A Computational Framework for Estimating Days of Maintenance Delay of Naval Ships

Gerald White
NJIT, US
gkw@njit.edu

Deep Mistry
NJIT, US
dm728@njit.edu

Kevin Chhoa
NJIT, US
kevin.chhoa@njit.edu

Senjuti Basu Roy
NJIT, US
senjutib@njit.edu

Lingyi Zhang
U.S. Naval Research Laboratory
lingyi.zhang.civ@us.navy.mil

Adam Bienkowski
UCONN, US
adam.bienkowski@uconn.edu

Krishna Pattipati
UCONN, US
krishna.pattipati@uconn.edu

ABSTRACT

This work proposes a computational framework for estimating Days of Maintenance Delay (DoMD) for US Navy ships, aiming to improve fleet maintenance planning. The data, containing Controlled Unclassified Information (CUI), is obfuscated and consists of both time-dependent and time-invariant attributes, forming a "fat" tensor with many attributes but few instances (around 200). The solution is a predictive maintenance pipeline that uses obfuscated data for training and then retrains on raw data in the Navy environment without human intervention. The pipeline includes modules for transforming raw data, identifying effective features, selecting machine learning models, and training models robust to outliers and noise. The framework addresses computational challenges and optimization opportunities, and its effectiveness is demonstrated experimentally within the Navy environment.

1 INTRODUCTION

In January 2023, Vice Admiral Roy Kitchener, then-Commander of the Naval Surface Forces, set a readiness goal for the US Navy to have 75 combat-ready surface ships available at all times. A major challenge to this goal is the frequent delays in the maintenance of these ships, which is planned 2-3 years in advance but often incurs costly execution delays. To address this, this work proposes a data science pipeline to accurately estimate Days of Maintenance Delay (DoMD) at any point during or before the execution process. The pipeline is being deployed as a back-end engine for a fleet-readiness application within the Navy's Ship Maintenance Data Improvement Initiative (SMDII).

DoMD Query. An end user logged into SMDII should be able to at any time query the estimated delay of any ongoing or future ship maintenance period, called availabilities or "avails".

Challenges. The Navy Maintenance Database (NMD) [9] contains only 200 maintenance instances for study, yet there exists a timestamped history of a myriad of structured attributes which could potentially cause delays in the avail. This makes the resulting dataset short and extremely wide, with tens of thousands of potential features for the 200 avails.

While the data is time-dependent and evolves throughout the execution of the maintenance period, delay is obviously only measured at the conclusion of the avail. Thus, DoMD modeling *is not a classical time series problem nor is it a standard time-independent regression problem and as the response variable is not measured at every time step,* similar to the field of game playing [29, 38]. Machine learning models on such datasets tend to overfit and fail to generalize well to new, unseen data.

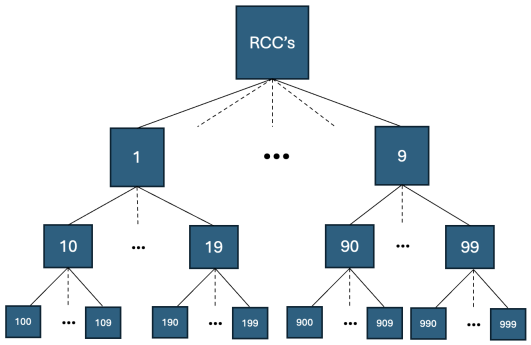


Figure 1: Ship Work List Item Number (SWLIN) hierarchy

Finally, any features and models utilized in the pipeline must be interpretable and actionable by Navy personnel. This precludes us from making use of complex feature transformations or opaque black-box techniques, such as deep neural networks [14].

Contributions. The DoMD estimation problem is studied as a regression problem with special attention paid to the temporality inherent in the data. The goal is to develop and optimize a data science pipeline which can effectively and efficiently generate interpretable features, select the most salient among them, propose an effective machine learning configuration, and train these supervised models to accurately and robustly estimate delay. Our main contributions are the following:

- We introduce and formalize of the Days of Maintenance Delay (DoMD) problem.
- We model the design of a data science pipeline which can be refit to raw data as an optimization problem.
- We study data management challenges associated with feature engineering and present a highly effective delay estimation framework inspired by time-series modeling [6] and ensemble learning [43].
- We present an experimental evaluation against baselines and show that our framework is highly effective. The framework has been deployed inside the Navy environment and currently being integrated with the user interface for use.

Accurately estimating DoMD is crucial for the US Navy, as each additional day of delay costs around $250,000, leading to budget overruns and reduced fleet availability. This framework allows for timely and interpretable delay predictions, helping decision-makers proactively allocate resources, adjust schedules, and mitigate risks before delays worsen.

2 DATA MODEL & PROBLEM DEFINITION

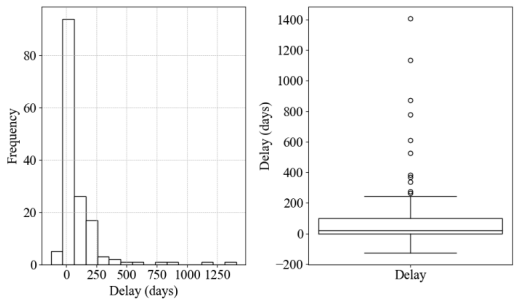


Figure 2: Delay distribution for all availabilities.

Availability. Each maintenance (or execution) period for a ship is referred to as an *availability* or *avail* for short. Each avail $a_i \in \mathcal{A}$ can be conceptualized at a high level by its identification number, planned start date, planned end date, actual start date, and actual end date as shown below.

$$a_i = \langle i, t_i^{planS}, t_i^{planE}, t_i^{actS}, t_i^{actE} \rangle$$

Using Table 1, there are 5 avails, where for avail ID 2, $t_2^{planS} = 5/7/2019, t_2^{planE} = 4/11/2020, t_2^{actS} = 5/7/2019, t_2^{actE} = 5/21/2021$.

avail id	ship id	status	plan start	plan end	actual start	actual end	delay
1	60	ongoing	8/20/23	12/4/24	8/20/23	-	-
2	246	closed	5/7/19	4/11/20	5/7/19	5/21/21	405
3	202	closed	7/18/18	6/11/19	7/18/18	6/11/19	0
4	1565	closed	3/1/21	11/8/22	3/1/21	12/17/22	39
5	1547	closed	1/31/20	8/19/20	2/27/20	8/19/20	-27

Table 1: Toy availability table with key attributes.

The full input data describing the availability for modeling is longitudinal in nature [9].

