Certain and Approximately Certain Models for Statistical Learning

CHENG ZHEN, Oregon State University, USA
NISCHAL ARYAL, Oregon State University, USA
ARASH TERMEHCHY, Oregon State University, USA
AMANDEEP SINGH CHABADA, Oregon State University, USA

Real-world data is often incomplete and contains missing values. To train accurate models over real-world datasets, users need to spend a substantial amount of time and resources imputing and finding proper values for missing data items. In this paper, we demonstrate that it is possible to learn accurate models directly from data with missing values for certain training data and target models. We propose a unified approach for checking the necessity of data imputation to learn accurate models across various widely-used machine learning paradigms. We build efficient algorithms with theoretical guarantees to check this necessity and return accurate models in cases where imputation is unnecessary. Our extensive experiments indicate that our proposed algorithms significantly reduce the amount of time and effort needed for data imputation without imposing considerable computational overhead.

CCS Concepts: • **Information systems → Data cleaning**; • **Computing methodologies → Supervised learning**.

Additional Key Words and Phrases: Data Preparation, Data Quality, Uncertainty Quantification

ACM Reference Format:
Cheng Zhen, Nischal Aryal, Arash Termehchy, and Amandeep Singh Chabada. 2024. Certain and Approximately Certain Models for Statistical Learning. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 126 (June 2024), 25 pages. https://doi.org/10.1145/3654929

1 INTRODUCTION

The performance of a machine learning (ML) model relies substantially on the quality of its training data. Real-world training data often contain a considerable number of examples with missing values, i.e., *incomplete data*. One may train an ML model by ignoring the training examples with missing values. This approach, however, may significantly reduce the accuracy of the resulting model as it may lose out on some useful examples [30].

To address the problem of training over incomplete data, users usually replace each missing data item with a value, i.e., data imputation, and train their models over the resulting *repaired data*. To repair incomplete data, users must figure out the mechanisms and causes of data missingness, e.g., completely at random or based on observed values of some features [28]. Based on this mechanism, they build a (statistical) model for missing data and replace the missing values with

Authors' addresses: Cheng Zhen, zhenc@oregonstate.edu, Oregon State University, 2500 NW Monroe Ave, Corvallis, Oregon, USA, 97331; Nischal Aryal, aryaln@oregonstate.edu, Oregon State University, 2500 NW Monroe Ave, Corvallis, Oregon, USA, 97331; Arash Termehchy, termehca@oregonstate.edu, Oregon State University, 2500 NW Monroe Ave, Corvallis, Oregon, USA, 97331; Amandeep Singh Chabada, chabadaa@oregonstate.edu, Oregon State University, 2500 NW Monroe Ave, Corvallis, Oregon, USA, 97331.

ACM 2836-6573/2024/6-ART126
https://doi.org/10.1145/3654929

some measurements defined over this model, e.g., mean. Users may also leverage a variety of ML models to repair missing values, e.g., tree-based or linear regression [21]. Researchers have shown that the desired imputation method may vary depending on the downstream ML task [20]. Hence, it is often challenging to find a model of data missingness that results in an accurate ML model for a downstream task [20]. The aforementioned steps of finding a missingness mechanism, constructing an accurate missingness model, and finding the right statistical measurement(s) for imputation usually require a significant amount of time and manual effort. Surveys indicate that most users spend about 80% of their time on data preparation and repair [18, 24].

Researchers have recently shown that one may learn accurate Datalog rules [25] and K-nearest neighbor classifier [10, 16] over a training dataset without cleaning and repairing it. Generally speaking, these methods check whether incomplete or inconsistent examples influence the target model. If this is not the case, they return the model learned over the original training data. This approach may save significant time and effort spent repairing data.

However, it is not clear whether the methods above can be used to check the necessity of data repair for other ML models. As opposed to learning Datalog rules or K-nearest neighbors, training popular ML models usually requires optimizing a continuous loss function. Moreover, these methods detect the necessity of data repair only for classification problems and do not handle learning over missing data for regression models. Also, each of these methods handles a single ML model. Due to the relatively large number and variety of ML models, one would ideally like to have a single approach to the problem of learning over data with missing values for multiple types of ML models.

In this paper, we aim to develop a general approach for learning accurate ML models over training data with missing values without any data repair. We focus on ML models that optimize loss functions over continuous spaces, which arguably contain the most popular ML models. We formally define the necessity of data repair for learning accurate models over training data with missing values. Our methods efficiently detect whether data repair is needed to learn accurate models. If data repair is not necessary, they learn effective models over the original training data. Particularly, we make the following contributions:

- We formally define the conditions where data repair is not needed for training optimal models over incomplete data for a large group of ML models (Section 3).
- We prove necessary and sufficient conditions for learning an optimal model without repairing incomplete data for *linear regression*. Based on these conditions, we design an efficient algorithm for 1) checking the existence of the optimal model, and 2) learning the optimal model if it exists (Section 4).
- We prove necessary and sufficient conditions for learning an optimal model without repairing incomplete data for *linear Support Vector Machine (SVM)*, a popular *classification* ML model. We present an efficient algorithm for checking and then learning the optimal model if it exists (Section 5).
- Linear SVM models only learn linear classifiers, limiting their representation power in nonlinear spaces. We prove necessary and sufficient conditions for learning an optimal model without repairing incomplete data for two popular *kernel SVMs*. Then we provide algorithms to check and then learn the optimal models for *each* kernel SVM (Section 6)
- We formalize the notion of certain models for *Deep Neural Networks* (DNNs). Due to the non-convexity of the loss functions in DNNs, it is challenging to design an algorithm that efficiently finds the optimal model for them. We prove the necessary conditions for having certain models for DNNs in some special cases (Section 7).
- It might not be possible to learn an optimal model over incomplete data without any data repair. Hence, we introduce and formally define the condition under which it is possible to

learn a model that is sufficiently close to an optimal model over incomplete data without any repair. We propose novel and efficient algorithms to check for the existence of these models over linear regression and SVM (Section 8).
- We conduct experiments to show cost savings in data cleaning and program execution time compared to mean imputation, a deep learning-based imputation algorithm, and a benchmark framework across real-world datasets with random corruption. We also extend the comparison to diverse real-world datasets with inherent missing values, yielding results consistent with randomly corrupted datasets. Our studies show that our algorithms significantly reduce data repair costs when optimal or approximately optimal models can be learned over incomplete data and introduce minimal computational overhead in other cases (Section 9).

2 BACKGROUND

2.1 Supervised Learning

In this section, we review ML terminology and notations.

Table 1. A training dataset for rain prediction

	Temperature(F)	Humidity(%)	Rainfall
Seattle	65	80	1
New York	50	*null*	-1

Dataset. In ML, we work with a relation consisting of a finite number of attributes and tuples. *For instance, the relation shown in Table 1 has two tuples and three attributes.* For an ML problem, a relation with tuples and attributes is generally referred to as a **dataset** with **rows** and **columns**. In supervised learning, an ML model takes certain columns from a dataset as input and makes predictions for a designated output column.

