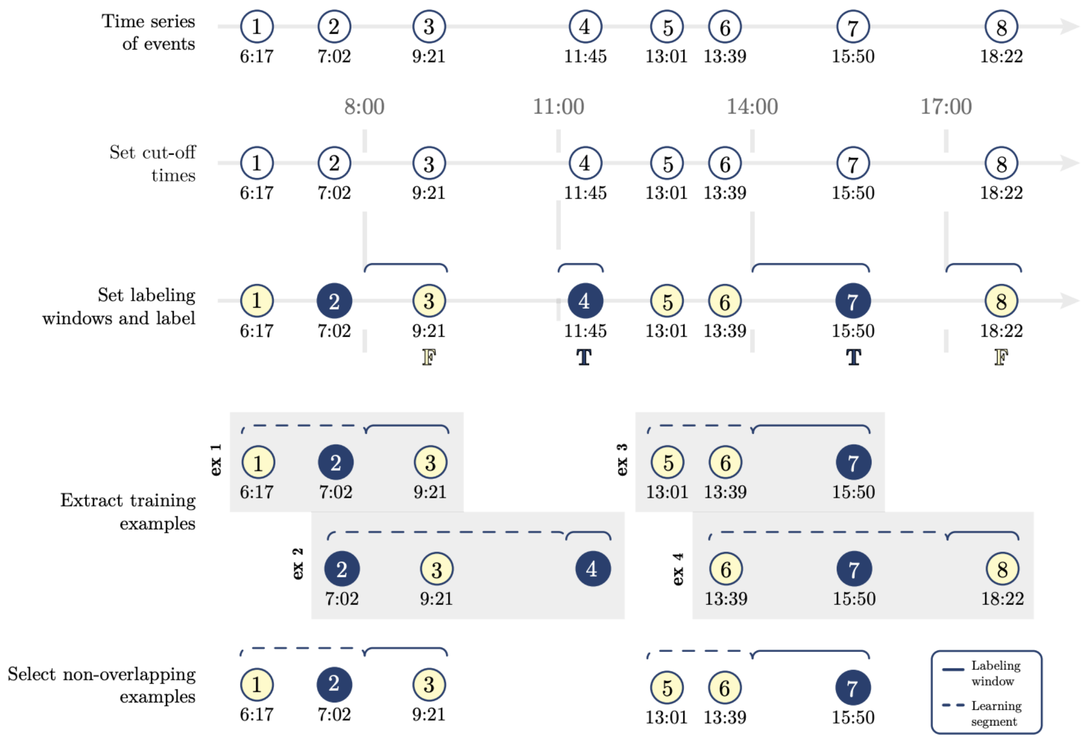


Figure 5-2: Figure illustrating the generation of data from event-based time-series data. Figure taken from Fig. 3. of Label, Segment, Featurize [11].

model.

2. **Changes in model predictions**. Changes in model predictions can be measured by comparing the outputs of two or more input data instances. Model predictions are mostly used to help users make decisions, so when the predictions change, the decisions may change as well. Some ML systems may use the predictions from a collection of models – Context B in Chapter 4 is one such use case. The models are given the same data, so the changes in feature values do not apply here. The changes in model predictions will be directly related to the changes of the prediction task.

3. **Changes in feature contributions**. As mentioned in Chapter 2, the feature contributions generated by a local explanation algorithm are specific to a data instance and its full coalition of feature values. Thus, some features may be relevant for a specific time point, but not for other time points. It can be interesting to see how the contributions of these feature evolve as time progresses. If the contribution changes a lot for the feature, but the feature values and model predictions do not change or change very slightly, this may signal that this case is worth investigating.

   In addition, since feature contributions tell us about the importance of each feature, one use case for feature contributions is to select the most strongly contributing features to build more lightweight models. This is extremely helpful in model deployment situations where resource constraints make it impossible to use all the features developers used in training. However, local feature contributions alone cannot ensure that a given set of features are sufficient to train a model that can perform well in deployment, since the feature contributions are specific to an instance. Now, we can collect the set of features that contribute the most to the predictions at different time steps for an `entity`, and use this collection to represent the features that are important for the model's prediction for this `entity`.

Chapter 6

System Design

This chapter describes the explanation system in detail and walks through how a developer (or bridges) can use our system to generate explanations for their brake pad failure prediction model.

6.1 Application Setup

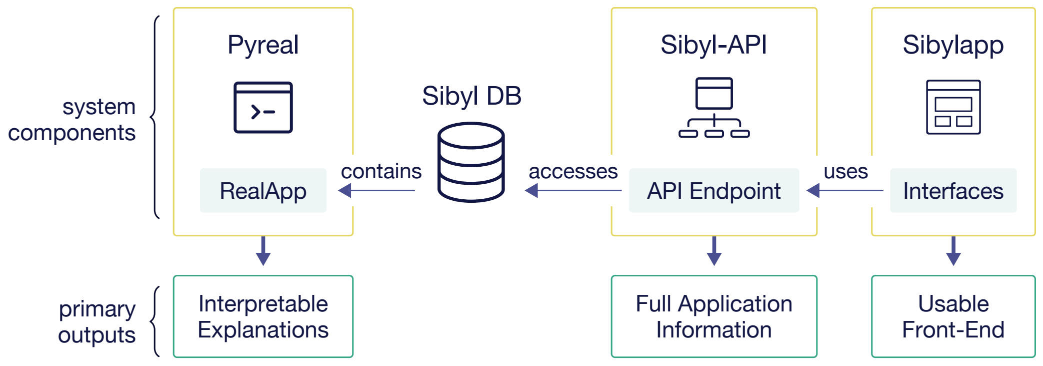


Figure 6-1: Diagram of our explanation system. The back end library Pyreal handles the calculation of explanations; e.g. feature contributions. Sibyl-API provides functions that accesses RealApp objects that contains information of data, models, and explanations. Sibylapp is the interface where the user interacts with the explanations.

Our explanation application is built on top of the system illustrated in Figure 6-1. This system contains three main components. The back end library **Pyreal** [24], the API handler **Sibyl-API**, and the front end interface **Sibylapp**.

6.1.1 Explanation Back End - Pyreal

We perform calculations of the explanation algorithm with Pyreal [24], which is a Python library that handles and generates useful ML explanations. Pyreal's primary component is the RealApp object, which fully encapsulates explanation applications that contain all necessary objects and information to make model predictions and generate ML explanations. This includes one or multiple ML models depending on the use case, explanation objects (explainers) that run explanation algorithms, and other functionalities that help make these explanations interpretable for humans[1].

Pyreal is designed with a low-code interface, so only minimal Python experience is required to set up a new RealApp object. This is the only step in the Sibyl configuration process that requires ML developer input, to the degree that setting up a new ML application requires developer input. Figure 6-2 shows the code snippet to create a RealApp object and use it to generate ML explanations.

In the system's workflow, the RealApp objects are created before the user interacts with the explanations. Throughout the applications relevant to this work, we create RealApp objects with `local feature contribution` explainers and store them in a MongoDB database. The model predictions and feature contributions displayed on the user interface are all pre-computed and accessed through Sibyl-API.

6.1.2 Data Communication - Sibyl-API

Sibyl-API is a framework that enables retrieval of information in the RealApp object through the implementation of REST-API. Table 6.1 lists the functions that we used for the work in this thesis.

The API functions we use fall into the following categories:

1. **Entity**. Functions in this category provide information about entities and the data associated with them.

2. **Feature**. Functions in this category provide information about input features, such as their readable descriptions, types, and categorizations.