Notation	Interpretation
a_i, d_i	Avail with ID i, with delay d_i
$t_i^{\text{planS}}, t_i^{\text{planE}}$	Planned start, end date of a_i
$t_i^{\text{actS}}, t_i^{\text{actE}}$	Actual start, end date of a_i
$s_i^{\text{plan}}, s_i^{\text{act}}$	Planned, actual duration of a_i

Table 2: Table of key avail notation.

Delay. Delay d_i is calculated using the difference between an avail's planned (s_i^{plan}) and actual (s_i^{act}) duration . Here, s_i^{plan} (similarly s_i^{act}) is calculated using the difference between the planned (similarly actual) end date t_i^{planE} (similarly t_i^{actE}) and the planned (similarly actual) start date t_i^{planS} (similarly t_i^{actS}). That is, $s_i^{plan} = t_i^{planE} - t_i^{planS}$, $s_i^{act} = t_i^{actE} - t_i^{actS}$.

$$d_i = s_i^{act} - s_i^{plan}$$

This definition of delay does not take into account the start date and thus can be agnostic of late starting of the avail. Using Table 1, d_1 cannot be calculated as it is an ongoing avail, but could be estimated once the model is trained. For avail id $i = 2$, $d_2 = 405$, as $s_2^{act} - s_2^{plan} = 745 - 340 = 405$.

It follows that the delay of an avail is positive (or the avail is tardy) when the avail takes longer to complete than planned (i.e., $s_i^a < s_i^p$), zero when an avail finishes exactly as planned (i.e., $s_i^a = s_i^p$), and negative when an avail is completed earlier than planned (i.e., $s_i^a > s_i^p$).

As shown in Figure 2, the avails under study have delays ranging from 0 (on-time) to multiple years. While majority of avails finish within a few months of their projected end dates, Navy SME's have identified that achieving an MAE of 30 days for 80% of avails as a successful first milestone. Thus with this pipeline we aim to achieve this standard.

Request for contract change (RCC). The Request for Contract Change (RCC) is crucial for managing unplanned work and is characterized by a timestamp, RCC type, and Ship Work List Number (SWLIN). The RCC type falls into three categories: Growth (G), New Work (NW), or New Growth (NG), indicating whether the work upgrades existing systems, creates new ones, or adds distinct components. The SWLIN is an 8-digit hierarchical code (Figure 1) that identifies specific physical locations on the ship. The first digit represents the general subsystem, with subsequent digits providing more specific module details of that subsystem.

Notationally, each RCC $r_j \in R$ associated with an avail a_i is represented as the following sextuple:

$$r_j = \langle j, a_i, w_j, t_j^s, t_j^e, m_j \rangle$$

Here, j is an id showing type, w_j is the 8 digit SWLIN , t_j^s is creation date , t_j^e is settled date, and m_j is settled amount (dollar amount associated with RCC once settled). The creation and settled date correspond to when the RCC begins and ends, and the settled amount corresponds to the dollar amount associated with the RCC. These base attributes alone provide numerous ways of incorporating RCC trends into modeling as there are generally hundreds of such RCCs per avail. Using Table 3, the first RCC r_{1G} associated with avail $a_i = 5$, could be described as– $j = 1G$ (growth type), t_j^s= 3/22/20, t_j^e=6/16/20, w_j=434-11-001, m_j=8000.

avail id	RCC	creation date	settled date	status	workspec	amount
5	1G	3/22/20	6/16/20	settled	434-11-001	8000
5	32G	4/0820	7/9/20	settled	911-90-001	34520
5	76N	5/26/20	8/5/20	settled	804-11-001	56724
3	7NG	8/10/18	5/7/19	settled	983-11-001	22497
3	14N	11/8/18	11/29/18	settled	565-11-001	78339

Table 3: Example RCC table outlining key attributes and dates.

Logical time. In order to capture the temporal structure of this problem, we define "logical time" t^* for each physical time t, representing the % of planned maintenance duration an avail a_i is in at t, as shown in Equation 1.

$$t^* = \frac{t - t_i^{actS}}{s_i^{plan}} \times 100 \quad (1)$$

As an example, for avail ID 2, $t = 7/06/19$ would represent a logical time of $t^* = 18\%$.

We then discretize the entire planned maintenance duration into a set of time windows, each of width x% (x is an input parameter and could be set up per application need). This process involves training $\lceil \frac{100}{x} \rceil$ models to predict DoMD for 0% to 100% planned maintenance duration.

Notation	Interpretation
r_j, a_i	RCC with ID j, associated avail a_i
w_j	SWLIN code of RCC r_j
t_j^s, t_j^e, m_j	Creation date, settled date, associated cost of r_j

Table 4: Table of key RCC notation.

Attributes of an avail. Attributes of an avail are time-invariant (*static*) and time-dependent (*dynamic*): **a. Static attributes F_i^S.** Static attributes generally predate the execution of the avail and thus do not change across t^*. Some examples are the ship's class, the maintenance center, and any relevant planning features, such as planned duration. F_i^S are integral to the "base prediction" of the avail's delay before the availability actually begins. **b. Dynamic attributes F_{i,t^*}^D.** These are primarily RCC attributes and change as a function of t^*.

Problem 1. Overarching Problem. DoMD Queries. *Given a physical timestamp t and its corresponding logical timestamp t^*, a model gap interval x, a set $\mathcal{A}_q$ of input avails, produce estimates of delay $\hat{d}_{a_i,0}$, $\hat{d}_{a_i,0+x}$, $\hat{d}_{a_i,0+2x}$ ..., $\hat{d}_{a_i,t^*}$ as outputs at every x% of planned avail duration from the beginning of maintenance upto t^*, for each avail $a_i \in \mathcal{A}_q$.*

As an example, on t=4/12/2024, if $x = 10\%$ and $\mathcal{A}_q = a_1$, 6 different DoMD of a_1 are estimated starting from 0%, 10%, 20%, 30%, 40%, and 50% of planned duration.

DoMD Queries are answered by training a set of $1 + \lceil 100/x \rceil$ supervised machine learning models over the logical timeline, where model m_{0+ix} estimates delay at $0 + ix$ ($1 \leq i \leq \lceil 100/x \rceil$) over the logical timeline. A subset of avails identified as train set ($\mathcal{T}$) is used to train each model.