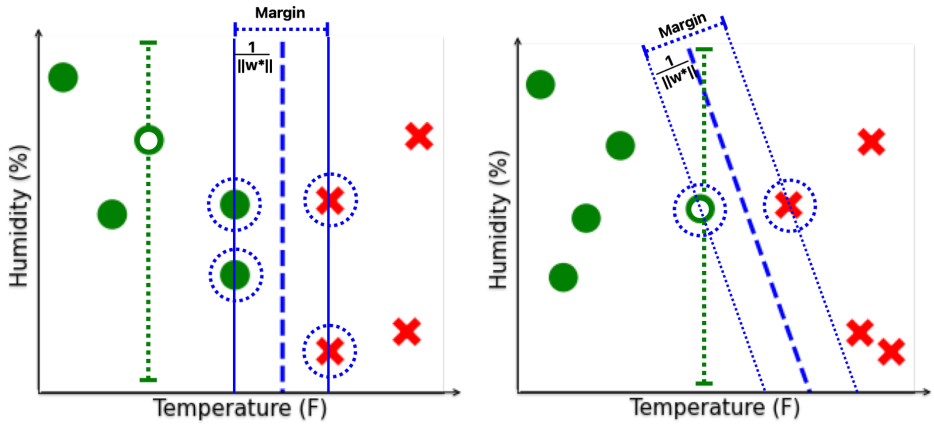
Features. The columns of the dataset we provide as input to an ML model are called features. *In Table 1, Temperature and Humidity are the two features that provide information on atmospheric conditions.* We denote a single feature as $\mathbf{z}$ and d features as $[\mathbf{z}_1, ..., \mathbf{z}_d]$. The *domain* of feature $\mathbf{z_i}$ is the set of values that appear in feature $\mathbf{z_i}$. To simplify our exposition, we assume that the domain of all values in a feature is the set of real numbers $\mathbb{R}$.

Label. The column of the dataset we want an ML model to make predictions on is called a label. *In our example, given current atmospheric conditions we want to predict chances of Rainfall. Therefore Rainfall is the label column, and it takes on two possible values: -1 to denote No Rain and 1 for Rain.* We represent a single label as y and the entire label column, consisting of n labels, as a vector $\mathbf{y} = [y_1, ..., y_n]$.

Training Example. We refer to a row in the dataset as a training example. *In Table 1, we observe two examples, Seattle and New York.* We denote a single training example as $\mathbf{x}$. For n training examples, a **training set** is a collection of an input matrix $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]^T$ and a corresponding label vector $\mathbf{y} = [y_1, ..., y_n]^T$. Each training example with d features in $\mathbf{X}$ can be expressed as a vector $\mathbf{x}_i = [x_{i1}, ..., x_{id}]$, where x_{ij} represents the j^{th} feature in the i^{th} example.

Target Function. We define the domain of examples as $\mathcal{X}$ and the domain of labels as $\mathcal{Y}$. For n examples and d features, we assume, $\mathcal{X}$ and $\mathcal{Y}$ are $\mathbb{R}^{n\times d}$, and $\mathbb{R}^n$, respectively. A target function $f(\mathbf{X}, \mathbf{w})$ transforms feature inputs into label outputs based on model $\mathbf{w}$, represented

as $f(\mathbf{X}, \mathbf{w}) : \mathcal{X} \rightarrow \mathcal{Y}$. Here, model $\mathbf{w}$ is a real-valued vector parameterizing the space of target functions that map from $\mathcal{X}$ to $\mathcal{Y}$. For instance, consider a single training example $\mathbf{x_i}$ consisting of d features. Given a vector of real numbers $\mathbf{w}$, the target function may be a *linear transformation* of the example $\mathbf{x_i}$, i.e $f(\mathbf{x_i}, \mathbf{w}) = w_1 \cdot x_{i1} + ... + w_d \cdot x_{id}$.



(a) Data cleaning is not needed (b) Data cleaning is needed

Fig. 1. Data cleaning may not always be necessary

Example 2.1. *Consider Figure 1a, which uses a popular ML algorithm called Support Vector Machine (SVM). The goal is to learn a linear boundary (blue rectangle) for rain prediction using temperature and humidity features from different cities (examples* $\mathbf{X}$*). The boundary (margin) separates the examples based on their Rain outcomes. The target function transforms all examples to one of the two possible* $\mathbf{y}$ *values [1, -1]. The approximation of the* **target function** *is* $f(\mathbf{X}, \mathbf{w})$=$\mathbf{w}^T\mathbf{X}$.

Loss function. A loss function, $\mathcal{L}$, is defined as a mapping of prediction for an example $\mathbf{x_i}$, i.e., $f(\mathbf{x_i}, \mathbf{w})$, with its corresponding label y_i to a real number $l \in \mathbb{R}$. l captures the similarity between $f(\mathbf{x_i}, \mathbf{w})$ and y_i. The exact form of the loss function varies between ML problems. One reasonable measure to capture similarity is to get the difference between prediction $f(\mathbf{x_i}, \mathbf{w})$ and actual label y_i. Aggregating over the n examples in the input matrix ($\mathbf{X}$), we find the *overall loss function*, L: $L(f(\mathbf{X}, \mathbf{w}), \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x_i}, \mathbf{w}), y_i) = \frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x_i}, \mathbf{w}) - y_i)^2$. For the rest of the paper, we will refer to the *'overall loss function'* as the **loss function** since we will be working with a matrix of examples rather than individual examples.

Example 2.2. *For the SVM in Figure 1, the* ***loss function****, L, is defined as* $L(f(\mathbf{X}, \mathbf{w}), \mathbf{y}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n} \max\{0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\}$.
Here, $\mathbf{w}^T\mathbf{x}_i$ *comes from the* ***target function*** *and represents the model's prediction for an example* $\mathbf{x}_i$*. The actual label is denoted as* y_i*. When the prediction and the label have the same sign, they are similar, therefore the loss is lower. The notation* $||\cdot||_2^2$ *represents the squared Euclidean norm, and* $C \in (0, +\infty)$ *is a tunable parameter.*

Classification and Regression. Supervised learning is divided into two types of ML problems. In a classification problem, the label domain $\mathcal{Y}$ consists of discrete values *(such as Rain(1) or No Rain(-1))*. Whereas if the label domain consists of continuous values *(e.g. inches of Rainfall)*, then it is a regression problem.

Model Training. Taking an input matrix $\mathbf{X}$, a label vector $\mathbf{y}$, a targte function f, and a loss function L, the goal of training is to find an optimal model $\mathbf{w}^*$ that minimizes the **training loss**, i.e., $\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathcal{W}} L(f(\mathbf{X}, \mathbf{w}), \mathbf{y})$.

Example 2.3. *For the SVM in Figure 1, **optimal model** $\boldsymbol{w}^*$ is the model that creates the widest margin between the example with different labels (red and green) while ensuring accurate predictions, to minimize training loss.*

2.2 Missing Values and Repairs

In this section, we formally define concepts for missing value repair.

