École Polytechnique

BACHELOR THESIS IN COMPUTER SCIENCE

Feature-incremental machine learning via robust optimization

Author:

Yubo Cai, École Polytechnique,

Bachelor of Science in Mathematics and Computer Science

Advisor:

Vassilis Digalakis Jr, HEC Paris ISOM

Michael Lingzhi Li, Harvard Business School

Academic year 2023/2024

Abstract

We consider the problem of feature-incremental machine learning, where features that are input to a trained machine learning model arrive sequentially and in unknown order. We address this issue primarily within medical diagnostics contexts, where the varying time requirements of different medical tests result in sequential data arrival. In the paper, we discuss an exact solution for feature-incremental machine learning and its practical limitations due to computational complexity. Consequently, we propose an alternating optimization approach to tackle robust optimization in Feature-incremental ML. Our framework offers versatility and can be integrated with any machine-learning model or package. We conduct a numerical evaluation of our framework on publicly available real-world datasets from the UCI repository using linear and logistic regression models and boosted trees. Our approach demonstrates a 10%-45% improvement over the naive approach and manages to keep the gap with the exact solution within 10%, while significantly reducing computation time.

Code description

Please find our code in this link. Considering our intention to further expand and publish the paper, we have not made the code publicly available on GitHub. Therefore, we've shared all relevant code in this Google Drive.

All information about the code and how to replicate the experiment results is clearly explained in the README.md file. Additionally, all experimental models are saved in the folder numerical_result to prevent the necessity of running the entire experiment again, which could be time-consuming.

Acknowledgments

I extend my deepest gratitude to Professors Vassilis Digalakis Jr and Michael Lingzhi Li for their invaluable guidance during my thesis journey. Their steadfast support was crucial to the completion of this paper. In addition, I would like to thank the HEC Paris ISOM department and Kristen Rakow for all the support and assistance they provided during my internship. Furthermore, I owe a special thank you to my family, friends and the Bachelor administration team.

Contents

1 Introduction

In numerous real-world applications, data is usually gathered from open and dynamic environments leading to an increase in both the volume of data points and the dimensions of features, thereby giving rise to situations involving *dynamic feature spaces* [Kaya et al., 2020, Ni et al., 2024].

Our research motivation originates from certain scenarios in healthcare. In disease diagnosis, the data from patient test results and individual physical information increases over time (for instance, information such as patient age, weight, fasting blood sugar, and serum cholesterol can be known in a **short** period, while CT scan results, routine blood test data, etc., require a **longer** time), forming a continuous sequence in the acquisition of feature information under such circumstances. Most supervised learning models require that all input features are available for every data point, which is difficult to achieve in datasets for real-world problems. Therefore, we aim to develop an algorithm and machine learning model that can make **accurate** and **robust** predictions and decisions in the event of missing features (for example, missing columns in the dataset) and can achieve better outcomes as **more features** become available.

1.1 Initial Problem Formulation

Assume that we have the dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n\}$, [1] where $\mathbf{X}$ represents the matrix of **data points** and $\boldsymbol{y}$ represents the **labels**. For a statistical perspective, we denote upper case $\boldsymbol{X} := X_1, \ldots, X_p$ and Y as denoting features (i.e., random variables/column names) and a set of features $S_{\boldsymbol{X}} = \{X_1, \ldots, X_p\}$. We will assume without loss of generality that the machine learning model operates on sets of features (i.e., permutation invariant), which can be easily accommodated for all models. Then, all order of the feature set is a symmetric group $\mathfrak{S}_p$.

We are interested in developing a **single computationally efficient** machine learning (ML) model $f(\cdot) : (\mathbb{R} \cup \{\emptyset\})^p \mapsto \mathbb{R}$ (where $\emptyset$ denote as missing values) to predict the outcome where the features "arrive" sequentially (possibly in unknowable order), and we want the model to perform "well" across the entire setting. Consider a permutation $\sigma \in \mathfrak{S}_p$ that the order of the sequence is $\sigma(1), \cdots, \sigma(p)$ and we have the prediction across $f\left(\left\{\mathrm{X}_{\sigma(1)}\right\}\right), \cdots, f\left(\left\{\mathrm{X}_{\sigma(1)}, \mathrm{X}_{\sigma(2)}, \cdots, \mathrm{X}_{\sigma(p)}\right\}\right)$. To be more specific, we define "performing well" based on the following two criteria (i.e. the object of the project):

- **Criterion in statistical perspective:** For a function class $\mathcal{F}$ (the collection of all **algebraic structures** of a given type, for example, the collection of all linear regression models, SVM, etc.), a "performing well" model f should have a relatively low average error across all possible permutations $\sigma \in \mathfrak{S}_p$. Therefore, this object can be expressed as:

$$\min_{f \in \mathcal{F}} \mathbb{E}_{\sigma \in \mathfrak{S}_p, i \in \{1, \cdots, p\}} l\left(\boldsymbol{y}, f\left(\left\{\mathrm{X}_{\sigma(1)}, \mathrm{X}_{\sigma(2)}, \cdots, \mathrm{X}_{\sigma(i)}\right\}\right)\right) \tag{1}$$

  , where l is the loss function.

- **Criterion in robustness:** We aspire for our model to demonstrate robustness, specifically, we hope the model f can yield smaller errors in predictions even for the worst-case permutations across all permutations:

$$\min_{f \in \mathcal{F}} \max_{\sigma \in \mathfrak{S}_p} \sum_{i=1}^{p} l\left(\boldsymbol{y}, f\left(\left\{\mathrm{X}_{\sigma(1)}, \mathrm{X}_{\sigma(2)}, \cdots, \mathrm{X}_{\sigma(i)}\right\}\right)\right) \tag{2}$$

In our research, we primarily focus on the **criterion of robustness**. A robust model performs reliably across different permutations of feature arrival, minimizing risks associated with erroneous predictions. This consistency is especially crucial in healthcare, where patient outcomes depend on accurate and reliable diagnostic

[1] All information and definitions of the notations are displayed in Table 5.

and prognostic assessments, despite the inherently unpredictable nature of medical data and its collection process. This robust optimization problem is more formally defined in Section 4.1. Furthermore, in Section 5, we demonstrate through numerical experiments that the algorithms proposed in this paper meet both criteria and effectively address the problem.

1.2 Contribution of our work

In our research, we mainly make the following contributions:

- To the best of our knowledge, this work is the first to explore the Feature-incremental ML problem in tabular data, thoroughly defining the issue, its importance, and the criteria for addressing it.
- Section 3 presents the oracle method, explains its effectiveness in tackling the problem, and analyzes its computational complexity, demonstrating its impracticality for large datasets due to inefficiency.
- In Section 4, we thoroughly define the Robust Optimization (RO) problem within Feature-incremental Machine Learning (ML) and present a detailed algorithm for addressing it through alternating optimization. This section also includes proof of effectiveness for each part of the algorithm and complexity analyses.
- In Section 5, extensive numerical experiments across various datasets highlight the strengths and areas for improvement of the alternating optimization algorithm.

2 Literature Review

The core issue in feature-incremental machine learning primarily revolves around handling missing values. Therefore, we begin by reviewing developments and methods in missing value imputations.

For decades, handling missing data has been a key research topic in both statistical inference and machine learning[Little and Rubin, 2019]. The most common methods for dealing with missing features involve **imputation** [Hastie et al., 2009], where the missing value of an important feature for a specific instance is estimated from the available data. Imputation methods mainly include (*Predictive*) *Value Imputation* (PVI), predominantly used in statistics, and *Distribution-based Imputation* (DBI), which is more popular in machine learning contexts.

However, the effectiveness of most imputation methods depends on the assumption of *Missing Completely at Random* (MCAR)[Hastie et al., 2009], where the occurrence of missing values is independent of the data itself. While the effectiveness of imputation methods under the MCAR assumption has been extensively studied, these methods may be unreliable when the data is *Missing Not at Random* (MNAR) [Little, 1988, Jaeger, 2006], highlighting a significant challenge in dealing with missing data scenarios that do not conform to the MCAR assumption. The Missing Indicator Method (MIM) is a common technique used in conjunction with estimation to address missing data issues, with its effectiveness supported by both empirical and theoretical evidence. A recent paper [Van Ness et al., 2023] introduced Selective MIM (SMIM), demonstrating superior performance in high-dimensional data compared to the general MIM approach, suggesting that SMIM could offer a more nuanced and effective solution for dealing with missing data in complex datasets.

Adaptive optimization has also made strides in addressing the problem of imputation in the MNAR scenario. A recent paper [Bertsimas et al., 2024] developed an adaptive linear regression model that extends the Missing Indicator Method [Van Ness et al., 2023] for the MNAR scenario. This approach was further expanded to nonlinear models, showing improvements in numerical tests. This advancement suggests a promising direction for enhancing imputation techniques and model accuracy in dealing with missing data, especially in scenarios where data is not missing at random (MNAR), presenting a more flexible framework for various model types.

3 Exact (Oracle) Approach

Consider a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$ with the corresponding features $S_{\boldsymbol{X}} = \{X_1, \ldots, X_p\}$. A straightforward approach is to construct a dictionary that maps **any combination** of features to a corresponding model. More specifically, let us assume we have a combination of features denoted by $\mathcal{S} \subseteq S_{\boldsymbol{X}}$, where $\mathcal{S}$ represents a subset of the total set of features $S_{\boldsymbol{X}}$. We construct a new dataset $\mathcal{D}_{\mathcal{S}} = \{\mathbf{X}_{\mathcal{S}} \in \mathbb{R}^{n\times|\mathcal{S}|}, \boldsymbol{y} \in \mathbb{R}^n\}$ by selecting the columns corresponding to the features in s from the original dataset. This new dataset $\mathcal{D}_{\mathcal{S}}$ is then used for training a model, which can be any model of choice, such as linear regression, logistic regression, SVM, etc.