[1]This paragraph of Pyreal description is taken from [24].

```python
import pyreal.applications.titanic as titanic
from pyreal import realapp
from pyreal.transformers import ColumnDropTransformer,
                                   MultiTypeImputer

# Load in data
x_orig, y = titanic.load_titanic_data()

# Load in feature descriptions -> dict(feature_name:
                                   feature_description, ...)
feature_descriptions = titanic.load_feature_descriptions()

# Load in model
model = titanic.load_titanic_model()

# Load in list of transformers
transformers = titanic.load_titanic_transformers()

# Create a RealApp object
realApp = RealApp(model, x_orig, y_orig=y,
                  transformers=transformers,
                  feature_descriptions=feature_descriptions,
                  )

# Generate and show explanation
explanation = real_app.produce_local_feature_contributions(
                                   sample_data)
swarm_plot(explanation, type="strip")
```

Figure 6-2: Code snippet for building a RealApp project and generating ML explanations. Here we show the demo with using the Titanic dataset [3].

Table 6.1: API functions available in Sibyl-API.

Category	URL	Description
entity	/api/v1/entities/	Get all Entities
entity	/api/v1/entities/{eid}/	Get an Entity by ID
entity	/api/v1/events/	Get the Events of an Entity
feature	/api/v1/categories/	Get all Categories
feature	/api/v1/features/	Get all Features
feature	/api/v1/features/{feature_name}/	Get a Feature by name
context	/api/v1/context/{context_id}/	Get a Context by ID
context	/api/v1/context/{context_id}/	Get a Context by ID
context	/api/v1/contexts/	Get all Context ids
computing	/api/v1/contributions/	Get feature contributions
computing	/api/v1/modified_contribution/	Get the feature contribution of an entity modified by changes
computing	/api/v1/modified_prediction/	Get the resulting model prediction after making all changes
computing	/api/v1/multi_contributions/	Get feature contributions for multiple eids, or for multiple row_ids in a single entity
model	/api/v1/models/	Get all Models
model	/api/v1/models/{model_id}/	Get a Model by ID
model	/api/v1/multi_prediction/	Get multiple predictions. If given multiple eids, return one prediction per eid

3. **Model**. Functions in this category provide access to all models stored in the database. A popular usage is to get models' predictions on data instances.

4. **Context**. Functions in this category provide context-specific information for setting up the usable ML interface. All terminology used can be given a domain-specific alternative — for example, entities could be "houses", "turbines", "customers", etc., and features could instead be referred to a "factors" or "properties". Additionally, these functions provide information such as how to format model outputs.

5. **Computing**. Functions in this category provide explanations and other computed information about the ML model or its predictions. The backend code for these endpoints usually use the saved RealApp object.

6.1.3 Dataset Preparation

Sibyl-API accesses a prepared MongoDB database. The database stores all information about entities, features, models, and context-specific configurations required to make usable explanation applications. Preparing such a database requires input from **bridges** to configure the application in an understandable way for **users**. This includes configuring feature descriptions and interpreting ML model outputs. Figure 6-3 shows the configuration to format Sibylapp. The Sibylapp setup for the application used in this work requires csv files for entities and feature information, and a list of pickled RealApp objects that contain all used models and their explanations. The details about the models are described in Chapter 4.

6.1.4 Application Front End - Sibylapp

Sibylapp is a front end interface developed in Streamlit. It is the interface **bridges** and **users** use for ML deployment and explanations. No coding is necessary to navigate through this application. The details of using Sibylapp to generate explanations for event-based time-series models are presented in the next section.

```
# Context configurations
output_preset: "failure"
# Whether to include entity rows in UIs (if False, only show first
                                  row).
use_rows: True
# Label to use for rows
row_label: "Timestamp"
# Whether to allow users to see prediction probabilities (only set
                                  to true if models have .
                                  predict_proba() functions)
show_probs: True

# Context-specific overrides for common terminology
terms:
  entity: "Turbine"
  prediction: "Predicted Outcome"
  increasing: "Risk" # As in, 'increasing features', or features
                                    that increase model prediction
  decreasing: "Protective" # As in, 'decreasing features', or
                                    features that decrease model
                                    prediction
```

Figure 6-3: Configuration of Sibyl-API for the brake pad failure prediction case study.

6.2 Components of Sibylapp

In this section, we will describe each function of the application in detail. This application starts by selecting an explanation to use as shown in Figure 6-4. There are currently six pages, corresponding to six usage scenarios. The pages relevant to this work are **Compare Cases**, **Change over Time**, and **Same Target Time**.

6.2.1 Application 1: Compare Predictions at Different Time Points

The first application is implemented under the **Compare Cases** page. This application corresponds to Context A in Chapter 4 and focuses on understanding how a model's prediction for a specific `entity` changes at different time points in future, fixed `lead`, and different t_c. As displayed in Figure 6-5, the application first prompts the user to selects a model and an `entity` (turbine) and two different instances (indicated by its cut-off time t_c). It then displays the selected model predictions for the two instances. For classification models like our brake pad failure case study,

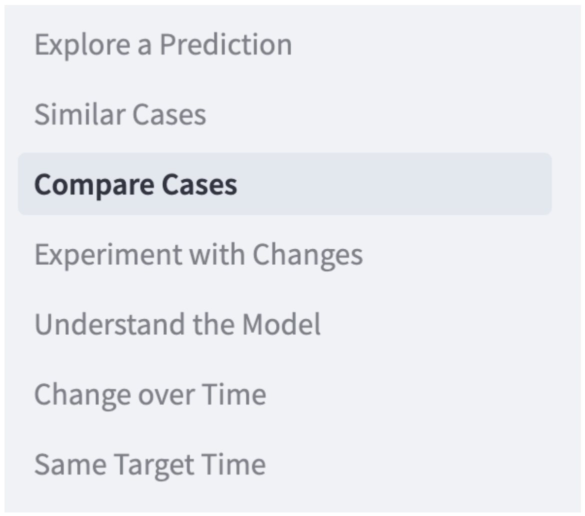
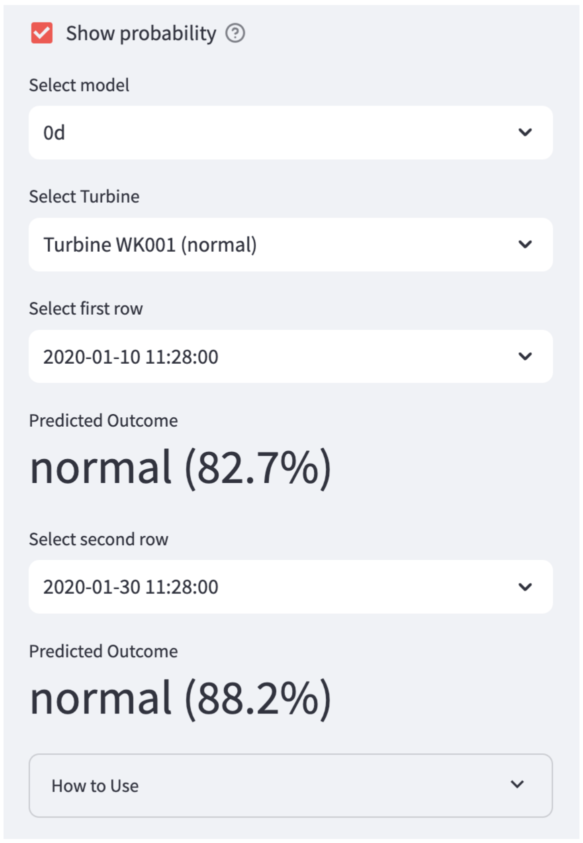


Figure 6-4: Explanation application selection box. Each option refers to a different type of explanation and scenario.

the binary outcome alone is not particularly informative for comparison between two instances, so we provided an option to display the prediction in terms of probability of the outcome.

Figure 6-6 shows the first part of the display of the **Compare Cases** application. It first shows the difference in model predictions for the two selected time points. This is calculated by subtracting the first model prediction from the second model prediction. There are some features that can help users to navigate the explanations. The "Search and filter" dropdown is available for every application; the user can filter features shown in the contributions table (Figure 6-7) to narrow their focus to certain features. The filtering supports filter by feature name and feature category.