Missing values. Any x_{ij} is a missing value (MV) if it is unknown (marked by *null*). We use the term **incomplete example** to refer to an example with missing values, and **incomplete feature** for a feature that contains missing values. Conversely, we use the terms **complete feature** and **complete example** to describe features and examples that are free of missing values. We further denote the set of incomplete examples as $MV(\mathbf{x}) = \{\mathbf{x}_i | \exists x_{ij}, x_{ij} = null\}$, and the set of incomplete features as $MV(\mathbf{z}) = \{\mathbf{z}_j | \exists x_{ij}, x_{ij} = null\}$.

Example 2.4. *In Table 1, the Humidity feature is an **incomplete feature** while the Temperature feature is a **complete feature**. Similarly, the New York example is an **incomplete example**, and the Seattle example is a **complete example***

Repair. A repair is a complete version of the raw data where all missing values (MV) are imputed i.e. replaced with values from the domain of features or examples (Subsection 2.1). More formally:

Definition 1. *(Repair) For an input matrix $\boldsymbol{X}$ having missing values (MV), $\boldsymbol{X}^r$ is a repair to $\boldsymbol{X}$ if 1) $dimension(\boldsymbol{X}^r) = dimension(\boldsymbol{X})$, 2) $\forall x^r_{ij} \in \boldsymbol{X}^r, x^r_{ij} \neq null$, and 3) $\forall x_{ij} \neq null, x^r_{ij} = x_{ij}$.*

Example 2.5. *In Table 1, the Humidity feature for the New York **example** has a missing value. From Definition 1, replacing the missing data with a value (e.g. 90) yields a **repair** ($\boldsymbol{X}^r$). However, deleting the humidity feature, which eliminates the missing value, is not a repair since it changes dimension(X).*

Set of possible repairs. The range of values that can be used to replace missing values is large. Consequently, a large number of repairs may exist. We denote this set of all possible repairs as $\mathbf{X}^R$. For brevity, we refer to *'a value replacing the missing value'* as a **repairing value**.

3 CERTAIN MODELS

In this section, we formally define certain models that minimize training loss irrespective of how missing data is repaired.

Definition 2. *(Certain Model) A model $\boldsymbol{w}^*$ is a certain model if:*

$$\forall \boldsymbol{X}^r \in \boldsymbol{X}^R, \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\mathcal{W}} L(f(\boldsymbol{X}^r, \boldsymbol{w}), \boldsymbol{y}) \tag{1}$$

Where $\boldsymbol{X}^r$ is one possible repair, $\boldsymbol{X}^R$ is the set of all possible repairs and $L(f(\boldsymbol{X}^r, \boldsymbol{w}), \boldsymbol{y})$ is the loss function

Definition's intuition: Intuitively, Definition 2 says that if a model is optimal (minimizes the training loss) for all possible repairs, this model is a certain model.

Example 3.1. *Consider the ML problem in Figure 1. Figures 1a and 1b display two sets of training examples with a missing humidity value, possibly due to a malfunctioning sensor. The green dashed line represents the range of possible values for the incomplete feature (empty circle). In Figure 1a, the*

incomplete example does not touch the blue rectangle in any possible repair ($\boldsymbol{X}^r \in \boldsymbol{X}^R$). Therefore, the model (decision boundary: blue dashed line) is optimal for all repairs. Hence, a certain model ($\boldsymbol{w}^*$) exists. But, in Figure 1b, since the example may touch the blue rectangle in many repairs, the optimal model changes from one repair to another and certain models do not exist.

Advantages of finding certain models: To repair incomplete data, users may resort to methods such as deleting data (e.g., entire examples or features), potentially leading to information loss. Another option is data imputation, which requires additional effort and domain expertise [30]. The data cleaning effort substantially increases with the exponentially growing number of ML applications since good imputations depend on downstream ML applications[20]. For instance, if the goal is to train a hundred different models with a dataset, a user may need to impute the dataset a hundred times by different imputation methods. Regardless of how well these data repair techniques are constructed, they may produce suboptimal results, i.e., the repaired data is not the ground truth [22]. However when a certain model exists, *imputing missing data is unnecessary* since this model is optimal for all possible repairs. Therefore users may save a significant amount of time and effort by finding certain models. Users may ignore missing values in practice to investigate the properties of the trained model. Nonetheless, there is no guarantee that their trained model is accurate. The concept of certain models ensures cases for which this approach leads to accurate models.

Prevalence of certain models: Certain models may not often exist from the restrictive definition (a model is optimal for *all* repairs). However, when they exist, we save a significant amount of time and resources. Furthermore, these savings significantly grow as the number of datasets increases alongside the rapid expansion of the ML community utilizing these datasets for model training.

Problems: We aim to solve our problem of finding certain models by solving the following sub-problems.

(1) **Certain Model Checking**: Given **inputs** (1) a training set consisting of a feature matrix $\mathbf{X}$ potentially with missing values and a label vector $\mathbf{y}$ (2) a target function $f(\mathbf{X}, \mathbf{w})$ and (3) a loss function L. The first problem is to determine *whether a certain model $\boldsymbol{w}^*$ exists* that minimizes the training loss $L(f(\mathbf{X}, \mathbf{w}), \mathbf{y})$ for all repairs ($\forall \mathbf{X}^r \in \mathbf{X}^R$) to the incomplete dataset. If a certain model ($\mathbf{w}^*$) exists, it implies that data imputation is unnecessary.
(2) **Certain Model Learning**: If a certain model exists, then the second problem is *learning a certain model* ($\mathbf{w}^*$), given a training set, loss function, and a target function as **inputs**. This certain model **output** can be used for downstream tasks.

Minimal overhead of not finding certain models: When certain models exist users *do not have to spend any effort* in repairing missing data. When certain models do not exist, the effort to check for them may appear wasteful. Therefore, an ideal solution would require minimal time to check for certain models even when they do not exist. Consequently, the *overhead of checking certain models* is negligible compared to the significant time and resources users may save by *finding certain models.*

Baseline Algorithm: Given Equation 1, a *baseline algorithm* for checking and learning a certain model is: (1) learning *possible models* from all possible repairs one by one, and (2) a certain model exists if all repairs share at least one mutual optimal model. Here, the set of possible repairs is often large (Subsection 2.2). Therefore, *learning models from all repairs may be incredibly slow.* More precisely, if we denote the training time for learning one model as $O(T_{train})$, the baseline algorithm's complexity grows in proportion to the size of all possible repairs ($\mathbf{X}^R$). This results in a complexity of $O(|\mathbf{X}^R| * T_{train})$, where $|\mathbf{X}^R|$ represents the total number of possible repairs. Therefore, we aim to find *efficient algorithms to check for certain models in multiple ML problems.*

4 CERTAIN MODELS FOR LINEAR REGRESSION

Linear regression is a popular and classic ML model. It assumes a linear relationship between feature input ($\mathbf{X}$) and label output ($\mathbf{y}$). The difference between the model's prediction and actual label, $\mathbf{Xw} - \mathbf{y}$, is the residue $\mathbf{e} = [e_1, ..., e_n]$.