Problem 2. Constructing Modeling Pipeline. *Identify optimal parameter configuration $\hat{x} \in X$ defining the modeling pipeline $\mathcal{M}(\hat{x})$– which include feature selection method $\hat{s} \in S$ which produce a set F of features, a base model family and architecture $\hat{m} \in M$, a loss function for optimization $\hat{l} \in L$, a hyperparameter determination method $\hat{p} \in P$, and a ensembling technique $\hat{f} \in \mathcal{F}$ for fusion across the timeline– utilizing the training set $\mathcal{T}$ to fit any machine learning models and the validation set $\mathcal{V}$ to evaluate the parameter settings, such that DoMD estimations can be produced at all necessary logical times ($1 + \lceil 100/x \rceil$. Formally, the optimization problem is defined as:*

$$\hat{x} = \underset{(s,m,l,p,f) \in (S,M,L,P,\mathcal{F})}{\arg\min} \sum_{t^*} \sum_{a_i \in \mathcal{V}} |d_i - f(\{m(t, F(s), l, H(p)\}_{t=0}^{t^*})| \quad (2)$$

$$\hat{x} = (\hat{s}, \hat{m}, \hat{l}, \hat{p}, \hat{f})$$

where the final pipeline is $\mathcal{M}(\hat{x})$ and the objective is to minimize the sum of absolute errors between the true delay d_i and the predicted pr estimated delay $f(\{m(t, F(s), l, H(p)\}_{t=0}^{t^*})$, across the validation set $\mathcal{V}$ over the entire timeline.

3 DOMD ESTIMATION FRAMEWORK

We study two key components: (a) Feature Engineering: This involves the preparation and cleaning of raw data, followed by the creation of a large number of effective features. The process emphasizes efficiency, while ensuring that the features are also relevant. (b) Modeling pipeline: This is formalized in Problem 2.

3.1 Feature Engineering

TASK 1. Feature Engineering. *For every logical timestamp t^*, produce feature set F_{i,t^*} from the static and dynamic attributes, $F_{i,t^*} = \tilde{\mathcal{T}}(F_i^S, F_{i,t^*}^D)$. $\tilde{\mathcal{T}}$ is the transformation function applied on raw attributes to create features.*

Across the entire avail set, the resulting features can be thought of as a tensor across the avail, feature set, and logical time dimensions. Each model is trained on a slice of that tensor generated at discrete logical times t^*.

Transformation function $\tilde{\mathcal{T}}$ over RCCs. Given RCC $r_j =< j, a_i, w_j, t_j^s, t_j^e, m_j >$, we first categorize the RCC based on attributes like SWLIN, then within each group categorize the RCCs as "active", "settled", or "created" which is a combination of the prior two. We then utilize aggregation functions like average and sum over attributes of the RCC (e.g. settled amount, duration, percent active) to generate a large amount of features. For example, a feature may measure the average settled amount for G RCC's of SWLIN first digit 1, and will be called "G1-AVG_SETTLED_AMT". For this example, only "settled" RCC's of type G and SWLIN code 1 will be included in the computation. See tech report [41] for further details.

```
SELECT
    RCC.ids,
    RCC.settled_amount,
    RCC.settled_date - RCC.creation_date AS
        duration
FROM
    RCC_table AS R
WHERE
    creation_date <= t^* AND
    settled_date <= t^* OR settled_date >= t
        ^*
GROUP BY
    RCC_types, SWLIN_Level_no,  ...
```

Figure 3: `Status Query`

A novelty of our proposed solution is to abstract feature engineering through a generic retrieval task, refer to as `Status Query`. Irrespective of the specific details of $\tilde{\mathcal{T}}$, `Status Queries` are repeatedly invoked in the abstract form shown in Figure 3.

Section 4 discusses efficiency opportunities in processing `Status Queries`.

3.2 Modeling Pipeline

Clearly, identifying all parameters of $\mathcal{M}(\hat{x})$ at the same time gives rise to a prohibitively large combinatorial search space. This problem lends itself to the classical experiment design problem [1], which is known to be NP-hard. For computational efficiency, we employ a greedy modeling pipeline design, where we solve each step locally one after the other. The problem is agnostic of the

order of optimization, but we choose the order intelligently to limit the size of the search space.

In order to do this sequential optimization, we assume default parameter values for each before a parameter has been optimized. As the default fusion method is no fusion, we drop the set notation for most of the optimization definitions for simplicity.

3.2.1 Feature Selection

The problem of selecting an optimal subset of k features is NP-hard [22], we resort to greedy optimization methods to find an approximate solution to Task 2.

TASK 2. Determining a feature selection method. *Given the set of all generated features F_{i,t^*} at timestamp t^* and a set of feature selection methods $s \in S$ which assign scores to the feature set, sort the feature set based on each score, and return the features with the top k scores, produce a set of k feature as follows:*

$$\hat{F}_{i,t^*}(s) = s(F_{i,t^*}), |F_{i,t^*}(s)| = k$$

Select an optimal scoring method which– with default modeling architecture m^0, loss function l^0, and hyperparameters H^0– minimizes the absolute validation error over the validation set $\mathcal{V}$:

$$\hat{s} = \arg\min_{s \in S} \sum_{t^*} \sum_{a_i \in \mathcal{V}} |d_i - f^0(m^0(t^*, F_{i,t^*}(s), l^0, H^0)|$$

We study both model-dependent (dependent on predictions from the underlying choice of model,e.g., Recursive Feature Elimination) and model-agnostic (independent of underlying model choice e.g., Pearson Correlation Coefficient, Spearman's Rank Coefficient, Mutual Information Coefficient) state-of-the-art feature selection methods [8, 12, 15, 26, 30, 37]. Feature selection is only applied to generated features, allowing for important static features to be included by default.

3.2.2 Base Model and Modeling Architecture

The machine learning architecture for this effort must be structured to handle the complexity and high-dimensionality inherent in the availability data, while also preventing overfitting. Our goal is to first choose a base model family (Linear Regression, Gradient Boosted Trees, etc.), and then deciding on their configuration (stacked or nested architecture, etc.).

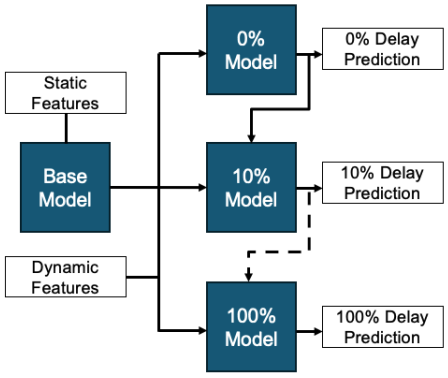


Figure 4: Stacked modeling and recursive prediction.