Algorithm 1 Feature Subset Model Mapping

```
Input:
    - A specified function class $\mathcal{F}$ (e.g., linear regression, classification trees).
    - Dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$.
Output:
    - A dictionary $\mathcal{M}$ mapping each of the $2^p$ subsets of features $S_{\boldsymbol{X}} = \{X_1, \ldots, X_p\}$ to a model.

1: Construct the power set of features $P(S_{\boldsymbol{X}})$.
2: Initialize an empty dictionary $\mathcal{M}$.
3: for each subset $\mathcal{S} \in P(S_{\boldsymbol{X}})$ do
4:     Construct a new dataset $\mathcal{D}_{\mathcal{S}} = \{\mathbf{X}_{\mathcal{S}} \in \mathbb{R}^{n\times|\mathcal{S}|}, \boldsymbol{y} \in \mathbb{R}^n\}$
5:     Train the fitting model $f_{\mathcal{S}}$ based on the model type.
6:     Add the model $f_{\mathcal{S}}$ and the subset of features $\mathcal{S}$ to $\mathcal{M}$ ($\mathcal{M}[\mathcal{S}] = f_{\mathcal{S}}$).
7: end for
8: return $\mathcal{M}$
```

The oracle approach[2] represents a highly **intuitive** but **computationally intensive** solution algorithm to our problem. For a given function class $\mathcal{F}$ and an arbitrary permutation $\sigma \in \mathfrak{S}_p$, we have independent models from the oracle function dictionary that $f_{\{\mathrm{X}_{\sigma(1)}\}}, \cdots, f_{\{\mathrm{X}_{\sigma(1)},\cdots,\mathrm{X}_{\sigma(p)}\}}$. Given that all models are trained to be optimal within datasets formed by subsets of features, this approach provides a comprehensive mapping from feature combinations to their best-performing models, considering the full spectrum of possible feature subsets, which implies that the oracle method meets the criteria from a statistical perspective. Additionally, since models are trained independently for all combinations of features, a specific model corresponds to any possible order of feature arrival (permutation). This ensures the oracle method's adherence to robustness criteria, providing a solid foundation for its effectiveness across various scenarios.

3.1 Complexity Analysis of Algorithm 1

However, Algorithm 1 generates $2^{|S_{\boldsymbol{X}}|} = 2^p$ independent machine learning models, which is computationally infeasible. To more intuitively present and calculate the complexity of the algorithm, we assume the function class to be the most commonly used **linear regression** model as an example.

Lemma 3.1. *For a given dataset* $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$ *and assume the function class* $\mathcal{F} = \mathrm{linearRegression}()$. *The complexity of Algorithm 1 is* $\mathcal{O}\left((p^3 + (2n+3)p^2 + 2np)2^{p-3}\right)$

Proof. From lemma A.2, we know training a linear regression model on $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$ is $\mathcal{O}\left(p^2 n + p^3\right)$. The complexity of Algorithm 1 stems from generating a total of 2^p linear regression models,

[2]Code of this section implemented in FeatureML/IndependentModelLearner.py#L122

thus allowing us to express its complexity as follows:

$$\sum_{k=0}^{p}\binom{p}{k}\mathcal{O}\left(k^2n+k^3\right)=\mathcal{O}\left(\sum_{k=0}^{p}\binom{p}{k}\left(k^2n+k^3\right)\right) \tag{3}$$

In order to compute the result of complexity, we first find the formula for combinatorial expression $\sum_{k=0}^{p}\binom{p}{k}k^2$, we have:

$$\begin{aligned}\sum_{k=0}^{p}\binom{p}{k}k^2 &= \sum_{k=0}^{p}\binom{p}{k}k(k-1)+\sum_{k=0}^{p}\binom{p}{k}k = p(p-1)\sum_{k=0}^{p-2}\binom{p-2}{k}+p\sum_{k=0}^{p-1}\binom{p-1}{k}\\ &= p(p-1)2^{p-2}+p\cdot 2^{p-1}=\boxed{p(p+1)2^{p-2}}\end{aligned}$$

Therefore, we can compute as follows:

$$\begin{aligned}\text{Equation (3)} &= \mathcal{O}\left(n\sum_{k=0}^{p}\binom{p}{k}k^2+\sum_{k=0}^{p}\binom{p}{k}k^3\right)\\ &= \mathcal{O}\left(n\sum_{k=0}^{p}\binom{p}{k}k^2+\sum_{k=0}^{p}\binom{p}{k}k(k-1)(k-2)+3\sum_{k=0}^{p}\binom{p}{k}k^2-2\sum_{k=0}^{p}\binom{p}{k}k\right)\\ &= \mathcal{O}\left((n+3)p(p+1)2^{p-2}-2p\cdot 2^{p-1}+p(p-1)(p-2)2^{p-3}\right)\\ &= \mathcal{O}\left((p^3+(2n+3)p^2+2np)2^{p-3}\right)\end{aligned}$$

■

It's evident that Algorithm 1 has an exponential complexity increase with respect to p and entails a substantial computational complexity. We extend the complexity analysis to an arbitrary model type, assuming its complexity to be $\mathcal{O}\left(g(n,p)\right)$, where g is a function of n and p. We can then express the algorithm's complexity as:

$$\sum_{k=0}^{p}\binom{p}{k}\mathcal{O}\left(g(n,p)\right)=\mathcal{O}\left(\sum_{k=0}^{p}\binom{p}{k}g(n,p)\right) \tag{4}$$

Considering that almost all basic machine learning models have a training complexity of at least $\mathcal{O}(np)$ or higher (for example, gradient boosting is $\mathcal{O}(npn_{trees})$, Naive Bayes is $O(np)$, and Decision Trees are $\mathcal{O}(n^2p)$ [Hastie et al., 2009]), we can deduce a **lower bound** for the complexity of Algorithm 1 as $\boxed{\mathcal{O}(np^2\cdot 2^{p-1})}$. This indicates that the oracle approach becomes impractical for larger datasets due to its substantial computational demands.

4 Alternating Optimization Algorithm

In this section, the focus is on robust optimization in the face of unknown sequences. A detailed formulation of the robust optimization problem in feature-incremental machine learning, along with the objectives, is presented in Section 4.1. The details and explanation of the alternating optimization algorithm in solving the robust optimization problem are given in Sections 4.2 - 4.4. Section 4.2 also introduces specific optimization methods for known sequence cases and expands to any subsets of the set of all permutations.

4.1 Robust Optimization Formulation

For our robust criterion in Section 1.1, we seek to develop a machine learning model $f(\cdot)$ that is robust to various permutations of feature arrival. Since we aim to minimize the worst-case prediction error, we utilize the *Wald minimax* model [Wald, 1945] to formulate this problem. Formally, we define the robust optimization (RO) problem as:

$$\min_{f\in\mathcal{F}}\max_{\sigma\in\mathfrak{S}_p} L(f,\sigma) \tag{5}$$

where $\mathfrak{S}_p$ represents all possible feature permutations and $\mathcal{F}$ is the function class. For efficiency, we no longer treat f as a dictionary of models but as a **single** model, i.e., f is not composed of 2^p independent models anymore. Instead, f is now a function mapping $(\mathbb{R}\cup\{\emptyset\})^p \mapsto \mathbb{R}$, where $\emptyset$ denotes missing value. $L(f,\sigma)$ is the loss function of model f under the permutation σ defined in definition 4.2.

Definition 4.1 (Partially Observed (PO) Dataset). Given a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$ and a feature permutation $\sigma \in \mathfrak{S}_p$. We define the partially observed dataset $\mathbf{X}_{\sigma,i}$ as

$$\mathbf{X}_{\sigma,i} = \left[\mathrm{X}_{\sigma(1)},\cdots,\mathrm{X}_{\sigma(i)},\underbrace{*,\cdots,*}_{p-i}\right] \tag{6}$$

where $\mathbf{X}_{\sigma(j)}$ is the j-th column in dataset $\mathbf{X}$ and $*$ defined as the imputed column **or** the column with missing value $\emptyset$. Given that the imputed values are real numbers, the definition of a partially observed dataset meets the model's requirement for inputs, ensuring compatibility between the model's expectations and the dataset's structure.

It's important to note that $\mathbf{X}_{\sigma,i}$ and $\mathrm{X}_{\sigma(i)}$ have different meanings. $\mathbf{X}_{\sigma,i}$ refers to a dataset, while $\mathrm{X}_{\sigma(i)}$ indicates a column within that dataset. For a clearer understanding, consider the following example:

Example 1 (partially observed dataset example). *Consider a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{2\times 3}, \boldsymbol{y} \in \mathbb{R}\}$ and we assume the feature order (i.e. permutation) is $\sigma = (2,1,3)$, we have the partially observed dataset as follows:*

$$\begin{aligned}
\mathbf{X}_{\sigma,1} &= \left[*, X_{\sigma(1)=2}, *\right] = [*, x_{12}, *]\\
\mathbf{X}_{\sigma,2} &= \left[X_{\sigma(2)=1}, X_{\sigma(1)=2}, *\right] = [x_{11}, x_{12}, *]\\
\mathbf{X}_{\sigma,3} &= \left[X_{\sigma(2)=1}, X_{\sigma(1)=2}, X_{\sigma(3)=3}\right] = [x_{11}, x_{12}, x_{13}]
\end{aligned}$$

Now, we can define the loss function in the context of feature-incremental machine learning problems.

Definition 4.2 (Feature-sequential loss function). Given a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$, a feature permutation $\sigma \in \mathfrak{S}_p$, and a machine learning model $f : (\mathbb{R}\cup\{\emptyset\})^p \mapsto \mathbb{R}$. We define the feature-sequential loss function as:

$$L(f,\sigma) = \frac{1}{p}\sum_{i=1}^{p} l\left(\boldsymbol{y}, f(\mathbf{X}_{\sigma,i})\right) \tag{7}$$

where $\mathbf{X}_{\sigma,i}$ is the partially observed dataset in definition 4.1 and l is the standard loss function used in statistics and machine learning (for example, mean squared error (MSE), log loss, mean absolute percentage error (MAPE), etc.).

For instance, in regression models, the Mean Squared Error (MSE) is commonly used as the standard loss function. Thus, in our robust optimization problem, the feature-sequential loss function is expressed as follows:

$$L(f,\sigma) = \frac{1}{p}\sum_{i=1}^{p} l\left(\boldsymbol{y}, f(\mathbf{X}_{\sigma,i})\right) = \frac{1}{np}\sum_{i=1}^{p}\sum_{j=1}^{n}\left(\boldsymbol{y}^{(j)} - \boldsymbol{x}_{\sigma,i}^{(j)}\right)^2$$

here $\boldsymbol{x}_{\sigma,i}^{(j)}$ represents the j-th instance (data point, row) in $\mathbf{X}_{\sigma,i}$.