The loss function (Section 2) for linear regression is $L(f(\mathbf{X}, \mathbf{w}), \mathbf{y}) = ||\mathbf{Xw} - \mathbf{y}||_2^2$. Here, $\| \cdot \|_2^2$ represents the squared Euclidean norm.

4.1 Conditions For Having Certain Models

Based on the definition of certain models (Definition 2), the **certain model $\mathbf{w}^*$ for linear regression** is defined as:

$$\forall \mathbf{X}^r \in \mathbf{X}^R, \mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathcal{W}} ||\mathbf{X}^r \mathbf{w} - \mathbf{y}||_2^2 \tag{2}$$

Where $\boldsymbol{X}^r$ is one possible repair to the input matrix $\boldsymbol{X}$, $\boldsymbol{X}^R$ is set of all possible repairs and $||\boldsymbol{X}^r \boldsymbol{w} - \boldsymbol{y}||_2^2$ is the loss function

Linear regression finds a model $\mathbf{w}^* \in \mathbb{R}^d$ such that the linear combination of all feature vectors, $w_1^* \mathbf{z}_1 + ... + w_d^* \mathbf{z}_d$, has the shortest Euclidean distance to the label vector $\mathbf{y}$, i.e., the minimum training loss. Intuitively, a certain model exists when this Euclidean distance is independent of any incomplete features $\mathbf{z}_j$, $j \in MV(\mathbf{z})$.

To formalize this intuition and avoid checking for all possible repairs, we introduce Theorem 4.2. Given an input matrix with n examples and d features, $\mathbf{X} \in \mathbb{R}^{n \times d}$, we denote a missing-value-free (**c**omplete) matrix $\mathbf{X}_c \in \mathbb{R}^{n \times m}$ as a submatrix ($m < d$) of the input matrix. $\mathbf{X}_c$ only consists of the m complete features $\mathbf{z}_j$ from $\mathbf{X}$, $\mathbf{z}_j \notin MV(\mathbf{z})$. Performing linear regression with $\mathbf{X}_c$ and the labels $\mathbf{y}$, we get the model $\mathbf{w}_c^* \in \mathbb{R}^m$. To facilitate subsequent analysis, we introduce another model $\mathbf{w}^\bullet$ by expanding $\mathbf{w}_c^*$ from $\mathbb{R}^m$ to $\mathbb{R}^d$ by appending $(d - m)$ zero coefficients corresponding to incomplete features. For example, if the columns 2 and 4 in $\mathbf{X} \in \mathbb{R}^4$ contain missing values, and $\mathbf{w}_c^* = [1, 1]^T$, we create $\mathbf{w}^\bullet$ by expanding $\mathbf{w}_c^*$ to $\mathbb{R}^4$ and inserting zeros in the second and fourth entries. This process results in an *expanded model*, $\mathbf{w}^\bullet = [1, 0, 1, 0]^T$. This step aligns the linear coefficients between $\mathbf{X}_c$ and $\mathbf{X}^r$, simplifying the following theorems and proof.

Lemma 4.1. *If a certain model $\boldsymbol{w}^*$ exists, $\forall \boldsymbol{z}_j \in MV(\boldsymbol{z})$, the corresponding coefficient $w_j^* = 0$. In other words, if a certain model exists, $\boldsymbol{w}^\bullet$ is a certain model.*

The proofs of the lemmas and theorems in this paper are detailed in our technical report [35]. Based on Lemma 4.1, we have the following result.

Theorem 4.2. *A certain model exists if and only if ; $\forall \boldsymbol{z}_j \in MV(\boldsymbol{z})$, these conditions are met: 1) $\forall x_{ij} = null$, $e_i = 0$; 2) $\sum_{x_{ij} \neq null} x_{ij} \cdot e_i = 0$.*

4.2 Checking and Learning Certain Models

Theorem 4.2 says that a certain model exists for linear regression if and only if the residue vector $\mathbf{e}$ is orthogonal to incomplete features. If a certain model exists, the incomplete features may be safely ignored without compromising the minimization of training loss since they do not contribute to a smaller training loss than $\mathbf{e}$.

Based on Theorem 4.2, we present Algorithm 1. Our algorithm has two major steps: 1) computing the residue vector $\mathbf{e}$ along with expanded model $\mathbf{w}^\bullet$ based on complete features, and 2) checking the orthogonality between $\mathbf{e}$ and all incomplete features. Finally, we obtain a certain model when it exists by getting $\mathbf{w}^\bullet$, in which the incomplete features are ignored by the zero linear coefficients.

The algorithm's time complexity is $O(T_{train})$, which is significantly faster than the baseline we discuss in Section 3. The efficiency of our algorithm stems from its ability to check for certain models without traversing over all possible repairs.

Algorithm 1 Checking and learning certain model for Linear Regression

```
MV(z) ← features with missing values (incomplete features)
w• ← expanded model trained with complete features
e ← fitting residue with complete features
n ← the number of training examples
for z_j ∈ MV(z) do
    innerProduct ← 0
    for i = 1, 2, ..., n do
        if x_ij = null AND e_i ≠ 0 then
            return "Certain model does not exist"
        else if x_ij ≠ null then
            innerProduct ← innerProduct + x_ij * e_i
        end if
    end for
    if innerProduct ≠ 0 then
        return "Certain model does not exist"
    end if
end for
return "A certain model w• exists"
```

5 CERTAIN MODELS FOR SVM

Another widely used ML model is SVM. In this section, we are specifically interested in *linear* SVM, which aims to learn a linear decision boundary to classify examples. This decision boundary is of the form $\mathbf{w}^T\mathbf{x} = 0$.

A typical soft-margin SVM's loss function comprises of a loss term and a regularizer, $L(f(\mathbf{X}, \mathbf{w}), \mathbf{y}) = \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{n} max\{0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\}$. Here, the first term is the regularization, the second term is the *hinge loss* [12], and $C \in (0, +\infty)$ is a tunable parameter. **Support vectors** are the closest training examples that decide a decision boundary, i.e. $(\mathbf{x}_i, y_i)$ is a support vector if $y_i\mathbf{w}^T\mathbf{x}_i \leq 1$.

5.1 Conditions For Having Certain Models

Similar to the definition in Subsection 4.1, **certain model, $\mathbf{w}^*$, for SVM** is defined as:

$$\forall \mathbf{X}^r \in \mathbf{X}^R, \mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathcal{W}} [\frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{n} max\{0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\}] \tag{3}$$

Where $\mathbf{X}^r$ denotes one possible repair, and $\mathbf{X}^R$ is the set of all possible repairs. $\mathbf{x}_i$ is an input example with d features, and y_i is its corresponding label. $\boldsymbol{w}^T\boldsymbol{x}_i$ comes from the target function and measures the proximity between the example $\boldsymbol{x}_i$ and the decision boundary

An SVM leverages support vectors to construct a decision boundary for classifying examples. Therefore, the existence of a certain model for an SVM implies that either incomplete examples are not support vectors in any repairs, or incomplete examples are support vectors in some repairs but exactly standing on the functional margin (Lemma 5.1).