TASK 3. Selecting the optimal machine learning architecture. *Given a set of k features $\hat{F}_{i,t^*}$ generated using the selected scoring method $\hat{s}$, select a machine learning model family and architecture from set M– with default loss function l^0, and hyperparameters H^0– which minimizes the absolute delay prediction error over the validation set $\mathcal{V}$:*

$$\hat{m} = \arg\min_{m \in M} \sum_{t^*} \sum_{a_i \in \mathcal{V}} |d_i - f^0(m(t^*, F_{i,t^*}(\hat{s}), l^0, H^0)|$$

We also propose a stacked model architecture [3, 27] (Figure 4) to estimate delay at any given point over the logical time. The stacked architecture employs a base or "static" model, which takes as input all static features and several dynamic or "timeline" models which take as input all features which change throughout the course of the availability as well as predictions from the static model.

3.2.3 Loss Function in Training

The choice of loss function directly influences the model's ability to minimize estimation error and adapt to data variance [13]. ℓ_2[40] Loss is the default loss function but generally recognized as being heavily sensitive to outliers, this loss function may not be the optimal choice for a dataset with such high relative variance in the response. ℓ_1[40] or absolute loss, on the other hand, is also appropriate in this context as it penalizes absolute differences between predicted and true values. Compared to ℓ_2, ℓ_1 loss is less sensitive to outliers. Huber loss[40] is an alternative option, which provides a compromise between squared and absolute losses. The Huber loss is less sensitive to outliers ℓ_2 and approaches ℓ_1 when the error is large, making it particularly well-suited for datasets with both small, frequent deviations and occasional large outliers.

Formally, the Huber loss function is defined as follows (where $x = |d_i - \hat{d}_{i,t^*}|$):

$$l_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \le \delta, \\ \delta(x - \frac{1}{2}\delta) & \text{otherwise,} \end{cases}$$

where x is the difference between the predicted and actual value, and δ is a threshold parameter that controls the point where the loss transitions from quadratic to linear [13]. The flexibility of δ allows for fine-tuning the model's sensitivity to outliers, ensuring robustness.

TASK 4. Determining a loss function. *Given a set of features $\hat{F}_{i,t^*}$ as well as a modeling architecture $\hat{m}$ with default hyperparameters H^0, select a loss function from a set L which minimizes the loss in the validation set $\mathcal{V}$:*

$$\hat{l} = \arg\min_{l \in L} \sum_{t^*} \sum_{a_i \in \mathcal{V}} |d_i - f^0(\hat{m}(t^*, F_{i,t^*}(\hat{s}), l, H^0))|$$

3.2.4 Hyperparameter Tuning

We design a fully automated hyperparameter tuning (AutoHPT) module which efficiently navigates the high-dimensional hyperparameter space using Bayesian chains. The key parameters to optimize are described in detail in Section 5.

TASK 5. Determining a hyperparameter selection method. *Given a set of availabilities $\mathcal{A}$, a hyperparameter selection method p should take as input the set of data and utilize the modeling architecture to return a set of hyperparameters as follows:*

$$\hat{H}(p) = p(\mathcal{A})$$

Thus, the challenge is to produce a hyperparameter determination method p from set P which minimizes the prediction error on the validation set $\mathcal{V}$:

$$\hat{p} = \arg\min_{p \in P} \sum_{t^*} \sum_{a_i \in \mathcal{V}} |d_i - f^0(\hat{m}(t^*, F_{i,t^*}(\hat{s}), \hat{l}, H(p)))|$$

We combine Tree-structured Parzen Estimation (TPE) and Sequential Model-based Optimization (SMBO) [4][17][31] for this, to optimize efficacy with minimal computational overhead. We first determine the hyperparameter values, following which we identify the number of optimization runs.

3.2.5 Fusion of Delay Estimation

Given that delays tend to compound over time, a key challenge is how to effectively incorporate information from earlier logical time predictions into subsequent models. This introduces a critical dependency across time steps, making the problem more complex than a standard regression task. The goal of this task is to identify the fusion approach, as formalized below.

TASK 6. Selecting ensembling method. *Select an ensembling function $f \in \mathcal{F}$ – which at time t^* takes as input all DoMD predictions up to time t^* and returns a single fused prediction– that minimizes absolute error across the validation set $\mathcal{V}$:*

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{t^*} \sum_{a_i \in \mathcal{V}} |d_i - f(\{\hat{m}(t, \hat{F}_{i,t}, \hat{l}, \hat{H})\}_{t=0}^{t^*})|$$

We experiment with minimum fusion, which takes the minimum prediction over the timeline and average fusion, which takes the average of all predictions over the timeline. There are many other possible ensembling methods but we leave these for future work.

4 EFFICIENT STATUS QUERY PROCESSING

We study 3 efficiency challenges of `Status Query`: how to search over the logical timeline and retrieve RCC's that satisfy dates, how to compute the group by's over SWLIN and RCC types, and how to enable incremental computation over t^*?

The Status Query is repeated several times throughout the pipeline to generate features and thus is directly related to the latency of the pipeline. Low-latency queries are critical because inefficiencies in this step would be compounded, hindering decision-making based on the most current data.

4.1 Designed Indexes

Note that a `Status Query` (Refer to Query in Figure 3) has two predicates involving logical time - RCC creation date and RCC settled date.

We present three designs for index structures $\mathcal{R}$ that allow efficient retrieval over t^*. These designs are based on interval trees, AVL balanced trees, and generic table joins, respectively. These methods should store the start, end, and ID of each RCC as shown:

$$(t^*_{start}, t^*_{end}, ID)$$

They should also allow for the querying of the following sets of RCC's at any specified logical time step.

$$R^A_{t^*} = \mathcal{R}[\text{point query @ } t^*] \quad (3)$$

$$R^S_{t^*} = \mathcal{R}[\text{overlap query @ } [-\inf, t^*)] \quad (4)$$

$$R^C_{t^*} = \text{union}(R^A_{t^*}, R^S_{t^*}) \quad (5)$$

$$R^N_{t^*} = \text{difference}(R, R^C_{t^*}) \quad (6)$$

The interval tree is used to hold the RCC duration (.e., the difference between the settled and creation date of RCCs) and allows the efficient search of all intervals that overlap with any given interval or point. These trees have an initial creation time of $O(n \log n)$ and takes $O(n)$ space. n corresponds to unique RCC intervals, which is $O(|RCC|)$. After creation, interval trees may be dynamic, allowing for efficient insertion and deletion in $O(\log n)$ time [32].

We also consider the AVL tree index as an alternative. AVL trees are self balancing binary search trees designed for efficient searching of RCCs over the logical timeline. Its look up, insertion, and deletion time are also similar to that of an interval tree. We utilize one AVL tree for start times and another for end times to allow for efficient maintenance of the index.