Example 2 (**detailed example refers to definition 4.2**). *Consider a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{2\times 3}, \boldsymbol{y} \in \mathbb{R}^2\}$ and we assume the feature order (i.e. permutation) is $\sigma = (2, 1, 3)$, we have the feature-sequential loss function as follows:*

$$L(f, \sigma) = \frac{1}{2\times 3}\sum_{i=1}^{2}\left[\left(y_i, f\left([*, x_{1i}, *]\right)\right)^2 + \left(y_i - f\left([x_{1i}, x_{2i}, *]\right)\right)^2 + \left(y_i - f\left([x_{1i}, x_{2i}, x_{3i}]\right)\right)^2\right]$$

where the standard error metric is MSE.

In this context, the search space is the function class $\mathcal{F}$ and the set of all permutations. However, since $|\mathfrak{S}_p| = p!$ and there are too many possibilities, brute-force searching for the worst-case permutation and solving this minimax problem is **numerically difficult** or **not feasible**. Moreover, for each solution to the outer minimization problem (in other words, the ML model obtained from the minimization problem), it is also necessary to identify the corresponding worst-case permutation. This requirement significantly increases the computational complexity of the process. Therefore, we adopt an *alternating optimization* strategy to address this issue, as it can to some extent shorten the optimization time [Bezdek and Hathaway, 2002], circumvent local optima [Bezdek and Hathaway, 2003], and find a relatively optimized result within a reasonable timeframe.

The alternating optimization algorithm involves two main steps:

- **Maximization Step:** Searching for the feature permutation that leads to the highest prediction error under the current model.
- **Minimization Step:** Refining the model to perform better on this "worst-case" permutation, effectively improving the model's overall robustness.

4.2 Minimization step (Optimization with known sequence)

In this section[3], we consider a fixed permutation σ and we try to find the model f that minimizes $L(f, \sigma)$. We primarily divide the minimization step into two types, depending on the type of model f: linear models and nonlinear models. In section 4.2.1, we provide specific optimization algorithms and proof for when model f is either linear regression or logistic regression. In Section 4.2.2, the Gradient Boosted Tree algorithm was chosen for optimization in nonlinear models, highlighting the superior performance of Boosted Tree models like LightGBM and XGBoost in processing tabular data compared to neural networks (NNs). This superiority is detailed and further discussed in papers [Gorishniy et al., 2021, Grinsztajn et al., 2022, Shwartz-Ziv and Armon, 2022]. Within our GBDT algorithmic framework, LightGBM [Ke et al., 2017] serves as the foundational model, leveraging its state-of-the-art capabilities to handle the complexities of tabular datasets efficiently. In the end, we will discuss our imputation strategy for dealing with missing values in datasets in Section 4.2.3.

4.2.1 Linear case: regression

We focus on two prevalent types of linear models: linear regression and logistic regression, as the foundation for a wide array of predictive modeling tasks across numerous domains.

Both linear and regression models feature a structure that embodies a linear combination of attributes and weights. The linear regression model can be expressed as $\hat{\boldsymbol{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ and logistic regression has one additional ingredient as $\hat{\boldsymbol{y}} = \phi\left(\mathbf{X}\hat{\boldsymbol{\beta}}\right)$, where ϕ is denoted as the sigmoid function. We use **log loss** as the standard loss function l for logistic regression, which is defined as follows:

$$E(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\left\{y_i \log\left(\phi_i\right) + \left(1 - y_i\right)\log\left(1 - \phi_i\right)\right\} \tag{8}$$

[3]The code for minimization methods are implemented in FeatureML/AlternatingOptimization.py#L415

where $\phi_i \equiv s\left(\boldsymbol{\beta}^\top \boldsymbol{x}_i\right)$.

Thus, the minimization problem can be reformulated as an optimization problem concerning the parameter $\boldsymbol{\beta}$.

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\min_{\boldsymbol{\beta}} \mathrm{L}(\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\beta}} \frac{1}{p}\sum_{i=1}^{p} l\left(\boldsymbol{y}, f\left(\mathbf{X}_{\sigma,i}\right)\right) \text{ (L is the sequential loss function)} \\
&= \begin{cases} \arg\min_{\boldsymbol{\beta}} \frac{1}{np}\sum_{i=1}^{p}\sum_{j=1}^{n}\left(\boldsymbol{y}^{(j)} - \boldsymbol{x}_{\sigma,i}^{(j)}\boldsymbol{\beta}\right)^2 & \text{(Linear)} \\ \arg\max_{\boldsymbol{\beta}} \frac{-1}{np}\sum_{i=1}^{p}\sum_{j=1}^{n}\left[\boldsymbol{y}^{(j)}\log\left(\phi\left(\boldsymbol{x}_{\sigma,i}^{(j)}\boldsymbol{\beta}\right)\right) + (1-\boldsymbol{y}^{(j)})\log\left(1-\phi\left(\boldsymbol{x}_{\sigma,i}^{(j)}\boldsymbol{\beta}\right)\right)\right] & \text{(Logistic)} \end{cases}
\end{aligned}$$

Evidently, obtaining an analytical formula for the optimal parameters is highly challenging. In order to solve this minimization problem, we can **reformulate** the problem by introducing new concepts.

Definition 4.3 (expanded dataset). Given a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$ and a permutation σ. We define the expanded dataset $\mathcal{D}' = \{\mathbf{X}' \in \mathbb{R}^{np\times p}, \boldsymbol{y}' \in \mathbb{R}^{np}\}$. Here $\mathbf{X}' = \left[\mathbf{X}_{\sigma,1}^\top, \cdots, \mathbf{X}_{\sigma,p}^\top\right]^\top$ and $\boldsymbol{y}' = \left[\boldsymbol{y}^\top, \cdots, \boldsymbol{y}^\top\right]^\top$.

Example 3. *The notation previously introduced will now be demonstrated through the same dataset from example 2. In this example, we have the expanded dataset*

$$\mathbf{X}' = \left[\mathbf{X}_{\sigma,1}^\top, \cdots, \mathbf{X}_{\sigma,p}^\top\right]^\top = \begin{bmatrix} * & x_{12} & * \\ * & x_{22} & * \\ x_{11} & x_{12} & * \\ x_{21} & x_{22} & * \\ x_{11} & x_{12} & a_{13} \\ x_{21} & x_{22} & a_{23} \end{bmatrix} \quad \boldsymbol{y}' = \left[\boldsymbol{y}^\top, \cdots, \boldsymbol{y}^\top\right]^\top = \begin{bmatrix} y_1 \\ y_2 \\ y_1 \\ y_2 \\ y_1 \\ y_2 \end{bmatrix}$$

Lemma 4.1. *Given a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$, a function class $\mathcal{F}$, and the standard loss function l. We have $l\left(\mathbf{X}'\right) = p \cdot L(\mathbf{X})$, where L is the sequential loss function endowed with standard loss function l*

Proof.

$$\begin{aligned}
l\left(\mathbf{X}'\right) &= l\left(\left[\mathbf{X}_{\sigma,1}^\top, \cdots, \mathbf{X}_{\sigma,p}^\top\right]^\top\right) \\
&= \frac{1}{p} l\left(\mathbf{X}_{\sigma,1}\right) + \cdots + l\left(\mathbf{X}_{\sigma,p}\right) \\
&= p \cdot \frac{1}{p}\sum_{i}^{p} \mathbf{X}_{\sigma,i} = \boxed{p \cdot L(\mathbf{X})}
\end{aligned}$$

■

Lemma 4.1 presents a seemingly straightforward result; however, it plays a critical role in facilitating the resolution of the optimization challenge. Base on the conclusion of Lemma 4.1, we have:

$$\text{Equation 4.2.1} = \begin{cases} \arg\min_{\boldsymbol{\beta}} \frac{1}{p} \cdot l\left(\boldsymbol{y}, \mathbf{X}'\boldsymbol{\beta}\right) & \text{(Linear)} \\ \arg\min_{\boldsymbol{\beta}} \frac{1}{p} \cdot l\left(\boldsymbol{y}, \phi\left(\mathbf{X}'\boldsymbol{\beta}\right)\right) & \text{(Logistic)} \end{cases} \tag{9}$$

This implies that we can directly use the expanded dataset $\mathbf{X}'$ as input to basic machine learning models (such as SVM, logistic regression, decision trees, etc.) for training, effectively addressing issues introduced by the sequential loss function. Moreover, this approach is universally applicable to any machine learning model, meaning it remains effective for **nonlinear** models as well. This universality broadens the scope of our framework, allowing it to accommodate a wide range of models and problem settings.

4.2.2 Nonlinear case: Gradient Boosting Trees

Gradient Boosting Decision Tree (GBDT) is an iterative decision tree algorithm, consisting of multiple decision trees whose conclusions are summed up to make the final answer. We choose GBDT as the optimization algorithm because it can flexibly handle various types of data, including continuous and discrete values. More importantly, it can use some **robust loss functions**, demonstrating strong robustness to outliers [Einziger et al., 2019], which aligns perfectly with our requirements for the minimization step method.

We will not delve into the principles and algorithmic details of GBDT, considering that it is not the focus of our paper. For an in-depth discussion, we refer readers to the comprehensive review provided in the cited survey article [Natekin and Knoll, 2013]. We utilized the LightGBM [Ke et al., 2017] package to perform GBDT computations, primarily considering the following two points: (i) it can directly handle missing values, and (ii) the interface allows for the easy implementation of custom loss functions.