Lemma 5.1. *If a certain model $\boldsymbol{w}^*$ exists, there are only two possible cases and they do not have any overlap. Case 1: none of the incomplete examples is a support vector with respect to $\boldsymbol{w}^*$ in any repair, i.e., $\forall \boldsymbol{X}^r \in \boldsymbol{X}^R, \forall \boldsymbol{x}_i \in MV(\mathbf{x}),\ y_i \boldsymbol{w}^{*T} \boldsymbol{x}_i^r > 1$. Case 2: $\exists \boldsymbol{x}_i \in MV(\mathbf{x}),\ y_i \boldsymbol{w}^{*T} \boldsymbol{x}_i^r = 1$. Also, $\forall \boldsymbol{z}_j \in MV(\mathbf{z}), w_j^* = 0$. And $\forall \boldsymbol{x}_i \in MV(\mathbf{x}),\ y_i \boldsymbol{w}^{*T} \boldsymbol{x}_i^r \geq 1$.*

Based on Lemma 5.1, properties in Lemmas 5.2 and 5.3 hold.

Lemma 5.2. *If a certain model exists by Case 1 in Lemma 5.1, $\mathbf{w}^\diamond$ is the certain model.*

Lemma 5.3. *If a certain model exists by Case 2 in Lemma 5.1, models trained with any repairs of $\mathbf{X}$ are certain models.*

To formalize this intuition, we present Theorem 5.4 to check for certain models. Similar to the notations used in Subsection 4.1, we denote a **c**omplete matrix $\mathbf{X}_c$ as a submatrix of $\mathbf{X}$ that consists of all the **c**omplete examples $\mathbf{x}_i, \mathbf{x}_i \notin MV(\mathbf{x})$. Similarly, we define a subvector $\mathbf{y}_c$ to include all labels corresponding to these complete training examples. We denote the SVM model trained with these complete examples and labels as $\mathbf{w}^\diamond = [w_1^\diamond, ..., w_d^\diamond]^T$.

Theorem 5.4. *A certain model exists if and only if one of the two sets of conditions below is met. Set 1: 1) $\forall \boldsymbol{z}_j \in MV(\mathbf{z}), w_j^\diamond = 0$, 2) $\forall \boldsymbol{x}_i \in MV(\mathbf{x}),\ y_i \sum_{x_{ij} \neq null} w_j^\diamond x_{ij} > 1$. Set 2: 1) training a model $\mathbf{w}'$ with a random repair $\mathbf{X}^{r'} \in \mathbf{X}^R, \forall \boldsymbol{z}_j \in MV(\mathbf{z}), w_j' = 0$, 2) $\forall \boldsymbol{x}_i \in MV(\mathbf{x}),\ y_i \sum_{x_{ij} \neq null} w_j' x_{ij} \geq 1$.*

5.2 Checking and Learning Certain Models

Theorem 5.4 says that a certain model for SVM exists if and only if none of the incomplete training examples are support vectors. Therefore, these incomplete examples are *redundant* when it comes to learning the decision boundary given other complete examples.

Using Theorem 5.4, we propose Algorithm 2 with two major steps: 1) learning $\mathbf{w}^\diamond$ from complete training examples, and checking the conditions in Set 1 in Theorem 5.4 against $\mathbf{w}^\diamond$. If a certain model exists, $\mathbf{w}^\diamond$ is the certain model. 2) If certain models are not found in step 1, learning $\mathbf{w}'$ from an arbitrary repair, and checking the conditions in Set 2 against $\mathbf{w}'$. If a certain model exists from this step, $\mathbf{w}'$ is the certain model. The algorithm's time complexity is $O(T_{train})$ as model training is the dominant part compared to condition checking.

6 CERTAIN MODELS FOR KERNEL SVM

SVM models in Section 5 can only separate classes linearly, limiting their representation power in the nonlinear space. A natural approach to overcome this limitation is to use *kernel SVM*.

Training a nonlinear model while maintaining the properties of linear SVM, a *kernel SVM* first maps the input feature vectors, denoted as $\mathbf{X}$, into a higher-dimensional space, often referred to as the *kernel space*, through a non-linear transformation Φ. After this transformation, the kernel SVM seeks to learn a linear SVM model within the kernel space. Therefore, the resulting model is non-linear with respect to the original feature space, while remaining linear within the kernel space.

However, transforming all training examples into kernel space is computationally expensive. To avoid this cost, **kernel function** $k(\mathbf{x}_1, \mathbf{x}_2) =< \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) >: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ offers a shortcut for computing inner products between two vectors in the kernel space without explicit transformation.

We presented the *primal problem* to linear **SVM's model training** in Section 5. Here, *to make use of kernel functions*, we present SVM training in terms of inner products through its *dual problem*.

$$\max_{\mathbf{a} \in \mathbb{R}^n} \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{4}$$

Algorithm 2 Checking and learning certain models for linear SVM

$MV(\mathbf{z}) \leftarrow$ incomplete features
$MV(\mathbf{x}) \leftarrow$ incomplete examples
$\mathbf{w}^{\diamond} \leftarrow$ the model trained with complete training examples
for $\mathbf{z}_j \in MV(\mathbf{z})$ **do**
  if $w_j^{\diamond} \neq 0$ **then**
    Case 1 $\leftarrow$ False
  end if
end for
if Case 1 $\neq$ False **then**
  for $\mathbf{x}_i \in MV(\mathbf{x})$ **do**
    if $y_i \sum_{x_{ij} \neq null} w_j^{\diamond} x_{ij} \leq 1$ **then**
      Case 1 $\leftarrow$ False
    end if
  end for
end if
if Case 1 $\neq$ False **then**
  return "A certain model $\mathbf{w}^{\diamond}$ exists"
else
  $\mathbf{w}' \leftarrow$ the model trained with an arbitrary repair
  for $\mathbf{z}_j \in MV(\mathbf{z})$ **do**
    if $w'_j \neq 0$ **then**
      return " Certain models do not exist"
    end if
  end for
  for $\mathbf{x}_i \in MV(\mathbf{x})$ **do**
    if $y_i \sum_{x_{ij} \neq null} w'_j x_{ij} < 1$ **then**
      return " Certain models do not exist"
    end if
  end for
  return "A certain model $\mathbf{w}'$ exists"
end if

$$\text{s.t.} \quad C \geq a_i \geq 0, i = 1, ..., n$$
$$\sum_{i=1}^{n} a_i y_i = 0$$

Based on this dual formulation, one can show that $\mathbf{w}^* = \sum_{i=1}^{n} a_i^* y_i \phi(\mathbf{x}_i)$ where $\mathbf{a}^* = [a_1^*, ..., a_n^*]^T$ is the solution to the dual problem. In Section 5, a training example $(\mathbf{x}_i, y_i)$ is a support vector in **linear space** if $y_i \mathbf{w}^{*T} \mathbf{x}_i \leq 1$. Representing $\mathbf{w}^{*T}$ by its dual form, a training example $(\mathbf{x}_i, y_i)$ is a support vector in **kernel space** if $y_i \sum_{j=1}^{n} a_j^* y_j k(\mathbf{x}_i, \mathbf{x}_j) \leq 1$.