A naive index, on the other hand, joins the avail table with the RCC table, stores it, and performs subsequent sorting, as needed. Theoretically, this cost $O(|RCC|)$ time and $O(|RCC|)$ space.

4.2 `Status Query` Processing Algorithm

Algorithm `StatusQ` (Algorithm 1) is designed to process `Status Queries` containing group by's and logical timestamp. It leverages the group by index structures RCC-Type-Tree $\mathcal{R}$ and SWLIN tree $\mathcal{ST}$ to retrieve the subtree that satisfies the group by predicates. Then, it uses the logical times based index structures $\mathcal{T}$ to retrieve RCC's that satisfy t^* for each node $M \in R^M$. It also makes use of incremental computation, as we discuss later.

Algorithm 1 Algorithm `StatusQ`

Require: t^*: `Status Query` Q, RCC-Type-Tree $\mathcal{T}$, SWLIN tree $\mathcal{ST}$, A: avails table, Logical time index structure $\mathcal{R}$

Ensure: R^M: subtree of hierarchies specified in the Group By conditions, $R^{C^*_t}$: RCC results produced according to the `Status Query`

$R^M \leftarrow$ Group Bys($\mathcal{T}$,$\mathcal{ST}$)
for all nodes M in R^M **do**
  $R^C_{t^*} \leftarrow \mathcal{R}(t^*)$processed according to index
  $R^{C^*_t} \leftarrow \{R^{C^*_t} \cap A\}$
end for

4.3 Incremental Computation

Recall Problem 1, where the goal is to produce estimates of delay of an avail a_i at $0+x$, $0+2x$, . . ., upto t^* as outputs - i.e., estimate delay at every x% of planned avail duration from the beginning of maintenance upto t^*. This requirement translates to performing repeated `Status Queries` progressively over the logical timeline, such that between two consecutive logical timestamps (e.g., $0, 0+x, 0+2x, ..$) there is only an additional x% of additional logical time for which `Status Query` predicates are to be retrieved. A naive approach would be to ignore the previous computations performed upto $0+(j)x$ and do everything from scratch at $0+(j+1)*x$. Algorithm `StatusQ` reuses $StatStructure(t^*_{xj})$ containing `Status Query` results upto time t^*_{xj} and only invokes `Status Query` for a much shorted logical time interval $(j, j+1)*x$ when $t^* = (j+1)*x$.

5 EXPERIMENTAL EVALUATION

The implementations and experiments are done using Python 3.11 on a MacOS Ventura, 8GB RAM, 512GB SSD M3 Silicon Chip, 8-core CPU, 10-core GPU. Results are presented as the average of 3 runs.

5.0.1 Datasets

The real data comes from the Navy Maintenance data (NMD) and contains two large tables. The avail table contains availability information of the ships and other details. The RCC table contains RCC information of those avails. Table 5 has further details.

Synthetic datasets. In addition to that, a synthetic dataset is created for the RCC table, where the temporal distribution of the RCCs are kept intact - only the number of RCCs of each type and SWLIN is increased by x folds [1]

Dataset	# Rows	Columns
Avail	190	73
RCC	52,959	187

Table 5: Statistics of the real dataset

5.1 Scalability Experiments

We evaluate the scalability of `Status Query` presented in Section 3.

Implemented Algorithms We implement naive solution (offered by Pandas merge library[42]), and compare that with the indexes proposed in Section 4.

Measures. We measure indexing creation cost (memory and time), as well as query processing cost.

5.1.1 Index Creation Cost

We measure the index creation time (in seconds) as the running time of the index structures plus processing time that would not be necessary without the indexes. We also measure the memory footprint of the proposed indexing method.

Dataset Scaling Factor	Memory Usage (MB)		
	Pandas Merge	AVL Tree	Interval Tree
Original	57.3	28.1	29.6
5x	274.7	137.6	146.4
10x	547.8	273.8	285.3
15x	820.8	410.0	427.0
20x	1090.0	556.1	578.5

Table 6: Index construction cost considering space.

Figure 5a and Table 6 show these results with increasing size of RCCs (20x denotes $20 \times 52,959$), the AVL tree turns out to be the winner unanimously benefiting from significantly lower creation time and memory utilization compared to the other indexes. AVL tree exhibits an order of magnitude reduction in index creation time and the twofold reduction in memory utilization for the AVL tree compared to the naive solution.

The actual indexing time of the interval tree takes longer than its theoretical runtime. Upon further investigation, we realize that this mismatch is purely due to implementation differences. Both AVL tree and Pandas merge methods are optimized for C and Cython, while there is no such existing implementation for interval tree. We leave this exploration for further work.

5.1.2 Query Processing Cost

Figure 5b presents the query processing time, whereas, Figure 5c shows the index creation plus query processing time. As depicted in Figure 5b, the AVL approach with incremental computation enjoys a 5x increase in runtime over the Pandas Merge method. This is massively important in developing a scalable data science pipeline, becuase DoMD queries must be answered with the least latency. Likely for reasons mentioned in the previous section, the interval tree implementation does diverge in runtime from its expected behavior. These figures corroborate the effectiveness of our indexing techniques. Incorporating this approach allows DoMD predictions to be generated with a low latency and a small memory footprint.

[1]The code and data is subject to stringent controls of the US govt and can not be shared publicly.

5.2 Modeling Experiments

In this section, we present the experiments carried out to optimize the parameters of the modeling pipeline.

5.2.1 Experimental Set up

Data Splits. We first carve out a test set of 30% recent avails as test set. From the rest of the 70% of avails, we take a random sample with 25% of the avails used for validation and 75% used for training. We utilize the validation set to set various pipeline parameters and the training set to fit the machine learning models.

Feature Set. We deal with 8 static features that do not change over time, such as ship class, RMC id, ship age, etc. We have 1490 RCC-dependent features.

Implemented Algorithms. We implement Recursive Feature Elimination (RFE), Pearson Correlation, Spearman Rank, Mutual Information, and Random Selection.

The machine learning models used to model DoMD are extreme Gradient Boosting Trees (XGBoost) [36] and Linear Regression. For stacking, we implement stacked and non-stacked architectures - the stacked architecture develops base models with static features and estimation delay only based on those. Then, it uses that estimated value as a feature of another model which utilizes RCC dependent features. The non-stacked architecture combines both static and RCC dependent features inside the same model. For fusion, we implement aggregation functions such as average and minimum, and compare that when no fusion is done.

Evaluation Measures. The efficacy of the framework is validated capturing different aspects of quality, such as general error magnitude (MAE), error sensitivity to larger deviations (MSE, RMSE), and the overall goodness-of-fit (R^2) [12].