We train regression models using *lightgbm.train()* and classification models with *lightgbm.LGBMClassifier()*. MSE and log loss serve as the standard loss functions within the feature-sequential loss function for regression and classification models, respectively, and are integrated into LightGBM's training process. Given LightGBM's requirement for the gradient of custom loss functions, we illustrate this process using a regression model as an example. For iteration m in GBDT, we have the gradient (details in Algorithm 6) of custom loss function as:

$$\begin{aligned} g_m^j &= -\frac{\partial \operatorname{Loss}\left(y_j, f_{m-1}(\mathbf{X}_{\sigma,i})\right)}{\partial f_{m-1}(\mathbf{X}_{\sigma,i})} \\ &= 2\,\boldsymbol{y}^{(j)} - \frac{2}{p}\sum_{i}^{p} f_{m-1}\left(\boldsymbol{x}_{\sigma,i}^{(j)}\right) \end{aligned}$$

4.2.3 Imputation method

When constructing an expanded dataset for model training, we encounter situations of missing values due to unknown features. LightGBM [Ke et al., 2017] can directly train with missing data, however, this is not applicable for linear and logistic regression models. On the other hand, performing imputation on missing data can also improve the accuracy of model predictions and reduce errors. We have designed the following four imputation strategies[4]:

- **Imputation with nan**: replace all the missing values with `np.nan`. However, this strategy is only available for the `lightgbm` model.
- **Imputation with zeros**: In this approach, all missing values are replaced with 0. This method assumes that the absence of a value can be reasonably represented by 0.
- **Imputation with mean values**: This strategy involves calculating the mean value for each feature across the training data and using these means to replace missing values. It's based on the assumption that missing values can be approximated by the average observed value, making the dataset more complete for models that cannot handle missing values directly. This method is effective in maintaining the general distribution of the data but may not always be appropriate for data with a non-normal distribution or when missing values have patterns that mean replacement cannot adequately capture.
- **ML-based imputation**

We specifically introduce the **ML-based imputation** method. The idea of this algorithm is based on the **correlation** between features in the dataset, for example, the correlation between age and blood pressure, blood sugar, and the correlation between height and weight. We predict the missing values through the known features and the detailed steps of the method are in Algorithm 2.

[4]Code link: FeatureML/AlternatingOptimization.py#L145

To improve computational efficiency, we first calculate the linear regression model between any two feature columns in the dataset and consolidate all the models into a dictionary $d : (X_i, X_j) \to \text{LinearRegression}(X_i, X_j)$, where $i, j \in \{1, \cdots, p\}$.

Algorithm 2 ML-based imputation

```
Input:
    - Dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$.
    - Feature-regression dictionary $d : (X_i, X_j) \to \text{LinearRegression}(X_i, X_j)$, where $i, j \in \{1, \cdots, p\}$.
    - Feature order $\sigma$.
Output:
    - Expanded dataset $\mathbf{X}'_{\text{imputed}}$ with imputed values.

1: Initialize the expanded dataset $\mathbf{X}'_{\text{imputed}} = \left[\mathbf{X}_{\sigma,1}^\top, \cdots, \mathbf{X}_{\sigma,p}^\top\right]^\top$.
2: for $\mathbf{X}_{\sigma,i} \in \mathbf{X}'_{\text{imputed}}$ do
3:     for each missing feature $X_{\text{missing}}$ in $\mathbf{X}_{\sigma,i}$ do
4:         Find the feature with the highest correlation with $X_{\text{missing}}$ from known features
5:         Get the linear regression model $f = d\left[(X_{\text{missing}}, X_{\text{known}})\right]$
6:         Replace the missing column with predict values from $f(\mathbf{X}_{\text{known}})$
7:     end for
8: end for
9: return Expanded dataset $\mathbf{X}'_{\text{imputed}}$
```

By systematically utilizing the correlation between features, ML-based imputation ensures that the imputed values are more reflective of the underlying data distribution, thus maintaining the integrity of the dataset. We also observed that datasets subjected to ML-based imputation exhibit improved model performance across various metrics when compared to those utilizing more simplistic imputation methods (See the comparison in table 9), which provides empirical evidence to our assumption.

We also believe that the time complexity of the ML-based imputation algorithm is acceptable.

Lemma 4.2 (Complexity of computing feature-regression dictionary d). *Consider a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$. The complexity of constructing the feature-regression dictionary d is $\mathcal{O}\left(np^4 + p^5\right)$.*

Proof. To construct the Feature-regression dictionary d, since the number of combinations of any two features is $\binom{p}{2} = \frac{p(p-1)}{2}$, the overall complexity to compute d is

$$\frac{p(p-1)}{2}\mathcal{O}(p^2 n + p^3) = \mathcal{O}\left(p^4 n + p^5\right)$$

■

Lemma 4.3 (Complexity of ML-based imputation). *Consider a dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$, the corresponding feature-regression dictionary d, and a permutation σ. The complexity of constructing the expanded dataset $\mathbf{X}'_{imputed}$ with ML-based imputation method is $\mathcal{O}\left(np^3\right)$.*

Proof. The complexity of Algorithm 2 primarily focuses on predicting the missing columns. Assuming the dataset has p features, the number of columns that need to be predicted is $\sum_{I=1}^{p-1} = \frac{(p-1)(p-2)}{2}$ ($p-1$ missing columns in $\mathbf{X}_{\sigma,1}$, etc.). Given that the prediction complexity of a linear regression model (for a single data point) is $\mathcal{O}(p)$, the complexity of Algorithm 2 is $\mathcal{O}(np^3)$. ■

For a dataset, the feature-regression dictionary d is invariant, hence it needs to be computed only once during the entire alternating algorithm. However, for each iteration of the alternating algorithm, due to different worst-case permutations, a new prediction is required. Overall, since the complexity of Algorithm 2 remains within cubic terms, the efficiency is considered acceptable.

4.3 Maximization step

For fixed model f, find permutation σ that maximizes $L(f, \sigma)$. The problem we aim to solve is:

$$\max_{\sigma \in \mathfrak{S}_p} L(f, \sigma) \tag{10}$$

We expect this step identifies the model's weaknesses. However, the search space for permutations is vast, and this constitutes an extremely non-convex discrete model, making it exceedingly difficult to find an exact solution. In response to this, we propose two algorithms[5]. In Section 4.3.1, we describe a gradient-based local search algorithm. In Section 4.3.2, we introduce a stochastic algorithm developed from the concept of *simulated annealing*, designed to avoid the issue of encountering local maxima.

4.3.1 Gradient-based Approach

We leverage the concept of gradient descent to attempt to solve this maximization problem. However, considering that the set of all permutations is highly discrete, we initially utilize *edit distance* to define our gradient.

Definition 4.4 (Edit distance of feature permutations). Inspired by the Levenshtein Distance [Navarro, 2001], we define the edit distance of feature permutations as the minimum number of swaps required to transform one permutation σ_1 into another σ_2. Specifically, given two permutations σ_1 and σ_2 of features, the edit distance is k if and only if it is possible to convert σ_1 to σ_2 (or opposite) through a sequence of k swap operations between any two entries within the permutations.

Example 4. *Consider the following feature permutations:*

$$\sigma_1 = [X_1, X_2, X_3], \sigma_2 = [X_2, X_1, X_3], \sigma_3 = [X_3, X_1, X_2]$$

By definition 4.4 we know that

$$\text{edit\_distance}(\sigma_1, \sigma_2) = 1, \text{edit\_distance}(\sigma_1, \sigma_3) = 2$$

Definition 4.5 (Gradient of permutation). Consider a permutation σ_0, we define the gradient of permutation σ_0 as:

$$\nabla \sigma_0 = [\sigma \text{ such that } \text{edit\_distance}(\sigma, \sigma_0) = 1] \tag{11}$$

i.e., σ can be obtained from σ_0 by swapping any two entries.

Similar to gradient descent, once we obtain the gradient for the current permutation, we iterate through to find the first permutation that exhibits a higher error, thereby updating the current permutation. The detailed steps are described in Algorithm 3.

4.3.2 Featured Simulated annealing

However, after conducting extensive testing on Algorithm 3, we found that the results tend to converge and become stuck at local maxima. Considering our search space is extremely complex and non-convex, we aim to employ a stochastic algorithm to find the global optimum. We adopt the method described in [Bertsimas, 1993, Bertsimas and Nohadani, 2010, section 1], with the steps of the simulated annealing algorithm outlined as follows and the details are illustrated in Algorithm 4:

1. Start with an initial solution x_0 and set the initial temperature T_0 and final temperature T_f, along with the constant k_0.

[5]Code link: FeatureML/AlternatingOptimization.py#L325

Algorithm 3 Gradient-Based Maximization Algorithm

```
Input:
    - A model $f$
    - Dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n\}$.
    - Initial feature order $\sigma_0$
Output:
    - (local) feature order with maximum sequential loss.

1: Initialize $\sigma = \sigma_0$, error $= L(f, \sigma_0)$, $\nabla\sigma = \nabla\sigma_0$.
2: while Ture do
3:     Compute $\nabla\sigma$
4:     for $\sigma'$ in $\nabla\sigma$ do
         • If $L(f, \sigma') >$ error: error $= L(f, \sigma')$, $\sigma = \sigma'$, go back to step 3.
         • Else: go next element in $\nabla\sigma$.
5:     end for
6:     If we go through all elements in $\nabla\sigma$: Break          ▷ worst-case permutation find!
7: end while
8: return error, feature order $\sigma$
```

2. Generate a new solution x_k and calculate the change in the objective function $\Delta f = f(x_k) - f(x)$. If $\Delta f > 0$, then accept $x = x_k$; otherwise, accept $x = x_k$ with a probability $P(\Delta f, T)$, where $x = x_0$.
3. Reduce the temperature T_0.
4. Repeat the process until the temperature reaches T_f, or until the iteration limit k_{max}, to find the solution x.

To control the cooling schedule and temperature reduction, there are generally two methods used. When the change $\Delta f > 0$, which means a worse solution is obtained, these methods can help determine the probability of acceptance:

1. $T_{k+1} = T_k \times r$, where r is between $(0.95, 0.99)$ for multiplicative cooling, which is commonly used and relatively stable.
2. $T_{k+1} = T_k - \Delta T$, where ΔT is a constant for subtractive cooling.

4.4 Alternating step

The most straightforward idea is to alternately run the maximization and minimization steps until the error converges or the maximum number of iterations is reached. However, unlike typical optimization problems with continuous decision variables, the set of all feature permutations constitutes a completely discrete space. Moreover, due to the nature of our feature-sequential loss function, which depends on the feature order, the feature order corresponding to the maximum error may be **entirely different** for **different models**. This point has already been confirmed in our exhaustive search for small-scale datasets (see figure 1).