The next functionality is the sorting of the features. The sorting was based on the change in feature contributions so that we can list the features that have the most different contributions on the top of the list. We implemented three sorting strategies. The first sort by the largest absolute feature contribution difference, which we see as the features that have their impact change the most between the two predictions. The second lists the features that experience the most positive change in feature contribution and the third lists those experience the most negative change. These are useful to identify which features are experiencing the most change from the perspective



☑ Show probability

Select model

0d

Select Turbine

Turbine WK001 (normal)

Select first row

2020-01-10 11:28:00

Predicted Outcome

normal (82.7%)

Select second row

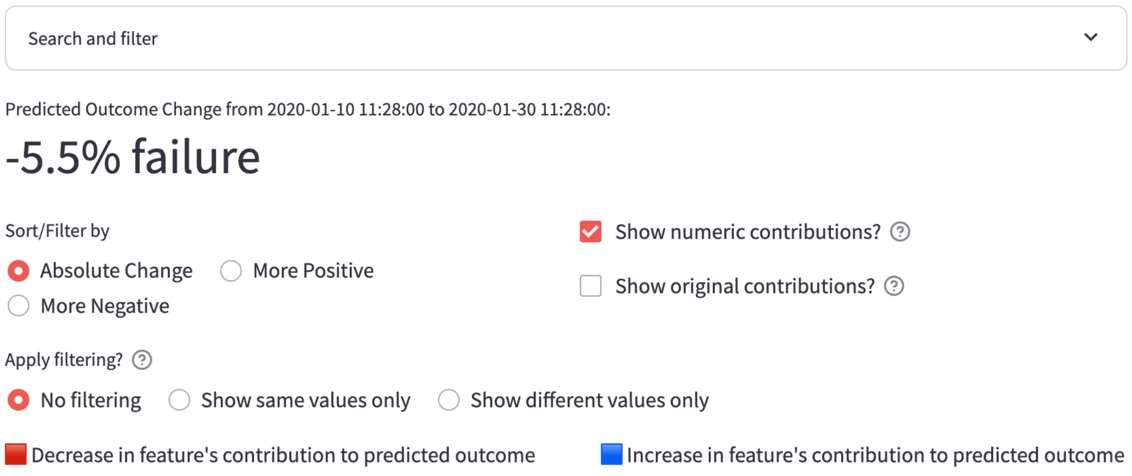
2020-01-30 11:28:00

Predicted Outcome

normal (88.2%)

How to Use

Figure 6-5: Selection of data and model for the **Compare Cases** application. The shown example selects model 0d (lead=0 days), Turbine WK001 as the explained entity, and two cut-off times at 2020-01-10 11:28:00 and 2020-01-30 11:28:00. The predictions for these two instances are 82.7% of normal and 88.2% of normal respectively.



Search and filter

Predicted Outcome Change from 2020-01-10 11:28:00 to 2020-01-30 11:28:00:

-5.5% failure

Sort/Filter by: [✓ Absolute Change] More Positive More Negative

☑ Show numeric contributions?

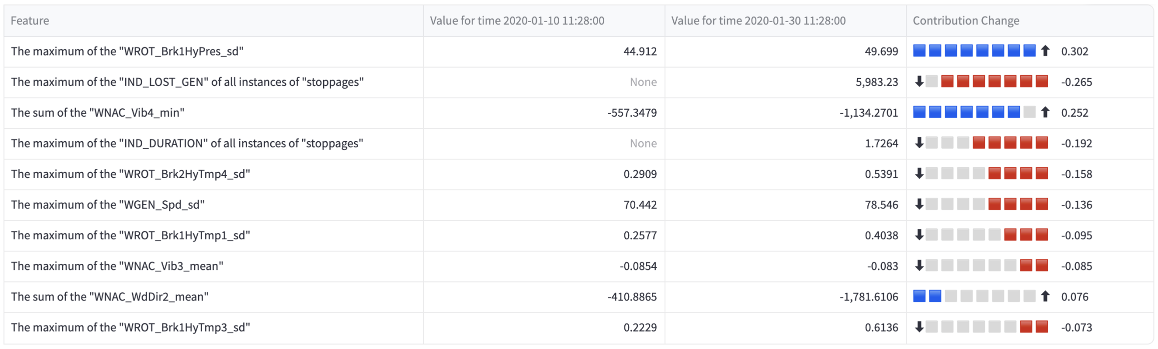
☐ Show original contributions?

Apply filtering?: [✓ No filtering] Show same values only Show different values only

🟥 Decrease in feature's contribution to predicted outcome

🟦 Increase in feature's contribution to predicted outcome

Figure 6-6: Difference in model predictions. The model predicts that the probability that the brake pad will experience failure at 2020-01-30 11:28:00 is 5.5% lower than that at 2020-01-10 11:28:00.



Feature	Value for time 2020-01-10 11:28:00	Value for time 2020-01-30 11:28:00	Contribution Change
The maximum of the "WROT_Brk1HyPres_sd"	44.912	49.699	🟦🟦🟦🟦🟦🟦🟦🟦 ↑ 0.302
The maximum of the "IND_LOST_GEN" of all instances of "stoppages"	None	5,983.23	↓ ⬜🟥🟥🟥🟥🟥🟥🟥 -0.265
The sum of the "WNAC_Vib4_min"	-557.3479	-1,134.2701	🟦🟦🟦🟦🟦🟦🟦⬜ ↑ 0.252
The maximum of the "IND_DURATION" of all instances of "stoppages"	None	1.7264	↓ ⬜⬜⬜🟥🟥🟥🟥🟥 -0.192
The maximum of the "WROT_Brk2HyTmp4_sd"	0.2909	0.5391	↓ ⬜⬜⬜⬜🟥🟥🟥🟥 -0.158
The maximum of the "WGEN_Spd_sd"	70.442	78.546	↓ ⬜⬜⬜⬜🟥🟥🟥🟥 -0.136
The maximum of the "WROT_Brk1HyTmp1_sd"	0.2577	0.4038	↓ ⬜⬜⬜⬜⬜🟥🟥🟥 -0.095
The maximum of the "WNAC_Vib3_mean"	-0.0854	-0.083	↓ ⬜⬜⬜⬜⬜⬜🟥🟥 -0.085
The sum of the "WNAC_WdDir2_mean"	-410.8865	-1,781.6106	🟦🟦⬜⬜⬜⬜⬜⬜ ↑ 0.076
The maximum of the "WROT_Brk1HyTmp3_sd"	0.2229	0.6136	↓ ⬜⬜⬜⬜⬜⬜🟥🟥 -0.073

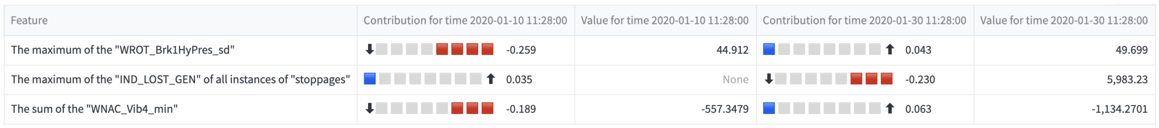
Figure 6-7: List of features sorted by absolute difference in feature contributions. The feature values of the two instances are shown side-by-side for clear comparison if the contribution variation coincides with feature value variation. The first column shows the feature names. The second column shows the feature values for the first selected data instance. The third column shows the feature values for the second selected data instance. The last column shows how much the feature contribution has changed.

of local feature contributions, which often is correlated with the changes in predictions between the two data instances.

For display purposes, Figure 6-7 only shows the contribution change for each feature and not the feature contribution values for the two instances. Checking the "Show original contributions" option in Figure 6-6 will reveal the original feature contributions for both instances for each feature. The three features with the largest change in contribution all had their contributions inverse in sign as shown in Figure 6-8. The contribution of `The maximum of the "WROT_Brk1HyPres_sd"` goes from -0.259 to 0.043; the contribution of `The maximum of the "IND_LOST_GEN" of all instances of "stoppages"` goes from 0.035 to -0.230; the contribution of `The sum of the "WNAC_Vib4_min"` goes from -0.189 to 0.063.