Pertinent Parameters. Feature selection methods: Recursive Feature Elimination, Pearson Correlation, Spearman Rank, Mutual Information, and Random Selection. Feature set sizes (k): 20 to 100 features, incremented by 10 features. Base machine learning models: XGBoost and Linear Regression. Machine learning modeling architectures: stacking and no stacking structures. Loss functions: ℓ_1, ℓ_2, Huber loss. Number of hyperparameter tuning optimization runs: [10, 20, 30, 40, 50, 100, 200]. Fusion technique: no fusion, min fusion, average fusion.

5.2.2 Determining Modeling Pipeline Parameters

The following study details a systematic experimental process to determine different parameter values by measuring the efficacy of the trained models over the validation set using MAE.

Finding Feature Selection Method and Feature Set size. Figure 6a demonstrates the efficacy of different feature selection methods for increasing feature set sizes, denoted by k. These include Recursive Feature Elimination (RFE), Pearson Correlation, Spearman Rank, Mutual Information, and Random Selection. Pearson Correlation method consistently achieves the best MAE throughout the entire planned duration and reaches its optimal

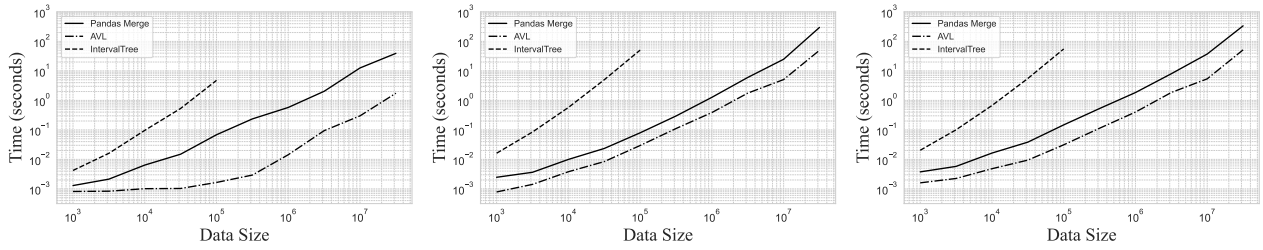


Figure 5: Scalability of indexing techniques (a) index creation time, (b) Query processing time, and (c) total time.

efficacy at $k = 60$ features. Consequently, we select Pearson Correlation with $k = 60$ as the optimal feature selection approach.

Identifying Base Model We compare XGBoost with Linear Regression. The latter is tuned with Elastic-Net - which uses both ℓ_1 and ℓ_2 for regularization. These two model families are selected as primary candidates to the nature of the problem, and we choose Elastic Net regression as a simpler model family, and XGBoost as a more powerful and SOA choice.

We train these models using $k = 60$ different features, where the features are selected using Pearson Correlation. The results in Figure 6b demomstrate XGBoost as the preferred base model, given its ability to handle non-linear relationships and complex interactions within the data.

Identifying the Effect of Stacking Figure 6c presents the MAE of validation set, comparing stacked vs. non-stacked architecture, where the base models are XGBoost, where $k = 60$ best features are selected using Pearson Correlation method for each model over the logical timeline. The results demonstrate that non-stacked architecture outperforms its counterpart. Consequently, we select non-stacked architecture as the winner from this stage.

Identifying Loss Functions In this experiment, we retain the previously chosen model parameters (XGBoost, no stacking, Pearson correlation with 60 features) but evaluate the effectiveness of different loss functions. In figure 6d, we assess each loss function's impact on the Mean Absolute Error (MAE) of the validation set. Squared Error (ℓ_2) heavily penalizes outliers, making it highly sensitive to outliers in contrast to Absolute Error (ℓ_1) which provides a more stable estimation by reducing the influence of outliers. Serving as a middle ground, Pseudo Huber Error loss function combines the benefits of both ℓ_1 and ℓ_2 by applying a smoother penalty curve, especially when tuning its threshold parameter δ. To optimize Pseudo Huber's efficacy, we tune its δ parameter to $\delta = 18$, which demonstrates a consistent improvement. Thus, we move proceed the Pseudo Huber Loss function with $\delta = 18$ as the loss function from this point on.

Identifying Hyperparameters For hyperparameter tuning, we employ a Bayesian optimization to fine-tune XGBoost's hyperparameters. In our study, we keep the currently chosen model parameters consistent, vary # trails, and observe their impact on MAE. Table 6e shows that increased # trials results in a declining MAE, which may indicate overfitting, where the model becomes overly complex, ultimately impairing its ability to generalize to new and unseen data. As a result, we decide 30 to be the number of trails for each model and decide the hyperparameter values accordingly.

Fusing Multiple Models We explore how the trained models fused considering different aggregation or "fusion" behave on the validation set. Figure 6f presents those results. Based on this, we select average fusion as the preferred way of performing this step.

Selected modeling pipeline parameters The following parameters are selected overall: (1) Pearson Correlation for selecting $k = 60$ features for each model.(2) eXtreme Gradient Boosting (XGBoost) as base model. (3) Non-stacked architecture. (4) Pseudo Huber ($\delta = 18$) loss function.(5) Obtain hyperparameters after 30 trails. (6) Average as the fusion technique.

5.2.3 Evaluation on Test Set

Table 7 shows MAE values at different percentiles. On an average, for 80% of avails, the MAE is 19.99 days, for 90% of avails that is 27.52 days and for all avails in the test set is below 40 days. These results demonstrate the effectiveness of the proposed solutions.

5.2.4 MAE, MSE, RMSE, R^2

Table 7 also presents MAE, RMSE, MSE, and R^2 measures on the test set. These results demonstrate, on an average, our current DoMD estimation is off by 38.97 days (MAE), giving rise to MSE of 3159.96 ($days^2$), RMSE of 56.14 days. Our produced pipeline generates a suite of predictors explaining over 88% of the variation.