This leads to **two** issues. First, our goal is to obtain a **single** machine learning model that generalizes well across all feature orders. However, in each iteration, we only obtain a model that is optimized for the worst-case permutation derived from the previous iteration. This does not necessarily mean that the resulting model is also effective in minimizing errors across the majority of permutations. This leads to a second problem: it's difficult to define a standard for convergence to decide when to stop the algorithm's execution and, in each

Algorithm 4 Featured Simulated Annealing Algorithm

Input:
- A model f
- Dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n\}$.
- Initial feature order σ_0
- Initial temperature T_0, Stop temperature T_f, Cooling rate $r \in (0.95, 0.99)$, Maximum iteration $k_{\max}$

Output:
- (**approximate**) feature order with **maximum** sequential loss.

```
1: Initialize $T = T_0$, $k = 0$, $\sigma = \sigma_0$, error $= L(f, \sigma_0)$
2: while $k \leq k_{\max}$ and $T \geq T_f$ do
3:     Select two indices $i, j$ at random within the length of $\sigma$
4:     Swap the positions of $\sigma[i]$ and $\sigma[j]$ to create $\sigma_{\text{new}}$
5:     Calculate $\Delta_{\text{error}} = L(f, \sigma_{\text{new}}) - L(f, \sigma)$
6:     if $\Delta_{\text{error}} > 0$ or $\exp(\Delta_{\text{error}}/T) \geq \xi$ then        ▷ where $\xi = U(0, 1)$
7:         error $= L(f, \sigma_{\text{new}})$
8:         $\sigma = \sigma_{\text{new}}$
9:     end if
10:    $T = T \times r$
11:    $k = k + 1$
12: end while
13: return error, feature order $\sigma$
```

iteration, to identify the model with the best performance. We cannot predict the worst-case permutation in future iterations, nor the error performance of the existing model for other permutations (since we obtain a model and its corresponding worst-case permutation in each iteration). This makes it challenging to define and predict the convergence behavior of the algorithm. In order to solve the problem, we design the following two strategies.

4.4.1 Enhanced expanded dataset

To address the first issue, in other words, to enhance the generalization ability of the model, we aim for the newly obtained model in each iteration to learn from the worst-case permutations identified in previous iterations. Our solution is very straightforward. For each iteration, we generate an expanded dataset from all permutations obtained in previous iterations and concatenate these datasets to form a comprehensive expanded dataset for training. This approach significantly enhances the robustness (empirical evidence in table 9) of our model by integrating information from different permutations, in which the model gains a broader understanding of the data's structure.

4.4.2 Cross-computation

To ensure the selected model exhibits good robustness, an effective approach is to employ cross-evaluation, which involves examining the performance of models trained across all iterations on all permutations tested. Ultimately, the model demonstrating the smallest error is chosen. More specifically, in each iteration, we obtain a worst-case permutation along with a model. After the execution of the algorithm, for each model, we test its performance on the worst-case permutations from all iterations and select the model that exhibits the lowest error. This approach is inherently based on the simulation. A detailed description of the optimization process is provided in Algorithm 5.

Algorithm 5 Alternating Optimization Algorithm

```
Input:                                        ▷ Detailed parameters setting in table 6
    - Dataset $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$.
    - Max number of iterations $k_{\max}$
Output:
    - Robust model $f_{\textbf{best}}$

1: Initialize $\sigma = \sigma_0$ (random selection), permutation record $\text{list}_\sigma$
2: Construct the expanded dataset $\mathbf{X}'_\sigma$
3: Initialize $f = \text{minimization\_method}(\mathbf{X}'_\sigma)$, model record $\text{list}_f$
4: while $k \le k_{\max}$ do
5:     $\sigma_{\text{worst}} = \text{maximization\_method}(f, \sigma)$
6:     $\text{list}_\sigma \leftarrow$ add $\sigma_{\text{worst}}$
7:     Construct the expanded dataset $\mathbf{X}'_{\sigma_{\text{worst}}}$ (Enhanced=True: $\mathbf{X}'_{\text{list}_\sigma}$)
8:     Compute $f_{\text{new}} = \text{minimization\_method}(\mathbf{X}'_{\sigma_{\text{worst}}})$ (Or $\text{minimization\_method}(\mathbf{X}'_{\text{list}_\sigma})$)
9:     Update $\sigma \leftarrow \sigma_{\text{worst}}$, $f \leftarrow f_{\text{new}}$, $\text{list}_f \leftarrow$ add $f_{\text{new}}$, $k \leftarrow k+1$
10: end while
11: Cross-computation to get the model $f_{\textbf{best}}$
12: return $f_{\textbf{best}}$
```

5 Numerical Experiment

We now present the result of a comprehensive numerical evaluation of the proposed Oracle approach (section 3) and the Alternating Optimization algorithm (section 4)[6]. All data and results were obtained in an environment with an Apple M2 Pro CPU @ 3.2 GHz, MacOS Ventura 13.3.1, running CPython 3.9.1. We selected three typical publicly available datasets from the **UCI Machine Learning** Repository. These datasets (information of datasets in table 7) cover basic associated tasks and possess varying dataset sizes and feature types, thereby demonstrating the universality of our algorithm. Meanwhile, these three datasets are focused on the Health and Medicine field, considering this is the primary application scenario for our algorithm.

In this section, we will benchmark our algorithm across various conditions, including different minimization and maximization methods, as well as imputation approaches for the dataset, to verify the robustness of its performance. Based on the size of the dataset (truncated, full) and the type of model used in the minimization step[7] (linear case: Linear Regression for regression tasks, Logistic Regression for classification tasks; nonlinear case: GBDT for regression tasks), we divided the comparison into four groups. The detailed data results are displayed in Tables 8 - 11.

Furthermore, we compared the improvements of the best-performing models within each imputation method relative to the naive approach, as well as the gaps between them and the theoretically optimal Oracle approach. Here, we employ a controlled variable approach to ensure that the comparative subjects share the same minimization method, training dataset, and imputation method for handling missing data, maintaining consistency across all comparison points.

The naive approach refers to directly training the model on the complete training dataset. The differences in imputation methods only pertain to how missing values are handled when calculating the feature-sequential error or accuracy score. Oracle refers to the methodology introduced in Section 3, which we have already demonstrated to be theoretically optimal in previous sections.

In Tables 8 - 11, we have highlighted the best-performing models within the same imputation method using a $\boxed{\text{green box}}$, and the optimal method for each dataset is shown in purple. Overall, the strategy of building an

[6]All the results of the numerical experiment can be reproduced by running the Jupyter notebooks in folder numerical_test/.
[7]All information of setting is in Table 6

enhanced expanded dataset for each iteration proves effective in minimizing both average and worst-case errors, thereby fulfilling our objectives for both statistical and robustness criteria. We also observed that when GBDT is the minimization method, utilizing MaxEnhanced (explanation in Section 5.2) as the custom loss function effectively reduces the error for worst-case permutations. However, its impact on optimizing the average error is not significantly noticeable. Additionally, using the simulated annealing algorithm in the maximization step outperforms the gradient-based approach, validating our hypothesis that gradient-based maximization methods tend to get trapped in local optima when dealing with datasets featuring a larger number of features. Furthermore, we observe that the performance of the oracle method is consistently the best across all comparisons, providing further empirical evidence in support of our theoretical proofs presented in Section 3.

5.1 Linear case results

For regression tasks, we utilized linear regression, measuring error through Mean Square Error (MSE), while for binary and multiclass classification tasks, model performance was evaluated using accuracy scores and R^2 for regression to capture both the accuracy and the proportion of variance explained by the model.

In analyzing the combinations of methods within the alternating algorithm, we focus on the impact of three main variations on the model results: (1) the imputation method, (2) the expanded dataset setting, and (3) the maximization method.

	Zero	**Mean**	**ML-based**
Breast Cancer Original	(10.59%, 8.05%)	(9.52%, 5.62%)	(3.37%, 2.17%)
	(-7.61%, -7.45%)	(-0.00%, -0.00%)	(-0.00%, -0.00%)
Parkinson Telemonitoring	(99.93%, 99.89%)	(99.83%, 99.6%)	(0.13%, 0.14%)
	(-0.38%, -0.57%)	(-0.23%, -0.27%)	(-0.00%, -0.02%)
Heart Disease	(2.04%, 6.0%)	(2.00%, 3.85%)	(4.08%, 5.77%)
	(-5.66%, -3.64%)	(-3.77%, -1.82%)	(-3.77%, -0.00%)
# Iteration = 50	Improvement from naive approach Gap with Oracle (Max error/Min accuracy, Average error/accuracy)		

Table 1: Comparison of model performance across different imputation methods for a truncated (small) dataset with **linear** minimization method. We selected the best-performing models from each imputation method and compared the enhancements and gaps with both the Naive approach and the Oracle approach. Further detailed results can be found in Table 8.

We conducted linear or logistic regression on the complete dataset and selected 5 features with the highest coefficients (absolute value) to construct a small dataset, thereby hoping to enhance the focus and interpretability of our models. When there are only 5 features, the total number of permutations $|S_X|$ is $5! = 120$. Thus, during the maximization step, it is feasible to employ a **brute-force** search to identify the deterministic worst-case permutation.

From the analysis in Tables 8-11, it appears that ML-based imputation outperforms other methods in reducing errors and enhancing model stability. This could be due to its ability to leverage the underlying relationships between features more effectively, providing more accurate estimates for missing values, especially in cases where data is incomplete or sparse. Additionally, for regression tasks, naive linear regression models trained directly on datasets with complete data showed significantly increased errors when zero and mean imputation were used for feature-sequence loss calculation (data marked in red color). This suggests models trained on complete datasets might not predict well on datasets altered by zero imputation, highlighting the robustness of models derived from the alternating algorithm against various maximization methods.

In Table 2, the alternating optimization algorithm demonstrates significant performance improvements over the naive approach, especially with zero and mean imputations. However, when using ML-based imputation,

	Zero	Mean	ML-based
Breast Cancer Original	(13.51%, 8.86%)	(13.58%, 6.74%)	(3.37%, 2.15%)
	(-8.70%, -9.47%)	(-0.00%, -0.00%)	(-0.00%, -0.00%)
Parkinson Telemonitoring	(99.92%, 99.73%)	(99.81%, 99.38%)	(0.98%, 0.66%)
	(-4.55%, -8.16%)	(-2.09%, -1.65%)	(-0.77%, -0.47%)
Heart Disease	(380.00%, 45.71%)	(4.26%, 3.85%)	(4.26%, 5.77%)
	(-5.88%, -8.93%)	(-3.92%, -3.57%)	(-3.92%, -1.79%)
# Iteration = 50	Improvement from naive approach Gap with Oracle (Max error/Min accuracy, Average error/accuracy)		

Table 2: Comparison of model performance across different imputation methods for a full dataset with **linear** minimization method. Further detailed results can be found in Table 9.

the improvement margin narrows, with error reduction between 2%-6%. This is likely because ML-based imputation closely approximates real data values, leaving less room for improvement. Nonetheless, the gap with the Oracle approach remains within a convincing 10%, notably under 5% for Mean and ML-based imputations, indicating the effectiveness and potential of the algorithm for enhancing model predictions.