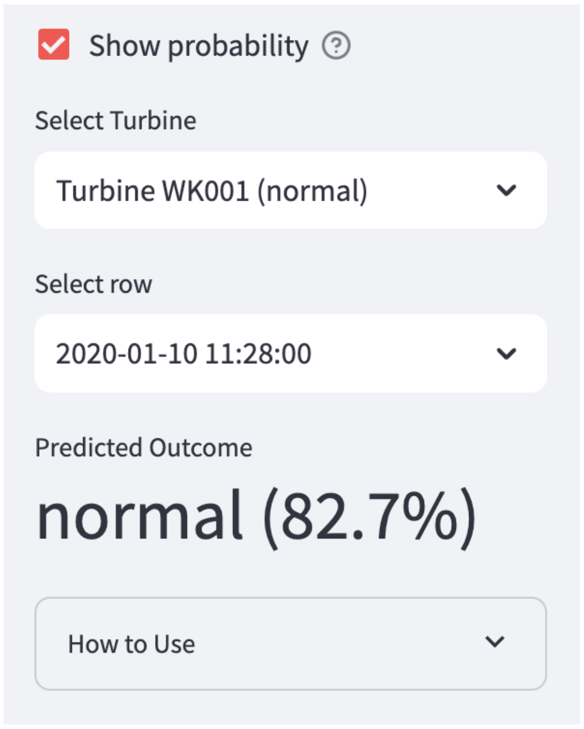
6.2.2 Application 2: Compare Predictions for Different Time into the Future

The second application is implemented under the **Change over Time** page. This application corresponds to Context B in Chapter 4, which shows the predictions of different `lead` with the same t_c. As shown in Figure 6-9, the user only selects



Feature	Contribution for time 2020-01-10 11:28:00	Value for time 2020-01-10 11:28:00	Contribution for time 2020-01-30 11:28:00	Value for time 2020-01-30 11:28:00
The maximum of the "WROT_Brk1HyPres_sd"	⬇ ⬜⬜⬜⬜🟥🟥🟥🟥 -0.259	44.912	🟦⬜⬜⬜⬜⬜⬜⬜ ⬆ 0.043	49.699
The maximum of the "IND_LOST_GEN" of all instances of "stoppages"	🟦⬜⬜⬜⬜⬜⬜⬜ ⬆ 0.035	None	⬇ ⬜⬜⬜⬜⬜🟥🟥🟥 -0.230	5,983.23
The sum of the "WNAC_Vib4_min"	⬇ ⬜⬜⬜⬜⬜🟥🟥🟥 -0.189	-557.3479	🟦⬜⬜⬜⬜⬜⬜⬜ ⬆ 0.063	-1,134.2701

Figure 6-8: The feature contribution values for the three features that have the largest difference. All three features have their contribution changed signs. The first column shows the feature names. The second column shows the feature contribution for the first selected data instance. The third column shows the feature value for the first selected data instance. The fourth column shows the feature contribution for the second selected data instance. The fifth column shows the feature value for the second selected data instance.



☑ Show probability

Select Turbine

Turbine WK001 (normal)

Select row

2020-01-10 11:28:00

Predicted Outcome

normal (82.7%)

How to Use

Figure 6-9: Selection of data for the **Change over Time** application. The shown example selects Turbine WK001 as the explained entity, and a single cut-off time at 2020-01-10 11:28:00.

lead (days)	0	5	10	15	20	25	30
Failure probability	0.173	0.116	**0.439**	0.160	0.306	0.342	0.264

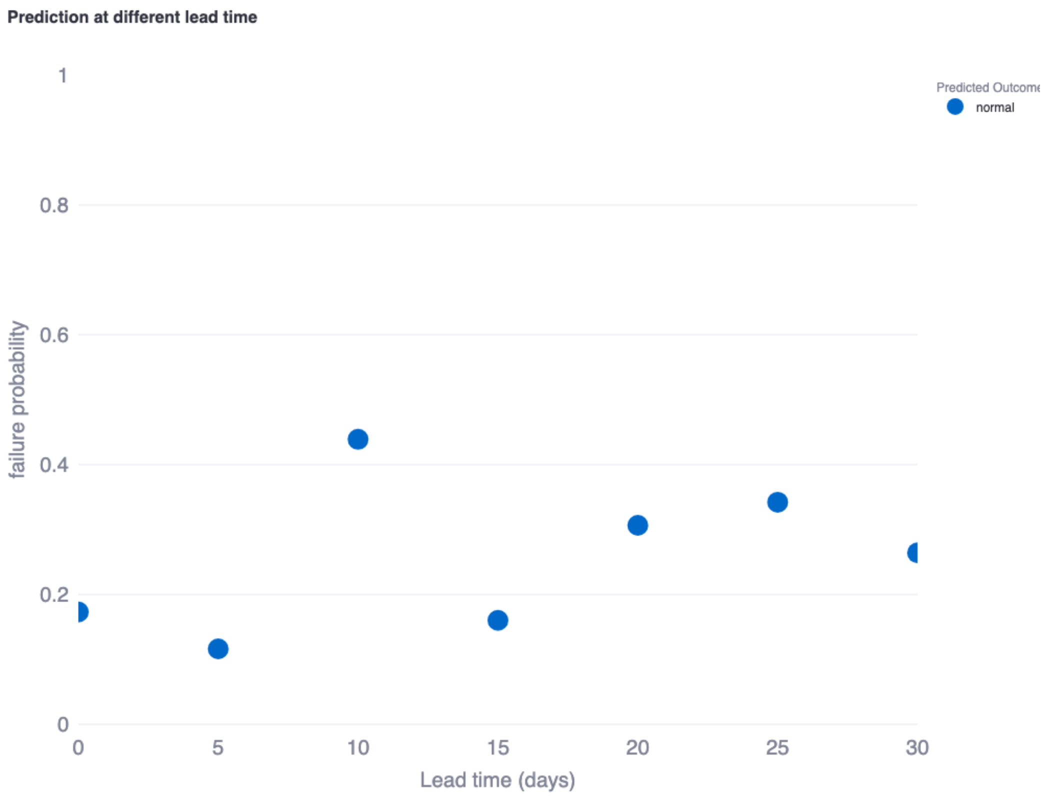
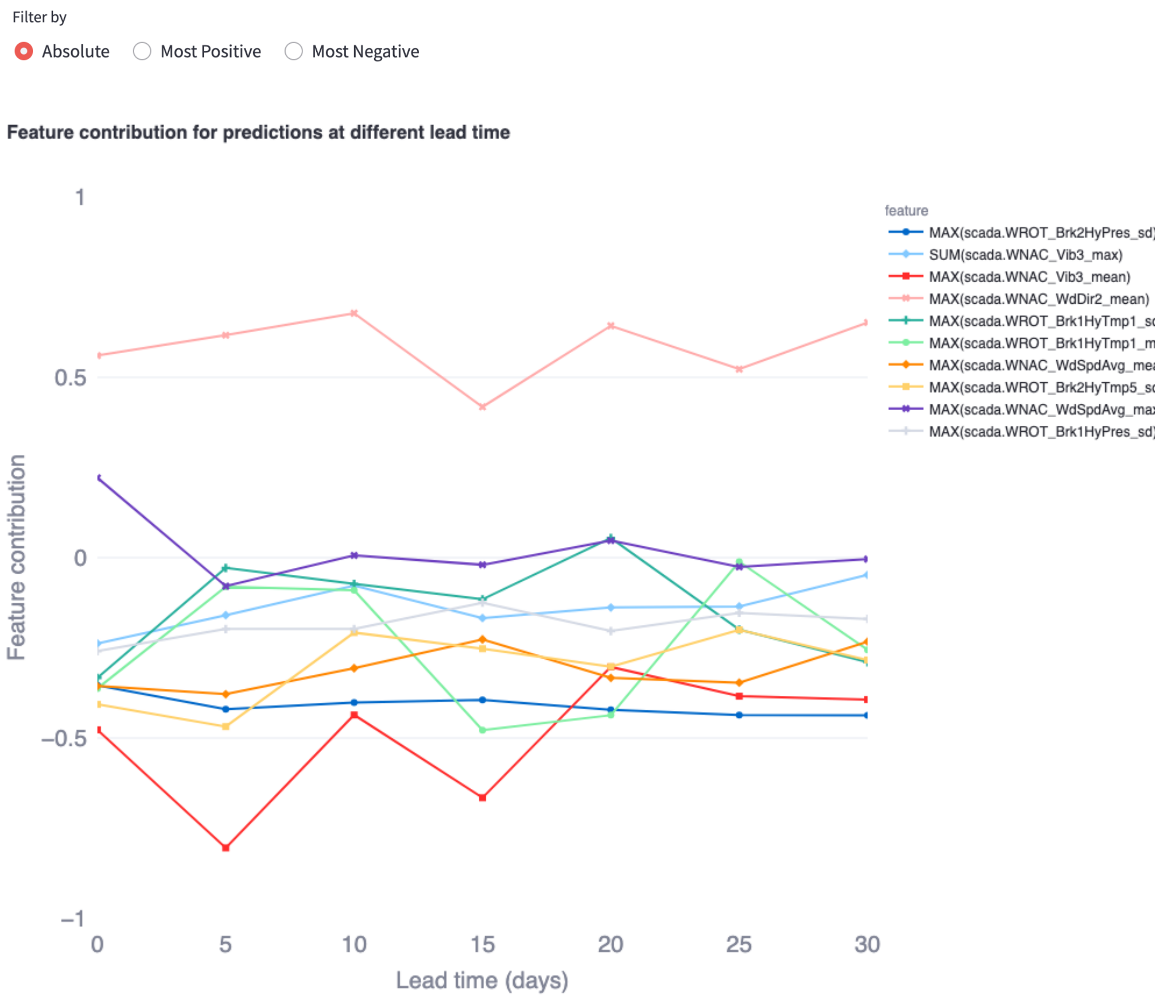