Logical Time (%)	Qualitative measures					
	MAE 80th	MAE 90th	MAE 100th	MSE	RSME	R^2
0	26.82	34.44	43.23	3210.11	56.66	0.88
10	21.62	30.06	41.90	3622.78	60.19	0.86
20	20.00	28.90	42.00	3926.44	62.66	0.85
30	17.51	25.69	38.59	3447.48	58.72	0.87
40	16.05	23.65	35.72	2973.63	54.53	0.89
50	17.86	25.22	36.80	2953.07	54.34	0.89
60	19.26	26.40	37.63	2933.56	54.16	0.89
70	20.51	27.37	38.45	2958.67	54.39	0.89
80	20.21	27.14	38.28	2943.48	54.25	0.89
90	20.20	27.17	38.29	2933.11	54.16	0.89
100	19.90	26.69	37.80	2857.26	53.45	0.89
Average	19.99	27.52	38.97	3159.96	56.14	0.88

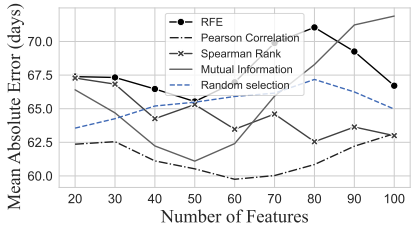
Table 7: Estimation quality over timeline on test set

5.2.5 Summary of Results

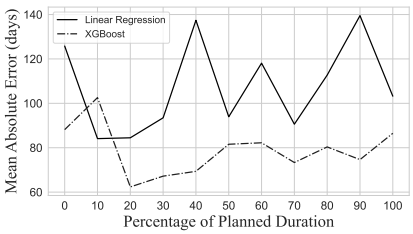
The evaluation on the test set highlights the following:

Robust estimation. The MAE remains consistently below 43.23 days across all avails, with lower values achieved for 80% and 90% of the data. This reflects the performance of the chosen configurations of our modeling pipeline, which displays reliable accuracy across different percentiles.

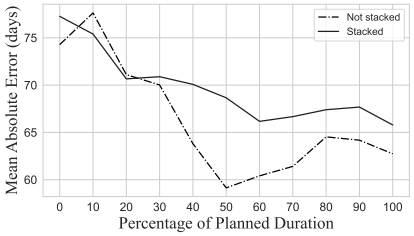
Interpretability of Models. Beyond achieving a high average R^2 of 0.88, our framework is designed to provide interpretable outputs that facilitate user understanding and decision-making. To assess user acceptability, we engaged subject matter experts (SMEs) in the Navy maintenance domain to evaluate the model's predictions. This evaluation includes a review of the top contributing features for each availability, enabling SMEs to validate



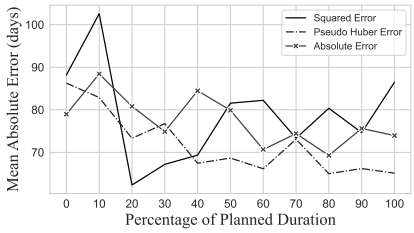
(a) Efficacy varying feature selection methods & k at 50% planned duration.



(b) Efficacy varying base models.



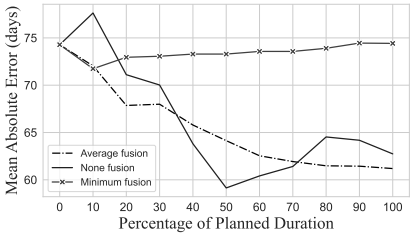
(c) Efficacy varying stacking vs. non-stacking.



(d) Efficacy varying loss functions.

Trials	Validation MAE (in days)
10	86.49
20	71.65
30	69.22
40	71.00
50	72.60
100	73.74
200	78.69

(e) Efficacy varying number of trials in hyperparameter tuning at 50% plannned duration.



(f) Efficacy varying fusion techniques

Figure 6: Experiments to determine modeling pipeline.

whether the most influential factors align with their domain expertise.

Specifically, our model surfaces the top-5 contributing features for each availability, allowing users to assess the primary drivers of predicted delays. This feature importance analysis ensures transparency and supports actionable insights, as SMEs can trace the rationale behind each prediction. Initial qualitative feedback suggests that this interpretability aids in practical decision-making, reinforcing the model's usability in real-world naval maintenance planning.

Balanced Error Metrics. The relatively low MSE and RMSE suggest that our solution strikes a good balance between minimizing errors and maintaining overall accuracy.

Effective temporal estimation. Error measures stabilize through the planned duration, confirming that the proposed framework is robust over timeline.

6 RELATED WORK

Our work falls under the umbrella of data management for machine learning for specialized application domain. Data science pipelines have been designed for specialized application domains in the past, including healthcare, e-commerce, manufacturing, software engineering, to name a few [5, 7, 16, 18, 25, 28]. However, there is no trivial adaptation of these frameworks to our problem due to two stark reasons - a. the data is purely longitudinal and obfuscated, highly wide, sparse, and the number of instances are small. b. the designed models need to be interpretable to the navy users.

This proposed work also relates to Automated Machine Learning or **AutoML** [10, 19–21, 24, 35]. The field of automated machine learning is rich with examples of work that seek to automate parts of the creation of data science pipelines, such as [11, 23, 33]. For example, TPOT [24] uses tree-based optimization and genetic programming to optimize a series of feature preprocessors and machine learning models. However, in our setting, a fully automated approach is neither necessary nor satisfactory for several reasons. First, the dataset used in this work is not only short yet wide but is also obfuscated to protect sensitive information, creating challenges that are not typical in most AutoML applications. Second, feature engineering involving RCCs require intricate domain knowledge, which fully automated approach can not achieve.

In addition to AutoML, we draw heavily from the field of **feature engineering** [2, 34, 39]. Feature engineering is essential for extracting meaningful insights from high-dimensional datasets, and while previous works [2, 34, 39] provide significant inspiration, they are not directly applicable in our context. The complexity of Navy ship maintenance data requires the development of novel, domain-specific features that address both the temporal and hierarchical structure of the data. Our feature set is uniquely designed to capture the nuances of Request for Contract Change (RCC) types, milestone delays, and dynamic progress updates, which are essential for predicting delays in ship availability.

7 CONCLUSION & FUTURE WORK

This work proposes a computation framework to estimate Days of Maintenance Delay (DoMD) of US Navy ships at anytime during the actual maintenance. Our application involves a small number of samples and a very high dimensional time-dependent feature space. We present a new class of predictive maintenance framework and investigate several data management challenges.

We believe the research challenges investigated in the work are likely to adapt to other application domains as well, including aircraft and spacecraft maintenance, manufacturing applications, such as, maintaining pumps, motors, conveyor belts, as well as automotive and rail. As an ongoing work, we are studying the feasibility of our solution on other manufacturing data.

ACKNOWLEDGMENT

The research conducted by NJIT researchers is funded through grants provided by the Office of Naval Research, with award numbers N000142112966 and N000142412466.

REFERENCES

[1] Zeyuan Allen-Zhu, Yuanzhi Li, Aarti Singh, and Yining Wang. 2017. Near-optimal design of experiments via regret minimization. In *International Conference on Machine Learning*. PMLR, 126–135.