5.2 GBDT results

In this section, GBDT is utilized for the minimization step, currently implemented only for regression tasks. Unlike the linear case, this approach includes an additional variable - nan imputation, given LightGBM's capability to directly handle missing values. Furthermore, when optimizing models using LightGBM, a custom loss function, **MaxEnhanced**, is defined. This approach calculates the loss for each record's worst-case permutation separately during each iteration, identifying the permutation with the worst loss. The gradient for this specific permutation is computed, and gradients for samples not corresponding to this permutation are set to zero. This process is utilized for gradient boosting in GBDT, focusing the model's learning on improving performance for the worst-case scenarios.

	Nan	Zero	Mean	ML-based
Breast Cancer Original	(19.48%, 13.25%)	(20.78%, 13.25%)	(1.09%, 1.08%)	(2.20%, 1.08%)
	(-1.08%, -1.05%)	(-0.00%, -1.05%)	(-0.0%, -1.05%)	(-0.00%, -0.00%)
Parkinson Telemonitoring	(43.47%, 35.18%)	(42.38%, 33.27%)	(13.68%, 17.24%)	(8.88%, 33.68%)
	(-0.74%, -2.36%)	(-4.53%, -2.74%)	(-1.76%, -2.91%)	(-0.50%, -1.50%)
Heart Disease	(16.67%, 24.49%)	(18.75%, 22.45%)	(14.58%, 20.00%)	(14.89%, 18.00%)
	(14.29%, 17.31%)	(16.33%, 15.38%)	(12.24%, 15.38%)	(10.20%, 13.46%)
# Iteration = 50	Improvement from naive approach Gap with Oracle (Max error/Min accuracy, Average error/accuracy)			

Table 3: Comparison of model performance across different imputation methods for a truncated (small) dataset with **GBDT** minimization method. Further detailed results can be found in Table 10.

Data from Table 11 demonstrates that MaxEnhanced, compared to other custom loss functions, plays a significant role in reducing Max error. Additionally, models trained through the naive approach, when dealing with either nan or zero imputation, yield almost identical average MSE and Worst MSE.

Upon comparing the results from Table 3 and Table 4 for the Parkinson telemonitoring dataset, we observed that the MSE for GBDT was about 20% higher. This discrepancy mainly stems from the dataset's labels being bifurcated into two classes, while LightGBM can only train on single-dimensional labels. Consequently, we selected just one of these classes for the training process, which inadvertently led to an increased error rate.

	Nan	Zero	Mean	ML-based
Breast Cancer Original	(22.67%, 15.85%)	(24.00%, 15.85%)	(14.81%, 4.40%)	(4.44%, 3.23%)
	(-2.13%, -1.04%)	(-1.06%, -1.04%)	(-2.13%, -1.04%)	(-0.00%, -0.00%)
Parkinson Telemonitoring	(40.14%, 23.27%)	(39.67%, 22.09%)	(7.74%, 4.21%)	(4.80%, 3.41%)
	N/A	N/A	N/A	N/A
Heart Disease	(25.33%, 34.00%)	(21.28%, 28.00%)	(21.28%, 32.00%)	(26.09%, 29.41%)
	(34.09%, 31.37%)	(29.55%, 25.49%)	(29.55%, 29.41%)	(31.82%, 29.41%)
# Iteration = 50	Improvement from naive approach Gap with Oracle (Max error/Min accuracy, Average error/accuracy)			

Table 4: Comparison of model performance across different imputation methods for the full dataset with **GBDT** minimization method. Further detailed results can be found in Table 11. N/A means the computation time of the oracle program exceeded 1200 minutes.

Overall, using GBDT as the minimization method shows improvements comparable to those observed in linear cases. Notably, we observed that for multi-class classification tasks (second row of the Heart Disease dataset in Table 3 and 4), the performance of the oracle method is not as strong as that of the alternating optimization algorithm. The specific reasons for this discrepancy are not yet clear, and will be explored in future research.

5.3 Convergence criterion

We conducted numerical experiments on Algorithm 5 and performed cross-evaluation, taking into account that different maximization, minimization, and imputation methods exhibit completely distinct behaviors. We have summarized the performance of the error in each round throughout the four types of iterative processes.

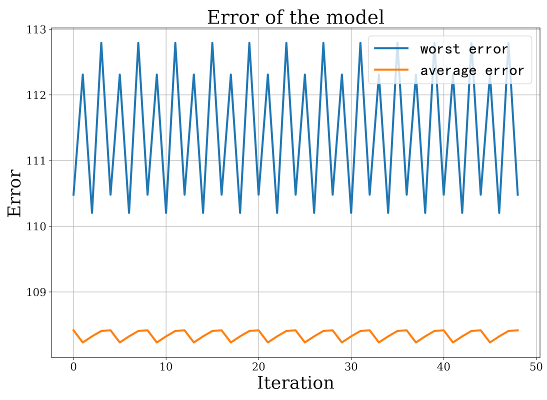

Figure 1: **Oscillation**
[Dataset: (Parkinson, small), Imputation: Mean, Minimization: (Gradient Boosted, NonEnhanced), Maximization: deterministic search]

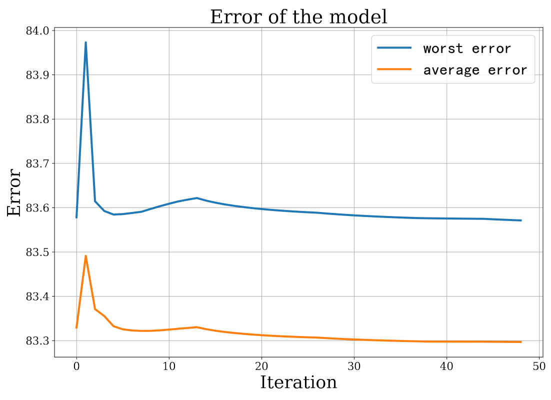

Figure 2: **Spike**
[Dataset: (Parkinson, full), Imputation: ML-based, Minimization: (Linear Regression, Enhanced), Maximization: simulated annealing]

- **Oscillation (figure 1)**: In this scenario, the model and the corresponding worst-case permutation generated from each iteration form a cycle over k rounds, leading to a pattern of regular oscillations behavior. However, this convergence behavior only occurs when the **dataset is sufficiently small** (for example, when the number of features p is less than 8), allowing for an exhaustive search during the maximization step, and the training dataset needs to be non-enhanced.

- **Spike (figure 2)**: In such scenarios, the error undergoes an initial sharp increase as iterations proceed, then slowly diminishes, appearing to converge. Yet, the final model achieved through this convergence may not outperform the initially trained model. This pattern can be misleading, potentially causing misinterpretation of the algorithm's convergence behavior.
- **Random (figure 3)**: The error performance is completely random and unpredictable, exhibiting a pattern of random fluctuations.
- **Convergence (figure 4)**: This scenario represents an ideal convergence, where the model's error decreases and converges with the increase in iterations.

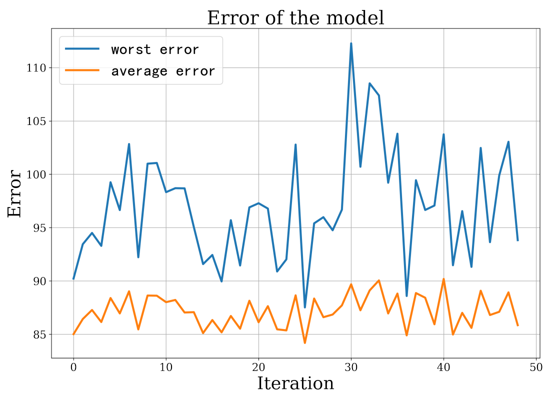

Figure 3: **Not Converge**
[Dataset: (Parkinson, full), Imputation: Mean, Minimization: (Gradient Boosted, NonEnhanced), Maximization: simulated annealing]

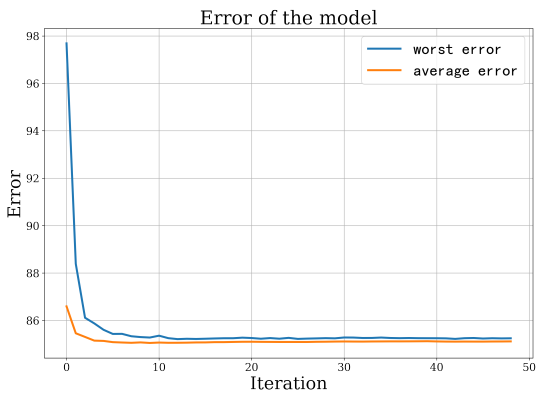

Figure 4: **Type 3 Convergence**
[Dataset: (Parkinson, full), Imputation: mean, Minimization: (Linear Regression, Enhanced), Maximization: gradient-based]

Our ideal behavior is convergence, and in our experimental results, most convergence occurs when using the Enhanced expanded dataset for training and gradient-based search for maximization (some convergence also occurs with simulated annealing). This is likely due to randomness in simulated annealing causing spikes. Interestingly, while gradient-based search tends to converge better than simulated annealing, it can get trapped in local maxima, so its final model errors may not always outperform those from simulated annealing (see Table 9 and 11).

6 Conclusion and Further Improvement

In medical diagnostics and customer preference prediction scenarios, dealing with missing values and features arriving sequentially and in unknown order is common. This paper presents a detailed definition of the Feature-incremental ML problem, introducing a versatile alternating optimization framework adaptable to various machine learning models to address this challenge effectively. We demonstrate numerically that our algorithm has a performance improvement of 10%-45% over naive training methods, significantly increasing the efficiency with certain imputation methods and narrowing the gap with the exact solution to within 10%.