Figure 6-10: Scatter plot of predictions (in terms of failure probability) over different lead time for entity Turbine WK001 and cut-off time 2020-01-10 11:28:00.

`entity` and t_c for this application. All models loaded into Sibylapp are used in the application so the user do not have to select a specific model. We use 7 models with `lead` ranging from 0 days to 30 days with 5-day increments. All model predictions and corresponding feature contributions are shown in scatter plots and line charts so that the user can clearly observe how these values progress over time. Figure 6-10 shows the predictions over different `lead` times for the selected data instance. This scatter plot informs the trend of model predictions over different `lead` times.

Figure 6-11 shows the feature contributions over different `lead` times. We currently sort features based on their feature contributions for `lead` time $=$ 0 days. Similar to the Compare Cases page, the sorting options include sort by the absolute value, most positive, and most negative feature contributions. We showed ten fea-


Filter by

[✓ Absolute] Most Positive Most Negative

Figure 6-11: Line plot of feature contributions over predictions of different lead time. The features listed in this plot are the 10 features with the largest absolute value of feature contribution for the model with lead = 0 days, for entity Turbine WK001 and cut-off time 2020-01-10 11:28:00.

tures for Figure 6-11 for clarity but this number can be adjusted. The features in the legend can be toggled to show/hide specific feature contributions. In this Figure, `MAX(scada.WNAC_WdDir2_mean)` remains the most positive contributing feature over all different `lead` times.

6.2.3 Application 3: Compare Predictions for a Specific Time

The third application is implemented under the **Same Target Time** page. This application corresponds to Context C in Chapter 4 and shows the variation of model predictions and feature contributions for models with different t_c but the same t_{target}. The user only selects `entity` and t_{target} for this application. As shown in Figure 6-12 and Figure 6-13, we used models with `lead` $=$ 0, 10, 20, 30 days (separation between each consecutive cut-off time is 10 days). Since the data instances for each `entity` are referenced through their cut-off times, we must find data instances for each model such that all four predictions are made for the same t_{target}. This is not a big problem if the data set is dense with respect to the `lead` times. However, when there are limited instances for the selected `entity`, it may be more suitable to first pick the cut-off times and then train models with `lead` such that cut-off time plus `lead` equals t_{target} for each t_c in the given times.

Figure 6-12 shows the predictions made at different cut-off times. Figure 6-13 shows the feature contributions of models at different cut-off times. Both of these two plots were designed in a similar fashion to Figure 6-10 of the Change over Time page.

lead time (days)	15	10	5	0
Failure probability	**0.147**	0.080	0.064	0.097

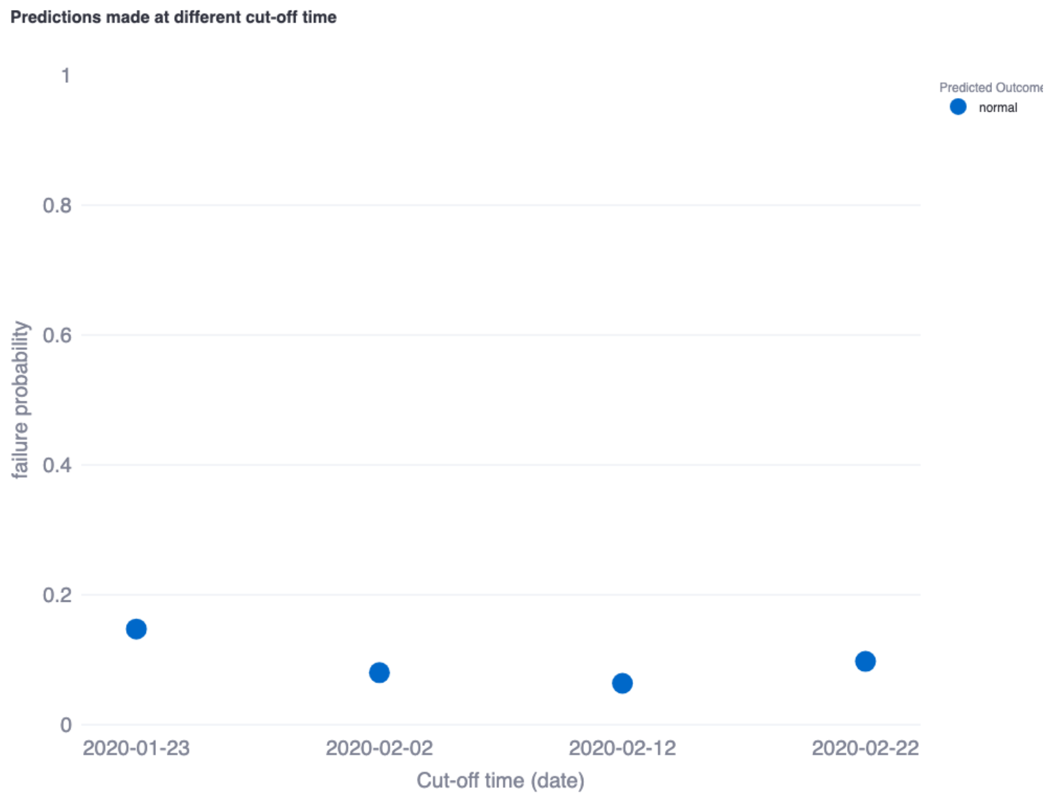
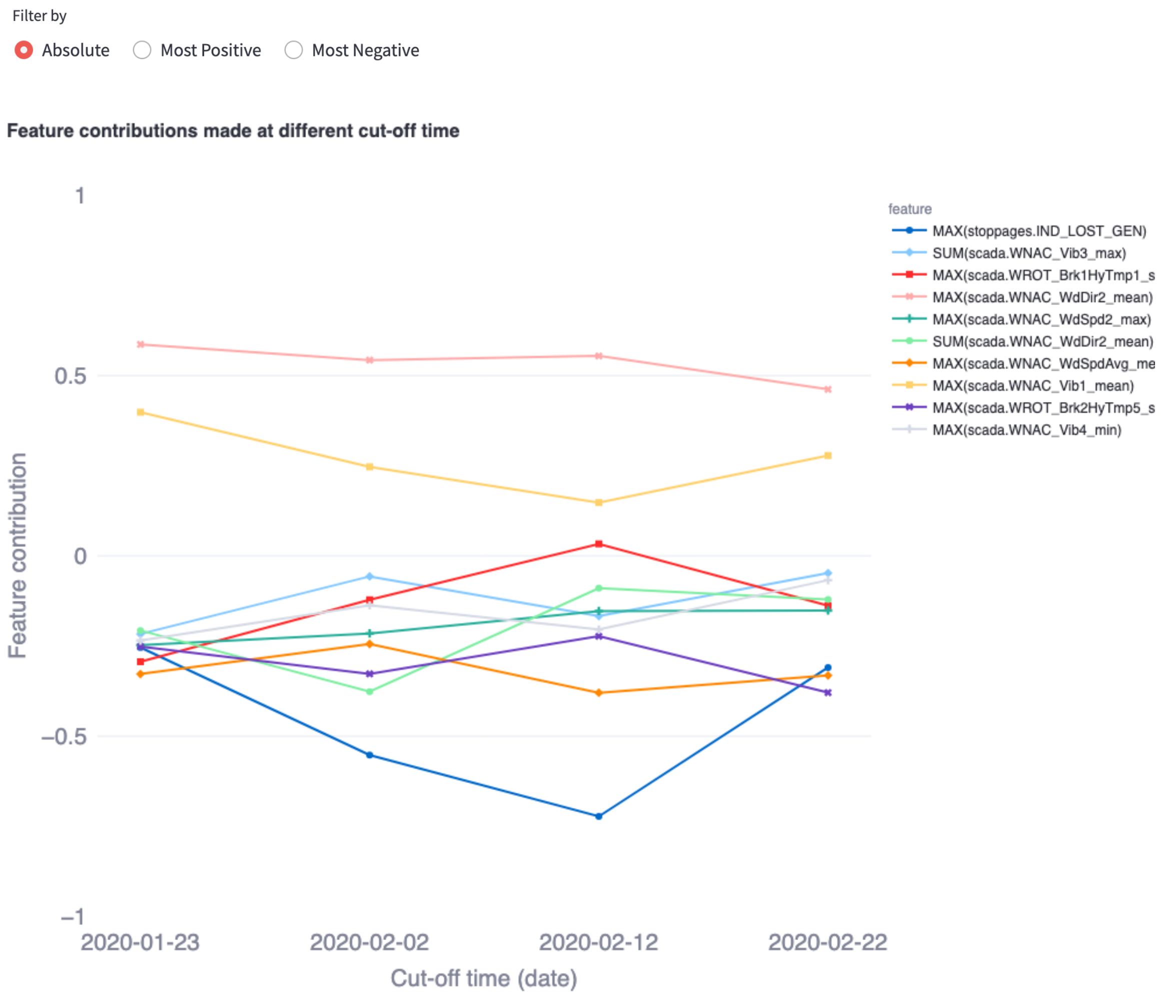