[2] Michael R Anderson and Michael Cafarella. 2016. Input selection for fast feature engineering. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 577–588.

[3] Senjuti Basu Roy, Ankur Teredesai, Kiyana Zolfaghar, Rui Liu, David Hazel, Stacey Newman, and Albert Marinez. 2015. Dynamic hierarchical classification for patient risk-of-readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1691–1700.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (Eds.), Vol. 24. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[5] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2022. The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large. In *Proceedings of the 44th International Conference on Software Engineering*. 2091–2103.

[6] George.E.P. Box and Gwilym M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. Holden-Day.

[7] Xiaoyu Chen and Ran Jin. 2018. Data fusion pipelines for autonomous smart manufacturing. In *2018 IEEE 14th international conference on automation science and engineering (CASE)*. IEEE, 1203–1208.

[8] Xue-wen Chen and Jong Cheol Jeong. 2007. Enhanced recursive feature elimination. In *Sixth international conference on machine learning and applications (ICMLA 2007)*. IEEE, 429–435.

[9] Peter Diggle. 2002. *Analysis of longitudinal data*. Oxford university press.

[10] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074* 24 (2020), 8.

[11] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf

[12] Jiawei Han, Micheline Kamber, and Jian Pei. 2012. Data mining concepts and techniques third edition. *University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University* (2012).

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA.

[14] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[15] Nguyen Thi Thao Ho, Torben Bach Pedersen, et al. 2022. Efficient temporal pattern mining in big time series using mutual information. *Proceedings of the VLDB Endowment* 15, 3 (2022), 673–685.

[16] Erik Johannes Husom, Simeon Tverdal, Arda Goknil, and Sagar Sen. 2022. UDAVA: An unsupervised learning pipeline for sensor data validation in manufacturing. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. 159–169.

[17] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization* (Rome, Italy) *(LION'05)*. Springer-Verlag, Berlin, Heidelberg, 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[18] Resham Jhangiani, Doina Bein, and Abhishek Verma. 2019. Machine learning pipeline for fraud detection and prevention in e-commerce transactions. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 0135–0140.

[19] Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. 2021. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–36.

[20] Teddy Lazebnik, Amit Somech, and Abraham Itzhak Weinberg. 2022. Substrat: A subset-based optimization strategy for faster automl. *Proceedings of the VLDB Endowment* 16, 4 (2022), 772–780.

[21] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. ICML San Diego, CA, USA.

[22] George Nemhauser, Laurence Wolsey, and M. Fisher. 1978. An Analysis of Approximations for Maximizing Submodular Set Functions—I. *Mathematical Programming* 14 (12 1978), 265–294. https://doi.org/10.1007/BF01588971

[23] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (Denver, Colorado, USA) *(GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 485–492. https://doi.org/10.1145/2908812.2908918

[24] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.

[25] Randal S Olson, Ryan J Urbanowicz, Peter C Andrews, Nicole A Lavender, La Creis Kidd, and Jason H Moore. 2016. Automating biomedical data science through tree-based pipeline optimization. In *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30–April 1, 2016, Proceedings, Part I 19*. Springer, 123–137.

[26] Patrick Pantel, Andrew Philpot, and Eduard Hovy. 2005. Aligning database columns using mutual information. In *Proceedings of the 2005 national conference on Digital government research*. Citeseer, 205–210.

[27] Bohdan Pavlyshenko. 2018. Using stacking approaches for machine learning models. In *2018 IEEE second international conference on data stream mining & processing (DSMP)*. IEEE, 255–258.

[28] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Jordan Henkel, Matteo Interlandi, Subru Krishnan, Brian Kroth, Venkatesh Emani, Wentao Wu, Ce Zhang, et al. 2022. Data science through the looking glass: Analysis of millions of github notebooks and ml. net pipelines. *ACM SIGMOD Record* 51, 2 (2022), 30–37.

[29] Sean Rhea, Eric Wang, Edmund Wong, Ethan Atkins, and Nat Storer. 2017. Littletable: A time-series database and its uses. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 125–138.

[30] Md Abdus Salam, Senjuti Basu Roy, and Gautam Das. 2023. Efficient approximate top-k mutual information based feature selection. *Journal of Intelligent Information Systems* 61, 1 (2023), 191–223.

[31] Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. *CoRR* abs/1907.10902 (2019). arXiv:1907.10902 http://arxiv.org/abs/1907.10902

[32] Jens M Schmidt. 2009. Interval stabbing problems in small integer ranges. In *Algorithms and Computation: 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings 20*. Springer, 163–172.

[33] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1171–1188. https://doi.org/10.1145/3299869.3319863

[34] Qitao Shi, Ya-Lin Zhang, Longfei Li, Xinxing Yang, Meng Li, and Jun Zhou. 2020. Safe: Scalable automatic feature engineering framework for industrial tasks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1645–1656.

[35] Micah J Smith, Carles Sala, James Max Kanter, and Kalyan Veeramachaneni. 2020. The machine learning bazaar: Harnessing the ml ecosystem for effective system development. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 785–800.

[36] Chen Tianqi and Guestrin Carlos XGBoost. [n. d.]. A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD* 1143 ([n. d.]), 785–794.

[37] Michael Vollmer, Ignaz Rutter, and Klemens Böhm. 2018. On Complexity and Efficiency of Mutual Information Estimation on Static and Dynamic Data.. In *EDBT*. 49–60.

[38] Chen Wang, Xiangdong Huang, Jialin Qiao, Tian Jiang, Lei Rui, Jinrui Zhang, Rong Kang, Julian Feinauer, Kevin A McGrail, Peng Wang, et al. 2020. Apache IoTDB: Time-series database for internet of things. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2901–2904.

[39] Kafeng Wang, Pengyang Wang, and Chengzhong Xu. 2023. Toward Efficient Automated Feature Engineering. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1625–1637.

[40] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. 2020. A comprehensive survey of loss functions in machine learning. *Annals of Data Science* (2020), 1–26.

[41] Gerald White et al. 2024. Tech Report. https://www.dropbox.com/scl/fo/r0qz3fbcomszklt3up4n0/ABat292W-1YGz94-Zr3EBqw?rlkey=tiwhid0ukpgx0wf6ubnotng6k&dl=0.

[42] Art Yudin and Art Yudin. 2021. Data Analysis with Pandas. *Basic Python for Data Management, Finance, and Marketing: Advance Your Career by Learning the Most Powerful Analytical Tool* (2021), 93–150.

[43] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms* (1st ed.). Chapman & Hall/CRC.