The numerical test results underscore our framework's robustness and optimization capabilities across various task types (regression or classification) and machine learning models.

However, our research still has much ground to explore and discover. We lack a deep understanding of why the alternating algorithm works in Feature-incremental ML problems, such as scientifically defining convergence criteria and predicting behavior in later iterations. On the other hand, inspired by the Oracle approach, we consider developing multiple, joint robust optimization models for a dataset as a future research direction, aiming to deepen our methodology's understanding and expand its applicability.

Supplementary information

Proofs of key lemma and detailed numerical results are available in the appendix.

7 References

[Bertsimas, 1993] Bertsimas, D. (1993). Simulated annealing. *Statistical science*, 8(1):10–15.

[Bertsimas et al., 2024] Bertsimas, D., Delarue, A., and Pauphilet, J. (2024). Adaptive optimization for prediction with missing data. *arXiv preprint arXiv:2402.01543*.

[Bertsimas and Nohadani, 2010] Bertsimas, D. and Nohadani, O. (2010). Robust optimization with simulated annealing. *Journal of Global Optimization*, 48:323–334.

[Bezdek and Hathaway, 2002] Bezdek, J. C. and Hathaway, R. J. (2002). Some notes on alternating optimization. In Pal, N. R. and Sugeno, M., editors, *Advances in Soft Computing — AFSS 2002*, pages 288–300, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Bezdek and Hathaway, 2003] Bezdek, J. C. and Hathaway, R. J. (2003). Convergence of alternating optimization. *Neural, Parallel & Scientific Computations*, 11(4):351–368.

[Einziger et al., 2019] Einziger, G., Goldstein, M., Sa'ar, Y., and Segall, I. (2019). Verifying robustness of gradient boosted models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2446–2453.

[Gorishniy et al., 2021] Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. (2021). Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943.

[Grinsztajn et al., 2022] Grinsztajn, L., Oyallon, E., and Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer.

[Jaeger, 2006] Jaeger, M. (2006). On testing the missing at random assumption. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pages 671–678. Springer.

[Kaya et al., 2020] Kaya, S. K., Navarro-Arribas, G., and Torra, V. (2020). Dynamic features spaces and machine learning: Open problems and synthetic data sets. In Huynh, V.-N., Entani, T., Jeenanunta, C., Inuiguchi, M., and Yenradee, P., editors, *Integrated Uncertainty in Knowledge Modelling and Decision Making*, pages 125–136, Cham. Springer International Publishing.

[Ke et al., 2017] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Little, 1988] Little, R. J. (1988). A test of missing completely at random for multivariate data with missing values. *Journal of the American statistical Association*, 83(404):1198–1202.

[Little and Rubin, 2019] Little, R. J. and Rubin, D. B. (2019). *Statistical analysis with missing data*, volume 793. John Wiley & Sons.

[Natekin and Knoll, 2013] Natekin, A. and Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21.

[Navarro, 2001] Navarro, G. (2001). A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88.

[Ni et al., 2024] Ni, H., Gu, S., Fan, R., and Hou, C. (2024). Feature incremental learning with causality. *Pattern Recognition*, 146:110033.

[Shwartz-Ziv and Armon, 2022] Shwartz-Ziv, R. and Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90.

[Van Ness et al., 2023] Van Ness, M., Bosschieter, T. M., Halpin-Gregorio, R., and Udell, M. (2023). The missing indicator method: From low to high dimensions. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5004–5015.

[Wald, 1945] Wald, A. (1945). Statistical decision functions which minimize the maximum risk. *Annals of Mathematics*, 46(2):265–280.

A Appendix: Proofs and Math Formulas

Lemma A.1. *Let* $\mathbf{X} \in \mathbb{R}^{n \times p}$ *be a matrix of features with* n *observations and* p *predictors, and let* $\boldsymbol{y} \in \mathbb{R}^n$ *be a vector of responses. Assume that* $\mathbf{X}$ *has full column* rank, *i.e.,* $\operatorname{rank}(\mathbf{X}) = p$*. Then, the coefficient vector* $\boldsymbol{\beta}$ *that minimizes the residual sum of squares in the linear regression model* $\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ *is given by* $\boldsymbol{\beta} = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\boldsymbol{y}$.

Proof. To find the coefficient vector $\boldsymbol{\beta}$ that minimizes the residual sum of squares, we set up the loss function $L(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|^2$. To minimize this function, we take its gradient with respect to $\boldsymbol{\beta}$ and set it to zero:

$$\begin{aligned}\nabla_{\boldsymbol{\beta}} L(\boldsymbol{\beta}) &= \nabla_{\boldsymbol{\beta}}\left[(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^\top(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})\right] = \nabla_{\boldsymbol{\beta}}\left[\boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\boldsymbol{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}\right] \\ &= -2\mathbf{X}^\top\boldsymbol{y} + 2\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} = -2\mathbf{X}^\top(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}).\end{aligned}$$

Setting the gradient to zero gives us:

$$\mathbf{X}^\top(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{0}.$$

Expanding this, we get:

$$\mathbf{X}^\top\boldsymbol{y} - \mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} = \mathbf{0},$$

which implies that:

$$\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\top\boldsymbol{y}.$$

Since $\mathbf{X}$ has full column rank, $\mathbf{X}^\top\mathbf{X}$ is invertible. Multiplying both sides by $\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}$, we find:

$$\boldsymbol{\beta} = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\boldsymbol{y},$$

which is the formula for the ordinary least squares estimator for the coefficients $\boldsymbol{\beta}$ in the linear regression model. ■

Algorithm 6 Gradient Tree Boosting Algorithm

```
1: Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2: for $m = 1$ to $M$ do
3:     for $i = 1, 2, \dots, N$ do
4:         Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
5:     end for
6:     Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$.
7:     for $j = 1, 2, \dots, J_m$ do
8:         Compute $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.
9:     end for
10:    Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
11: end for
12: Output $\hat{f}(x) = f_M(x)$.
```

Lemma A.2. *Given a dataset* $\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n\}$*, the computational complexity of training a linear regression model using the ordinary least squares method is* $\mathcal{O}(p^2 n + p^3)$.

Proof. Training a linear regression model via the ordinary least squares (OLS) approach essentially involves performing the matrix computation $\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}$, as established in Lemma A.1. Therefore, the computation mainly involves the following parts:

1. The multiplication $\mathbf{X}^\top\mathbf{X}$ requires $\mathcal{O}(p^2n)$ operations since $\mathbf{X}$ is an $n \times p$ matrix, and we are performing a dot product for each of the p^2 entries of the resulting $p \times p$ matrix.
2. The matrix-vector multiplication $X^\top y$ requires a complexity of $\mathcal{O}(pn)$.
3. Computing the matrix inverse $\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}$ has a computational complexity of $\mathcal{O}(p^3)$, which is the complexity of Gaussian elimination or other standard methods for matrix inversion.

Therefore, the total complexity is the sum of the two operations, which is $\mathcal{O}(p^2n) + \mathcal{O}(pn) + \mathcal{O}(p^3) = \mathcal{O}(p^2n + p^3)$. ■

B Appendix: Tables and Charts

Notations	Description
$\mathcal{D} = \{\mathbf{X} \in \mathbb{R}^{n\times p}, \boldsymbol{y} \in \mathbb{R}^n\}$	Dataset
$\mathbf{X}$	Matrix of data points
$\boldsymbol{y}$	Labels
$\boldsymbol{X} := X_1, \ldots, X_p$	Denoting features (i.e. random variables/column names)
$S_{\boldsymbol{X}}$	Set of features
$\mathfrak{S}_p$	The group of the permutations of $\boldsymbol{X}$
X_i	The i-th column of $\mathbf{X}$
$\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,p}]$	The i-th row of $\mathbf{X}$
$\mathcal{F}$	Function class: The collection of all algebraic structures of a given type
$\mathbf{X}_{\sigma,i}$	Partially observed dataset, see Definition 4.1
$*$	Imputed column or the column with missing value $\emptyset$
$\mathcal{S}$	Subset of the set of features S_X
$\phi(\cdot)$	Sigmoid function
$\mathbf{X}'$	Expanded dataset, see Definition 4.3
σ	Permutation of features (order of features)

Table 5: Table of Notations

Imputation method	Nan imputation	Only available for GBDT
	Zero imputation	
	Mean imputation	
	Heuristic imputation	
Dataset size	small (truncated)	Select 5 columns from the original dataset
	Full	
Maximization Step	Brute-force	only if dataset size is small
	Gradient-based approach	
	Simulated annealing	
Minimization Step	Linear Regression	
	Logistic Regression	
	Gradient Boosting Trees	only available for classification tasks
Expanded dataset	Enhanced	
	NonEnhanced	
	MaxEnhanced	only if the minimization method is GBDT

Table 6: Summary of methodological variations and their applicability based on dataset characteristics and algorithmic steps

	Dataset Characteristics	**Associated Tasks**	**Feature Type**	**# Instances n**	**# Features p**
Breast Caner Wisconsin (Original)	Multivariate	Classification (Binary)	Integer	699	9
Parkinsons Telemonitoring	Tabular	Regression	Integer, Real	5875	15
Heart Disease	Multivariate	Classification (Multiclass)	Categorical, Integer, Real	303	13

Table 7: The basic information about the UCI datasets used in Numerical Experiments focuses on the Health and Medicine field.

	Breast Cancer Original (Binary Classification) $n = 699, p = 5$	**Parkinson Telemonitoring (Regression)** $n = 5875, p = 5$	**Heart Disease (Multiclass Classification)** $n = 303, p = 5$
(Zero, NonEnhanced)	(0.74, 0.81)	(87.58, 86.99)	(0.50, 0.51)
(Zero, Enhanced)	(0.85, 0.87)	(86.96, 86.87)	(0.50, 0.53)
(Mean, NonEnhanced)	(0.89, 0.93)	(87.16, 86.71)	(0.51, 0.54)
(Mean, Enhanced)	(0.92, 0.94)	(86.83, 86.61)	(0.51, 0.54)
(Heuristic, NonEnhanced)	(0.92, 0.94)	(86.63, 86.40)	(0.51, 0.54)
(Heuristic, Enhanced)	(0.92, 0.94)	(86.63, 86.40)	(0.51, 0.55)
Naive (Zero)	(0.76, 0.80)	(127895.58, 52652.09)	(0.49, 0.50)
Naive (Mean)	(0.84, 0.89)	(50542.08, 21834.96)	(0.50, 0.52)
Naive (Heuristic)	(0.89, 0.92)	(86.74, 86.52)	(0.49, 0.52)
Oracle	(0.92, 0.94)	(86.63, 86.38)	(0.53, 0.55)
# Iterations = 50	**(Minimum, Average) Accuracy Score**	**(Maximum, Average) MSE**	**(Minimum, Average) Accuracy Score**

Table 8: Linear case with the truncated (small) dataset, the variations on methods in the alternating algorithm: (Imputation method, Expanded dataset).