6.1 Conditions For Having Certain Models

The kernel function transforms input data to a higher dimension while the SVM model remains linear. The linear properties of the kernel SVM are preserved within the kernel space. Hence, certain model conditions in Section 5 still apply. *A certain model exists if and only if none of the incomplete examples are support vectors for any repair in the kernel space.*

We now formally present these conditions for kernel SVM. Following the same notations used in Section 5, we use $\mathbf{w}^\diamond$ to denote the model learned from $\mathbf{X}_c$, the subset of data that only containing complete training examples, and $\mathbf{y}_c$, the corresponding labels. As derived from Equation 4, $\mathbf{w}^\diamond = \sum_{\mathbf{x}_j \notin MV(\mathbf{x})} a_j^\diamond y_j \phi(\mathbf{x}_j)$. Hence, $\mathbf{x}_i^r$, a repair to an incomplete training example, is a support vector in kernel space if $y_i \sum_{\mathbf{x}_j \notin MV(\mathbf{x})} a_j^\diamond y_j k(\mathbf{x}_i^r, \mathbf{x}_j) \leq 1$. Therefore, the **certain model conditions for kernel SVM** are represented as:

$$\forall \mathbf{x}_i \in MV(\mathbf{x}), \forall \mathbf{X}^r \in \mathbf{X}^R, y_i \sum_{\mathbf{x}_j \notin MV(\mathbf{x})} a_j^\diamond y_j k(\mathbf{x}_i^r, \mathbf{x}_j) \geq 1 \tag{5}$$

Further, seeking the opportunity to avoid materializing all possible repairs, we reformulate the above condition to an optimization problem over possible repairs:

$$\forall \mathbf{x}_i \in MV(\mathbf{x}), \min_{\mathbf{X}^r \in \mathbf{X}^R} y_i \sum_{\mathbf{x}_j \notin MV(\mathbf{x})} a_j^\diamond y_j k(\mathbf{x}_i^r, \mathbf{x}_j) \geq 1 \tag{6}$$

From the dual problem, we note that a complete example, $\mathbf{x}_j, \mathbf{x}_j \notin MV(\mathbf{x})$, is a support vector if and only if the corresponding solution $a_j^\diamond \neq 0$. Hence, only complete examples that are support vectors play a role in Inequality 6

In the following sections, we apply these general conditions for certain model existence in kernel SVM to popular kernel functions.

6.2 Polynomial kernel

The kernel function for a polynomial kernel is $k_{POLY}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^\lambda$, where $\lambda = 1, 2, 3, ...$ is the degree of the polynomial and $c \geq 0$ is a free parameter tuning the impact of higher-degree versus lower-degree terms.

We first intuitively look at how $k_{POLY}(\mathbf{x}_i, \mathbf{x}_j)$ remains the same value for all repairs. For an incomplete training example $\mathbf{x}_i$ and a complete example $\mathbf{x}_j$, $\mathbf{x}_i^T \mathbf{x}_j$ can be expanded to $x_{i1} \cdot x_{j1} + ... + x_{id} \cdot x_{jd}$. Suppose the m^{th} feature value x_{im} is missing in $\mathbf{x}_i$, the inner product $\mathbf{x}_i^T \mathbf{x}_j$ goes to infinity when $x_{im} = +\infty$ or $-\infty$, unless the corresponding element x_{jm} equals 0, which ensures $x_{jm} \cdot x_{im} = 0$. Hence, in order to satisfy Inequality 6, the **set of support vectors**, SV , for $\mathbf{w}^\diamond$ should have zero entries at all incomplete features $\mathbf{z}_m$. This condition enforces that the value for $k_{POLY}(\mathbf{x}_i^r, \mathbf{x}_j)$ *is independent of the missing value repairs*. We formalize these conditions in the following theorem.

Theorem 6.1. *A certain model exists if and only if the two conditions are met: 1)* $\forall \boldsymbol{x}_j \in SV$, $\forall \boldsymbol{z}_m \in MV(\mathbf{z})$, $x_{jm} = 0$, *and 2)* $\forall \boldsymbol{x}_i \in MV(\mathbf{x})$, $y_i \sum_{\boldsymbol{x}_j \in SV} a_j^\diamond y_j (\sum_{x_{iq} \neq null} x_{iq} \cdot x_{jq} + c)^\lambda > 1$

Checking and Learning Certain Models: Informally Theorem 6.1 says that a certain model for a polynomial kernel SVM (p-SVM) exists if (1) all the examples that are support vectors have zero entries for corresponding incomplete features and (2) all incomplete examples are not support vectors. Based on this theorem, Algorithm 3 efficiently checks and learns certain models. Similar to the algorithm for linear SVM in Section 5.2, if a certain model is determined to exist, $\mathbf{a}^\diamond$ is exactly the certain model based on Lemma 5.2. This algorithm's time complexity is also $O(T_{train})$.

6.3 RBF kernel

The RBF kernel function is $k_{RBF}(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2)$. This kernel function's transformation depends on the squared Euclidean distance between the two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$.

To check if a certain model exists for the polynomial kernel, we derived conditions for $k_{POLY}(\boldsymbol{x}_i^r, \boldsymbol{x}_j)$ to remain the *same* for all repairs. In contrast, $k_{RBF}(\boldsymbol{x}_i^r, \boldsymbol{x}_j)$ *changes* among repairs as the Euclidean distance between two vectors changes. Therefore, to check if a certain model exists for SVM with RBF kernel (RBF-SVM), we need to directly solve the minimization problem in Inequality 6.

Algorithm 3 Checking and learning certain models for p-SVM

```
MV(z) ← incomplete features
MV(x) ← incomplete examples
SV ← set of support vectors
a^◇ ← the model trained with complete training examples
for z_m ∈ MV(z) do
    for x_j ∈ SV do
        if x_j m ≠ 0 then
            return "Certain model does not exist"
        end if
    end for
end for
for x_i ∈ MV(x) do
    if y_i Σ_{x_j ∈ SV} a_j^◇ y_j (Σ_{x_iq ≠ null} x_iq · x_jq + c)^λ ≤ 1 then
        return "Certain model does not exist"
    end if
end for
return "A certain model a^◇ exists"
```

However, this optimization problem is *not convex*, which means it is hard to find a method for checking certain models with theoretical guarantees. Nonetheless, we can still discover the *lower bound* (lwb_i) of the following optimization target:

$$\forall \mathbf{X}^r \in \mathbf{X}^R, lwb_i \leq y_i \sum_{\mathbf{x}_j \notin MV(\mathbf{x})} a_j^{\diamond} y_j k_{RBF}(\mathbf{x}_i^r, \mathbf{x}_j) \tag{7}$$

For each missing value x_{im}, we denote the possible range of missing value repairs such that $x_m^{min} \leq x_{im}^r \leq x_m^{max}, \forall \mathbf{z}_m \in MV(\mathbf{z}), \forall \mathbf{X}^r \in \mathbf{X}^R$. This range may come from *integrity constraint* for features: any value in a feature $\mathbf{z}_m$ is between its minimum x_m^{min} and maximum x_m^{max}. Now, we apply this lower bound idea to reformulate the general certain model conditions for kernel SVM from Inequality 6.