Figure 6-12: Scatter plot of predictions (in terms of failure probability) made at different cut-off time for entity Turbine WK002 and target time 2020-02-22 15:33:00. The time of the day for the cut-off time is 15:33:00 for every data point and is not displayed in the Figure.


Filter by

[✓ Absolute] Most Positive Most Negative

Figure 6-13: Line plot of feature contributions over predictions made at different cut-off times. The features listed in this plot are the 10 features with the largest absolute value of feature contribution for the model with the earliest cut-off time at 2020-01-23 15:33:00.

Chapter 7

Conclusion

7.1 Conclusion

In conclusion, this thesis has addressed the critical gap in ML explanation design, particularly focusing on real-world usable deployment for event-based time-series data. We developed two explanation scenarios and proposed explanations based on changes in feature values, model predictions, and feature contributions; we provided insights into understanding ML models applied to temporal prediction problems such as turbine brake pad failure predictions. Our solution, comprising a usable ML interface and context-aware explanations, serves as a framework for decision-makers to comprehend prediction mechanisms effectively.

Through the development and demonstration of our usable ML interface integrated with explanations tailored to deployment scenarios, we have showcased the practical applicability of contextual understanding in deploying ML models effectively through the Pyreal, Sibyl-API, Sibylapp system.

As ML continues to permeate various domains, ensuring transparency and interpretability in model predictions becomes increasingly imperative. Our work contributes to this endeavor by providing a concrete methodology and framework for developing context-aware ML explanations, ultimately enhancing trust, usability, and decision-making in real-world applications. Further research in this direction could explore additional deployment scenarios and refine explanation techniques to accommodate evolving complexities in ML systems.

7.2 Future Directions

The usable ML system we developed is designed as a general purpose application and not limited to wind turbine brake pad failure prediction. Most of the explanations and scenarios in this work are currently still intended for our specific use case, so we can improve the existing system by applying it to more use cases. In addition, more flexibility can be added to existing applications. For example, we can enable model selection in the **Change over Time** page so that users can choose which `lead` time to compare the model predictions and feature contributions.

7.2.1 Custom Explanation scenarios

One big part of this project is the formation of those explanation scenarios. There may be other common scenarios in time-series modeling that we did not cover in this work. The design of Sibylapp is generalizable to add new application scenarios and they will not interfere with other parts of the system.

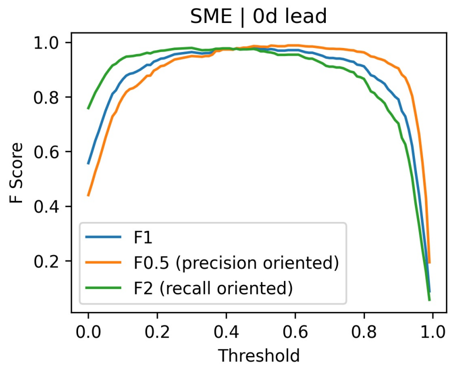
7.2.2 Evaluation of Explanations

The frameworks and explanations proposed in this work has not gone through a thorough case study with the ML users yet, which is critical for evaluating the usability of our system. We would also like to explore further use cases of contribution difference for model deployment.

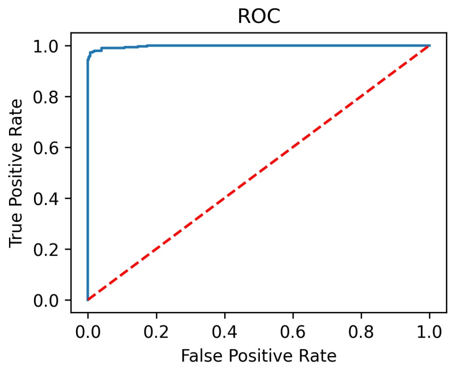
Appendix A

Supplemental plots

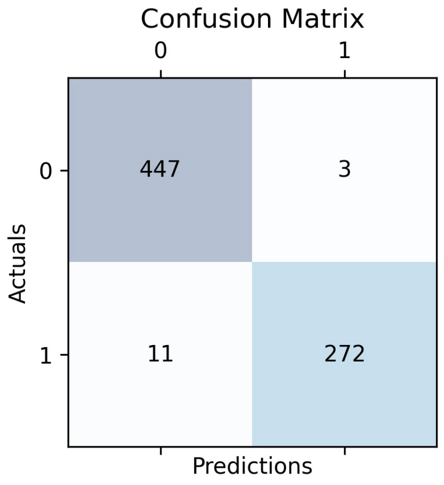
A.1 Prediction metric plots for models used in usable ML system



(a) F scores for model with lead = 0 days.