	Breast Cancer Original (Binary Classification) $n = 699, p = 9$	**Parkinson Telemonitoring (Regression)** $n = 5875, p = 19$	**Heart Disease (Multiclass Classification)** $n = 303, p = 13$
(Zero, NonEnhanced, Gradient)	(0.71, 0.80)	(92.65, 86.72)	(0.31, 0.48)
(Zero, NonEnhanced, Simulated)	(0.74, 0.83)	(92.76, 87.01)	(0.34, 0.47)
(Zero, Enhanced, Gradient)	(0.71, 0.80)	(87.05, 86.16)	(0.48, 0.51)
(Zero, Enhanced, Simulated)	(0.84, 0.86)	(86.66, 86.03)	(0.48, 0.51)
(Mean, NonEnhanced, Gradient)	(0.81, 0.93)	(90.41, 81.60)	(0.48, 0.51)
(Mean, NonEnhanced, Simulated)	(0.89, 0.94)	(89.09, 81.72)	(0.49, 0.54)
(Mean, Enhanced, Gradient)	(0.92, 0.95)	(84.62, 80.85)	(0.47, 0.51)
(Mean, Enhanced, Simulated)	(0.92, 0.95)	(84.64, 80.76)	(0.47, 0.51)
(Heuristic, NonEnhanced, Gradient)	(0.92, 0.95)	(83.53, 79.91)	(0.49, 0.54)
(Heuristic, NonEnhanced, Simulated)	(0.92, 0.95)	(83.76, 79.92)	(0.48, 0.54)
(Heuristic, Enhanced, Gradient)	(0.92, 0.95)	(83.53, 79.91)	(0.48, 0.55)
(Heuristic, Enhanced, Simulated)	(0.92, 0.95)	(83.67, 79.95)	(0.49, 0.55)
Naive (Zero)	(0.74, 0.79)	(114690.86, 32165.60)	(0.10, 0.35)
Naive (Mean)	(0.81, 0.89)	(44520.15, 13079.73)	(0.47, 0.52)
Naive (Heuristic)	(0.89, 0.93)	(84.36, 80.44)	(0.47, 0.52)
Oracle	(0.92, 0.95)	(82.89, 79.54)	(0.51, 0.56)
# Iterations = 50	**(Minimum, Average) Accuracy Score**	**(Maximum, Average) MSE**	**(Minimum, Average) Accuracy Score**

Table 9: Linear case with the full dataset, the variations on methods in the alternating algorithm: (Imputation method, Expanded dataset, maximization method).

	Breast Cancer Original (Binary Classification) $n = 699, p = 5$	**Parkinson Telemonitoring (Regression)** $n = 5875, p = 5$	**Heart Disease (Multiclass Classification)** $n = 303, p = 5$
(Nan, NonEnhanced)	(0.82, 0.90)	(112.36, 107.94)	(0.56, 0.59)
(Nan, Enhanced)	(0.92, 0.94)	(106.38, 104.98)	(0.56, 0.61)
(Nan, MaxEnhanced)	(0.92, 0.94)	(105.26, 104.76)	(0.56, 0.61)
(Zero, NonEnhanced)	(0.82, 0.89)	(142.60, 117.51)	(0.51, 0.55)
(Zero, Enhanced)	(0.93, 0.94)	(106.09, 105.14)	(0.57, 0.60)
(Zero, MaxEnhanced)	(0.93, 0.94)	(106.29, 105.58)	(0.51, 0.60)
(Mean, NonEnhanced)	(0.87, 0.92)	(110.92, 107.84)	(0.55, 0.59)
(Mean, Enhanced)	(0.93, 0.94)	(106.92, 106.10)	(0.55, 0.60)
(Mean, MaxEnhanced)	(0.93, 0.94)	(106.33, 105.32)	(0.55, 0.60)
(Heuristic, NonEnhanced)	(0.92, 0.94)	(117.18, 104.88)	(0.53, 0.56)
(Heuristic, Enhanced)	(0.93, 0.95)	(105.45, 104.25)	(0.54, 0.59)
(Heuristic, MaxEnhanced)	(0.93, 0.95)	(105.01, 103.88)	(0.54, 0.59)
Naive (Nan)	(0.77, 0.83)	(175.85, 136.53)	(0.48, 0.49)
Naive (Zero)	(0.77, 0.83)	(175.85, 136.53)	(0.48, 0.49)
Naive (Mean)	(0.92, 0.93)	(115.25, 109.95)	(0.48, 0.50)
Naive (Heuristic)	(0.91, 0.93)	(110.30, 107.55)	(0.47, 0.50)
Oracle	(0.93, 0.95)	(104.49, 102.34)	(0.49, 0.52)
# Iterations = 50	**(Minimum, Average) Accuracy Score**	**(Maximum, Average) MSE**	**(Minimum, Average) Accuracy Score**

Table 10: Gradient Boosted Tree with the truncated (small) dataset, the variations on methods in the alternating algorithm: (Imputation method, Expanded dataset).

	Breast Cancer Original (Binary Classification) $n = 699, p = 9$	**Parkinson Telemonitoring (Regression)** $n = 5875, p = 19$	**Heart Disease (Multiclass Classification)** $n = 303, p = 13$
(Nan, NonEnhanced, Gradient)	(0.80, 0.90)	(105.57, 97.88)	(0.56, 0.65)
(Nan, NonEnhanced, Simulated)	(0.82, 0.92)	(105.64, 94.28)	(0.56, 0.66)
(Nan, Enhanced, Gradient))	(0.91, 0.95)	(106.24, 100.87)	(0.54, 0.65)
(Nan, Enhanced, Simulated)	(0.91, 0.95)	(103.94, 96.54)	(0.58, 0.68)
(Nan, MaxEnhanced, Gradient)	(0.91, 0.95)	(100.374, 90.69)	(0.54, 0.65)
(Nan, MaxEnhanced, Simulated)	(0.92, 0.95)	(99.41, 88.50)	(0.59, 0.67)
(Zero, NonEnhanced, Gradient)	(0.83, 0.91)	(107.39, 88.27)	(0.53, 0.60)
(Zero, NonEnhanced, Simulated)	(0.82, 0.90)	(106.72, 81.33)	(0.55, 0.64)
(Zero, Enhanced, Gradient))	(0.89, 0.95)	(106.05, 94.65)	(0.57, 0.64)
(Zero, Enhanced, Simulated)	(0.93, 0.95)	(105.10, 94.32)	(0.55, 0.64)
(Zero, MaxEnhanced, Gradient)	(0.89, 0.95)	(107.14, 94.07)	(0.57, 0.64)
(Zero, MaxEnhanced, Simulated)	(0.92, 0.95)	(101.33, 91.10)	(0.55, 0.65)
(Mean, NonEnhanced, Gradient)	(0.73, 0.91)	(106.37, 93.65)	(0.55, 0.65)
(Mean, NonEnhanced, Simulated)	(0.84, 0.92)	(105.56, 94.21)	(0.54, 0.63)
(Mean, Enhanced, Gradient))	(0.91, 0.95)	(106.62, 104.29)	(0.55, 0.65)
(Mean, Enhanced, Simulated)	(0.92, 0.95)	(104.58, 95.04)	(0.55, 0.64)
(Mean, MaxEnhanced, Gradient)	(0.91, 0.95)	(100.11, 90.78)	(0.55, 0.65)
(Mean, MaxEnhanced, Simulated)	(0.93, 0.95)	(99.48, 91.00)	(0.57, 0.66)
(Heuristic, NonEnhanced, Gradient)	(0.93, 0.95)	(103.85, 92.10)	(0.55, 0.66)
(Heuristic, NonEnhanced, Simulated)	(0.92, 0.95)	(101.35, 90.65)	(0.58, 0.66)
(Heuristic, Enhanced, Gradient))	(0.94, 0.96)	(100.50, 71.76)	(0.55, 0.65)
(Heuristic, Enhanced, Simulated)	(0.93, 0.96)	(99.86, 72.62)	(0.56, 0.66)
(Heuristic, MaxEnhanced, Gradient)	(0.94, 0.96)	(98.48, 87.91)	(0.55, 0.68)
(Heuristic, MaxEnhanced, Simulated)	(0.93, 0.96)	(97.74, 89.27)	(0.57, 0.67)
Naive (Nan)	(0.75, 0.82)	(175.85, 136.53)	(0.47, 0.50)
Naive (Zero)	(0.75, 0.82)	(175.85, 136.53)	(0.47, 0.50)
Naive (Mean)	(0.81, 0.91)	(115.25, 109.95)	(0.47, 0.50)
Naive (Heuristic)	(0.90, 0.93)	(110.30, 107.55)	(0.46, 0.51)
Oracle	(0.94, 0.96)	Computation Time Exceeded†	(0.44, 0.51)
# Iterations = 30	**(Minimum, Average) Accuracy Score**	**(Maximum, Average) MSE**	**(Minimum, Average) Accuracy Score**

Table 11: Gradient Boosted Tree with the full dataset, the variations on methods in the alternating algorithm: (Imputation method, Expanded dataset, maximization method). †: The computation time of the program exceeded 1200 minutes.

Statement of Academic Integrity Regarding Plagiarism

I, the undersigned ....Cai Yubo.. [family name, given name(s)], hereby certify on my honor that:

1. The results presented in this report are the product of my own work.

2. I am the original creator of this report.

3. I have not used sources or results from third parties without clearly stating thus and referencing them according to the recommended rules for providing bibliographic information.

Declaration to be copied below:

I hereby declare that this work contains no plagiarized material.

Date ....March 18th, 2024...

Signature Yubo CAI