Lemma 6.2. *For any kernel SVM, a certain model exists if*

$$\forall \boldsymbol{x}_i \in MV(\mathbf{x}), lwb_i = \sum_{\boldsymbol{x}_j \notin MV(\mathbf{x})} \min_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} \beta_{ij} k(\boldsymbol{x}_i^r, \boldsymbol{x}_j) > 1 \tag{8}$$

where $\beta_{ij} = y_i a_j^{\diamond} y_j$ *and*

$$\min_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} \beta_{ij} k(\boldsymbol{x}_i^r, \boldsymbol{x}_j) = \begin{cases} \beta_{ij} \min\limits_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} k(\boldsymbol{x}_i^r, \boldsymbol{x}_j) & \text{if } \beta_{ij} > 0 \\ \beta_{ij} \max\limits_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} k(\boldsymbol{x}_i^r, \boldsymbol{x}_j) & \text{if } \beta_{ij} < 0 \\ 0 & \text{if } \beta_{ij} = 0 \end{cases}$$

From Lemma 6.2, we see the **key to an efficient implementation** is to compute $\min\limits_{\mathbf{x}_i^r \in \mathbf{x}_i^R} k(\mathbf{x}_i^r, \mathbf{x}_j)$ and $\max\limits_{\mathbf{x}_i^r \in \mathbf{x}_i^R} k(\mathbf{x}_i^r, \mathbf{x}_j)$ without materializing repairs. We formalize this idea in Theorem 7.3.

Theorem 6.3. *For the RBF kernel, the minimum and maximum kernel function values between an incomplete example and a complete example are as follows:*

$$\min_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} k_{rbf}(\boldsymbol{x}_i^r, \boldsymbol{x}_j) = exp\{-\gamma\{ \sum_{x_{im}=null} MAX[(x_m^{max} - x_{jm})^2, (x_m^{min} - x_{jm})^2] + \sum_{x_{im}\neq null} (x_{im} - x_{jm})^2\}\}$$

$$\max_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} k_{rbf}(\boldsymbol{x}_i^r, \boldsymbol{x}_j) = exp\{-\gamma[\sum_{x_{im}\neq null} (x_{im} - x_{jm})^2]\}$$

Checking and Learning Certain Models: Similar to the algorithm for the polynomial kernel, we can use Theorem 6.3 to check and learn the certain model in $O(T_{train})$ time.

7 CERTAIN MODELS FOR DNN

DNNs are popular ML models for a wide variety of tasks such as natural language processing and image classification [2]. Training a DNN involves solving complex non-convex optimization problems, making the discovery of an optimal model a challenging task [8]. Finding a certain model for DNN adds another layer of difficulty because the certain model needs to be optimal for all repairs within the non-convex loss landscape.

Fortunately, some well-studied kernel SVMs have been shown to approximate DNNs [8]. Therefore, our goal in this section is to build on the conditions we prove for kernel SVMs in Section 6 to prove the conditions for having certain models for DNN.

More specifically, we employ the *arc-cosine* kernel, which is used in SVM to approximate DNN's computation [7]. The justification behind this approximation stems from the following property. Feeding two input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ individually into a single-layer network with polynomial activation functions, we obtain the corresponding output vectors $\mathbf{y}_i$ and $\mathbf{y}_j$. Under some assumptions, the inner product between these two output vectors can be represented by the arc-cosine kernel function, i.e., $k_{arccos}(\mathbf{x}_i, \mathbf{x}_j) =< \mathbf{y}_i, \mathbf{y}_j >$ [7]. This implies that the arc-cosine kernel function mimics the computation in a single-layer network. Then, iteratively performing kernel transformation, i.e., $< \phi(\phi(...\phi(\mathbf{x}_i))), \phi(\phi(...\phi(\mathbf{x}_j))) >$, should mimic the computation in a multi-layer network. The most basic arc-cos kernel function is defined by the inverse cosine of the dot product between two vectors divided by the product of their Euclidean norms, i.e. $k_{\arccos}(\mathbf{x}_i, \mathbf{x}_j) = \cos^{-1}\left(\frac{\mathbf{x}_i \cdot \mathbf{x}_j}{|\mathbf{x}_i| \cdot |\mathbf{x}_j|}\right)$. By discovering the certain model conditions for SVM with the arc-cosine kernel (arccos-SVM), we approximate the certain model conditions for DNN.

To check the existence of certain models, we need to solve the minimization problem in Inequality 6. As the problem is non-convex, we find a lower bound (lwb_i) for the necessary condition:

$$\forall \mathbf{X}^r \in \mathbf{X}^R, lwb_i \leq y_i \sum_{\mathbf{x}_j \notin MV(\mathbf{x})} a_j^{\diamond} y_j k_{arccos}(\mathbf{x}_i^r, \mathbf{x}_j)$$

Following a similar approach as we describe in Subsection 6.3, we look for the lower bound defined in Lemma 6.2. The key of this process is to find the minimum and maximum values for $k_{arccos}(\mathbf{x}_i^r, \mathbf{x}_j)$ for a pair of incomplete example $\mathbf{x}_i$ and complete example $\mathbf{x}_j$.

However, finding the minimum and maximum values for the arc-cosine kernel function in the presence of missing data is also challenging due to the non-convex nature of the problem. Nonetheless, when each incomplete example $\mathbf{x}_i$ has only one missing value x_{iz}, the problem significantly simplifies. In the following proof, we show that any stationary point is a global

minimum under this assumption. Therefore, our analysis focuses on training sets with one missing value per example. The investigation of scenarios with multiple missing values per example is left for future work.

Following Theorem 6.3, we formalize this idea in Theorem 7.1.