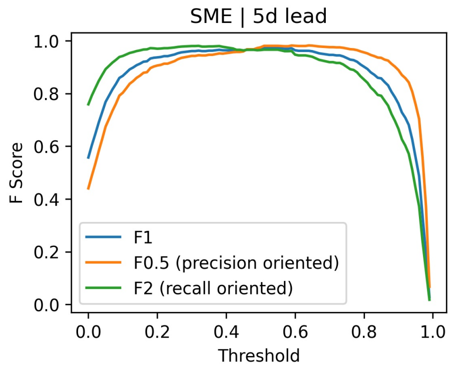


(b) ROC for model with lead = 0 days.

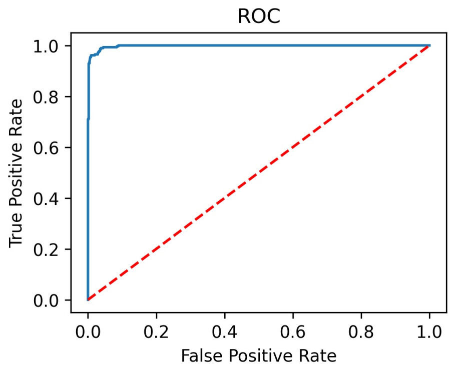


(c) Confusion matrix for model with lead = 0 days.

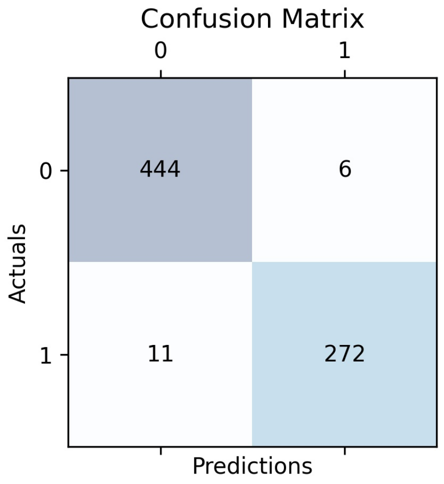
Figure A-1: Model performance metrics for model with lead = 0 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.



(a) F scores for model with lead = 5 days.



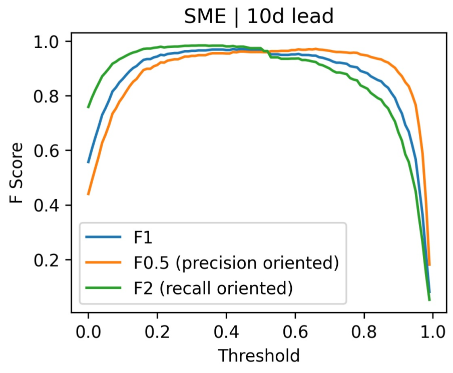
(b) ROC for model with lead = 5 days.


Confusion Matrix
Actuals
Predictions

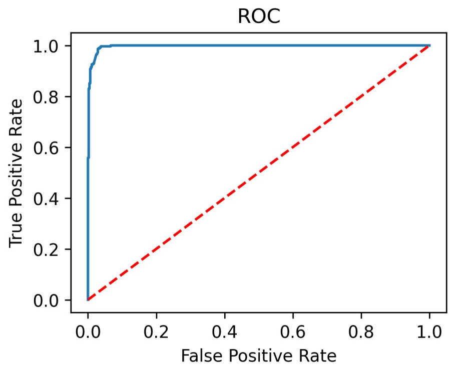
	0	1
0	444	6
1	11	272

(c) Confusion matrix for model with lead = 5 days.

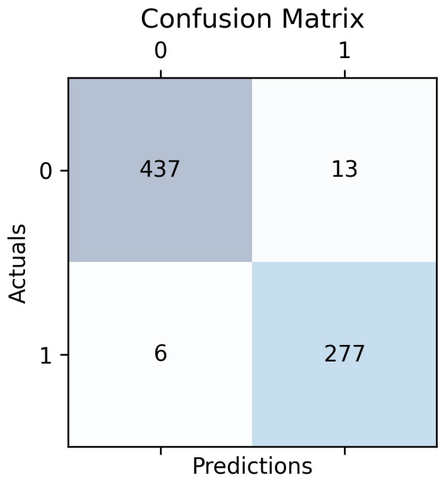
Figure A-2: Model performance metrics for model with lead = 5 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.



(a) F scores for model with lead = 10 days.

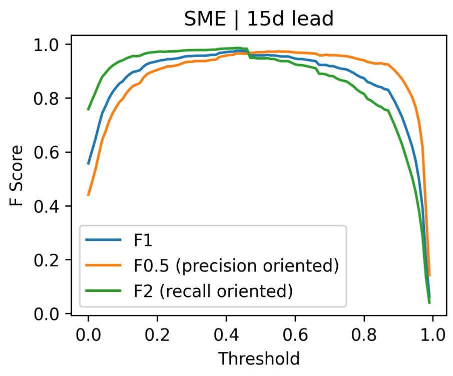


(b) ROC for model with lead = 10 days.

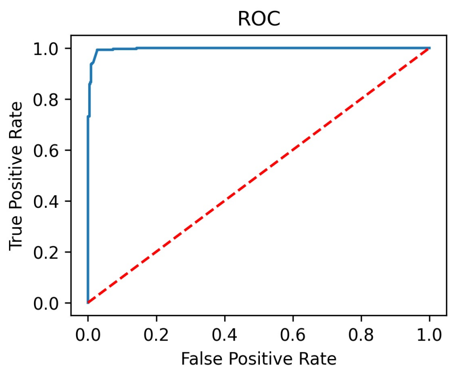


(c) Confusion matrix for model with lead = 10 days.

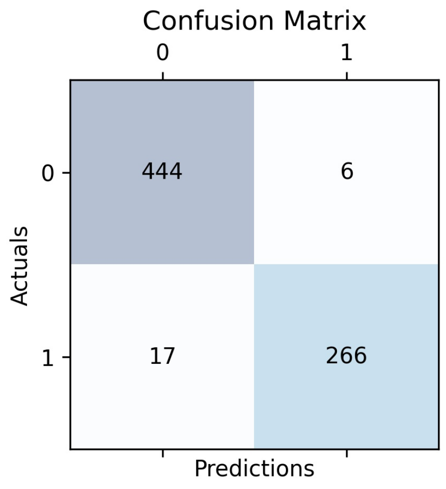
Figure A-3: Model performance metrics for model with lead = 10 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.



(a) F scores for model with lead = 15 days.

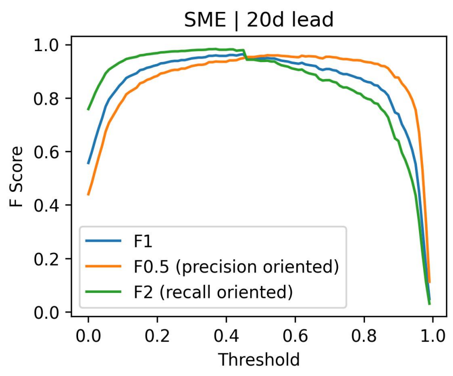


(b) ROC for model with lead = 15 days.

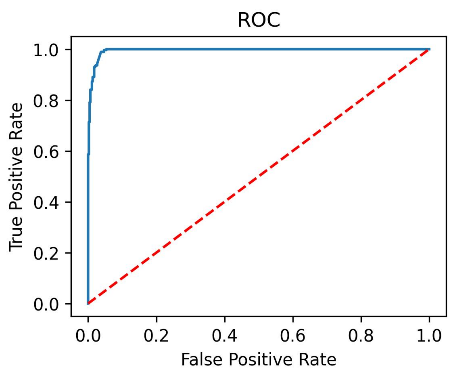


(c) Confusion matrix for model with lead = 15 days.

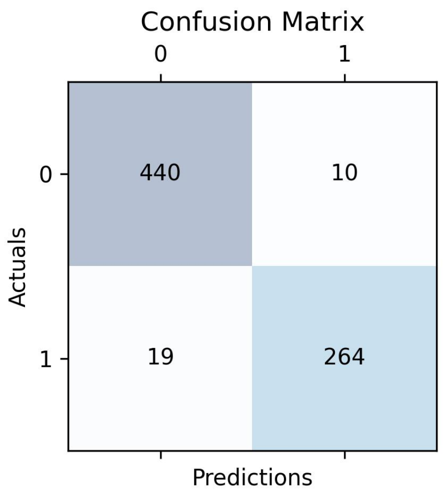
Figure A-4: Model performance metrics for model with lead = 15 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.



(a) F scores for model with lead = 20 days.

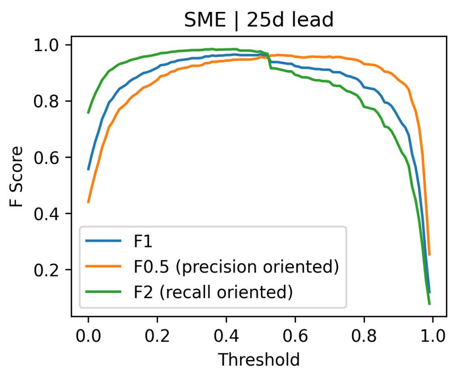


(b) ROC for model with lead = 20 days.

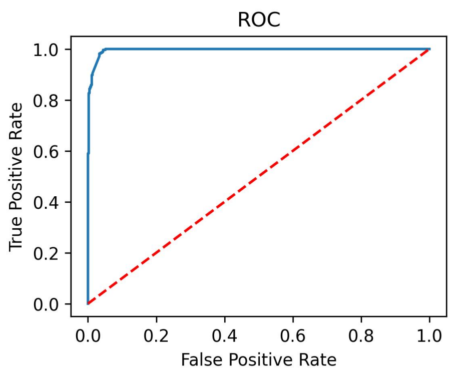


(c) Confusion matrix for model with lead = 20 days.

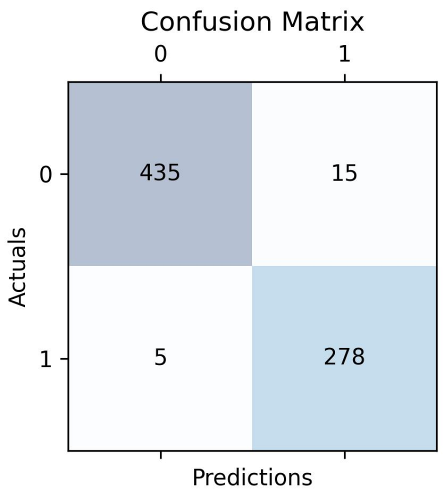
Figure A-5: Model performance metrics for model with lead = 20 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.



(a) F scores for model with lead = 25 days.



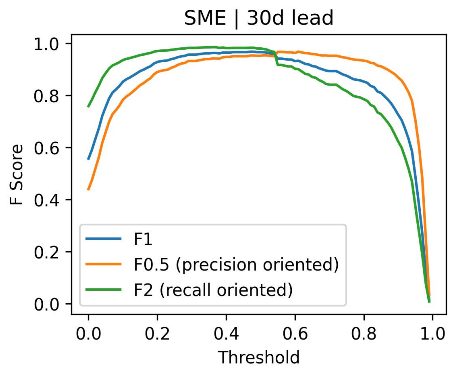
(b) ROC for model with lead = 25 days.



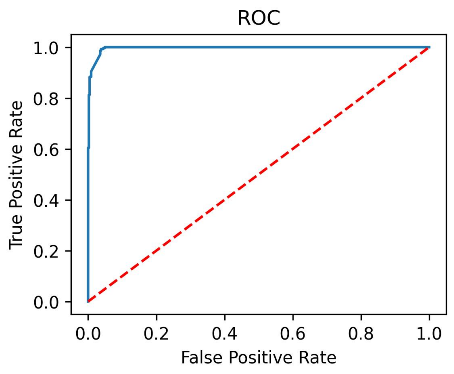
	0	1
0	435	15
1	5	278

(c) Confusion matrix for model with lead = 25 days.

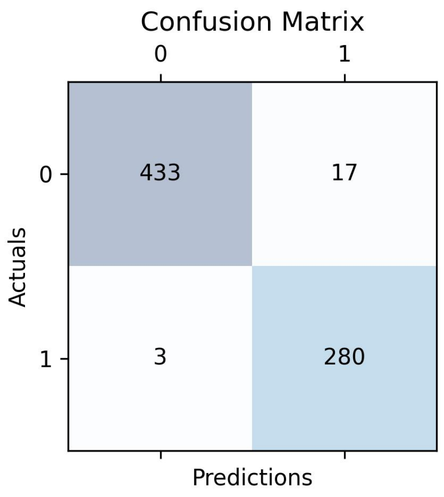
Figure A-6: Model performance metrics for model with lead = 25 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.



(a) F scores for model with lead = 30 days.



(b) ROC for model with lead = 30 days.



Confusion Matrix
Actuals
Predictions

	0	1
0	433	17
1	3	280

(c) Confusion matrix for model with lead = 30 days.

Figure A-7: Model performance metrics for model with lead = 30 days. SME stands for subject matter experts who enabled feature generation to give high predictive accuracy for the models.

Bibliography

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. Number of pages: 10 Place: San Francisco, California, USA tex.acmid: 2939785.

[2] Jonathan Crabbe and Mihaela van der Schaar. Explaining time series predictions with dynamic masks. In *International Conference on Machine Learning*, 2021.

[3] Will Cukierski. Titanic - machine learning from disaster, 2012.

[4] Hans de Bruijn, Martijn Warnier, and Marijn Janssen. The perils and pitfalls of explainable ai: Strategies for explaining algorithmic decision-making. *Government Information Quarterly*, 39(2):101666, 2022.

[5] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017.

[6] Riccardo Guidotti, Anna Monreale, Francesco Spinnato, Dino Pedreschi, and Fosca Giannotti. Explaining any time series classifier. *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pages 167–176, 2020.

[7] Maël Guillemé, Véronique Masson, Laurence Rozé, and Alexandre Termier. Agnostic local explanation for time series classification. *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 432–439, 2019.

[8] Frances R. Hartwell. Zephyr: a data-centric framework for predictive maintenance of wind turbines, 2023.

[9] Fabian Hinder, Valerie Vaquet, Johannes Brinkrolf, and Barbara Hammer. Model based explanations of concept drift, 2023.

[10] Helen Jiang and Erwen Senge. On two xai cultures: A case study of non-technical explanations in deployed ai system, 2021.

[11] James Max Kanter, Owen Gillespie, and Kalyan Veeramachaneni. Label, segment, featurize: A cross domain framework for prediction engineering. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 430–439, 2016.

[12] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015.

[13] Zachary C. Lipton. The mythos of model interpretability, 2017.

[14] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[15] Scott M. Lundberg, Bala G. Nair, Monica S. Vavilala, Mayumi Horibe, Michael J. Eisses, Trevor L. Adams, David Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry H. Kim, and Su-In Lee. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature biomedical engineering*, 2:749 – 760, 2018.

[16] Christoph Molnar. *Interpretable Machine Learning.* 2 edition, 2022.

[17] Maximilian Muschalik, Fabian Fumagalli, Barbara Hammer, and Eyke Hüllermeier. Agnostic explanation of model change based on feature importance. *KI - Künstliche Intelligenz*, 36(3-4):211–224, 2022.

[18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. 2016.

[19] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, 2019.

[20] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[21] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr, 2018.

[22] Alexandra Zytek, Dongyu Liu, Rhema Vaithianathan, and Kalyan Veeramachaneni. Sibyl: Understanding and addressing the usability challenges of machine learning in high-stakes decision making, 2021.

[23] Alexandra Zytek, Wei-En Wang, Sofia Koukoura, and Kalyan Veeramachaneni. Lessons from usable ml deployments and application to wind turbine monitoring, 2023.

[24] Alexandra Zytek, Wei-En Wang, Dongyu Liu, Laure Berti-Equille, and Kalyan Veeramachaneni. Pyreal: A framework for interpretable ml explanations, 2023.