Theorem 7.1. *For arc-cos kernel, the maximum and the minimum kernel function values between an incomplete example ($\boldsymbol{x}_i^r$) and a complete example ($\boldsymbol{x}_j$) are as follows:*

$$\max_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} k_{arccos}(\boldsymbol{x}_i^r, \boldsymbol{x}_j) = \pi - MAX[cos^{-1}(\frac{a}{c}), cos^{-1}(-\frac{a}{c})]$$

$$\min_{\boldsymbol{x}_i^r \in \boldsymbol{x}_i^R} k_{arccos}(\boldsymbol{x}_i^r, \boldsymbol{x}_j) = \pi - cos^{-1}(\frac{a^2 \cdot d + b^2}{c \cdot \sqrt{a^2 \cdot d^2 + b^2 \cdot d}})$$

Suppose $x_{iz} = null$. To simplify notations, we define $a = x_{jz}$, $b = \sum_{x_{ip} \neq null} x_{ip} \cdot x_{jp}$, $c = ||\boldsymbol{x}_j||$, and $d = \sum_{x_{ip} \neq null} x_{ip}^2$.

Theorem 7.1 shows that the maximum and minimum values for the arc-cos kernel can be efficiently computed without materializing repairs. Further, plugging these values in Lemma 6.2, we approximate a certain model condition for DNN that says a certain model exists if Inequality 8 holds.

8 APPROXIMATELY CERTAIN MODELS

The conditions for having certain models might be too restrictive for many datasets as it requires a single model to be optimal for all repairs of a dataset. In practice, however, users are usually satisfied with a model that is *sufficiently close* to the optimal one. In this section, we leverage this fact and propose the concept of *approximately certain model*, which relaxes the conditions on certain models. An approximately certain model is within a given threshold from every optimal model for each repair of the input dataset. If there is an approximately certain model for a training task, users can learn over incomplete data and skip data cleaning. We also propose novel and efficient algorithms for finding approximately certain models for linear regression and SVM.

8.1 Formal Definition

Definition 3. *(Approximately Certain Model) Given a user-defined threshold $e \geq 0$, the model $\mathbf{w}^{\approx}$ is an approximately certain model (ACM) if the following condition holds:*

$$\forall \mathbf{X}^r \in \mathbf{X}^R, L(f(\mathbf{X}^r, \mathbf{w}^{\approx}), \mathbf{y}) - \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{X}^r, \mathbf{w}), \mathbf{y}) \leq e \tag{9}$$

where $\boldsymbol{X}^r$ is a possible repair, $\boldsymbol{X}^R$ is the set of all possible repairs and $L(f(\boldsymbol{X}^r, \boldsymbol{w}), \boldsymbol{y})$ is the loss function.

Definition 3 ensures that the training losses of ACMs are close to the minimal training loss for all repairs. Therefore, when e is sufficiently small, ACMs are accurate for all repairs. In this scenario, data imputation is unnecessary and users can proceed with an ACM without compromising the model's performance significantly. Certain models are special cases of ACMs by setting $e = 0$.

8.2 Learning ACMs Efficiently

The condition in Definition 3 is equivalent to $g(\mathbf{w}') \leq e$ where

$$g(\mathbf{w}') = \sup_{\mathbf{X}^r \in \mathbf{X}^R} h(\mathbf{w}', \mathbf{X}^r) \tag{10}$$

and

$$h(\mathbf{w}', \mathbf{X}^r) = L(f(\mathbf{X}^r, \mathbf{w}'), \mathbf{y}) - \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{X}^r, \mathbf{w}), \mathbf{y})$$

If there is a model $\mathbf{w}' \in \mathcal{W}$ that satisfies this condition, it is an ACM. Hence, to find an ACM, we can check every $\mathbf{w}' \in \mathcal{W}$ for the condition in 10. This is equivalent to checking $\min_{\mathbf{w}' \in \mathcal{W}} g(\mathbf{w}') \leq e$.

Lemma 8.1. *The problem* $\min_{\mathbf{w}' \in \mathcal{W}} g(\mathbf{w}') \leq e$ *is convex for every model whose loss function* $L(f(\mathbf{X}, \mathbf{w}), \mathbf{y})$ *is convex with respect to* $\mathbf{w}$.

The loss functions of many types of models including linear regression and SVM are convex with respect to $\mathbf{w}$. Thus, Lemma 8.1 reduces the problem of finding ACMs to a convex optimization problem for many types of models. Nonetheless, this problem is still challenging to solve via common techniques, e.g., gradient descent, because computing $\nabla g(\mathbf{w})$ involves finding the supremum over a large set of possible repairs $\mathbf{X}^R$. We can reduce the search for finding the supremum to a significantly smaller subset of repairs.

Definition 4. *(Edge Repair) Assume each missing value* x_{ij} *in* $\mathbf{X}$ *is bounded by an interval such that* $x_{ij}^{min} \leq x_{ij} \leq x_{ij}^{max}$. *A repair* $\mathbf{X}^e$ *is an edge repair if* $x_{ij}^e = x_{ij}^{min}$ *or* x_{ij}^{min} *for all missing values* x_{ij}. $\mathbf{X}^E$ *denotes the set of all possible edge repairs* $\mathbf{X}^e$.

Theorem 8.2. *For linear regression and SVM, we have*

$$g(\mathbf{w}') = \sup_{\mathbf{X}^e \in \mathbf{X}^E} h(\mathbf{w}', \mathbf{X}^e)$$

when the intervals for all missing values are $[-\infty, +\infty]$.

Based on Theorem 8.2, we can compute $g(\mathbf{w}')$ by finding the supremum of $h(\mathbf{w}', \mathbf{X}^r)$ only from edge repairs. In practice, the edge repairs associated with infinite intervals can be approximated by sufficiently wide intervals. This approach is efficient for datasets with a relatively small number of missing values. However, it may take long for datasets with many missing values because the number of edge repairs is $2^{n(MV)}$ where $n(MV)$ is the number of missing values in $\mathbf{X}$.

To accelerate finding ACMs for linear regression and SVM, Algorithm 4 randomly samples edge repairs and estimates the supremum of $h(\mathbf{w}', \mathbf{X}^r)$. This estimation is reasonable when the number of samples s is large. The algorithm's time complexity is $O(k \cdot d \cdot n \cdot s)$, where k stands for the number of iterations in gradient descent.

Algorithm 4 Learning ACM

$s \leftarrow$ the number of edge repairs to sample
$e \leftarrow$ user-defined threshold for approximate optimality
$\mathbf{X}^E_{sample} \leftarrow random.sample(\mathbf{X}^E, s)$
▷ randomly add s edge repairs to the sample set
for $\mathbf{X}^e \in \mathbf{X}^E_{sample}$ **do**
  $h(\mathbf{w}', \mathbf{X}^e) \leftarrow L(f(\mathbf{X}^e, \mathbf{w}'), \mathbf{y}) - \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{X}^e, \mathbf{w}), \mathbf{y})$
end for
$\mathbf{w}^{\approx} \leftarrow arg\min_{\mathbf{w} \in \mathcal{W}} \sup_{\mathbf{X}^e \in \mathbf{X}^E_{sample}} h(\mathbf{w}', \mathbf{X}^e)$
▷ This optimization is solved by existing algorithms
if $g(\mathbf{w}^{\approx}) \leq e$ **then**
  return $w^{\approx}$
else
  return "Approximately certain models do not exist"
end if

Table 2. Details of Real World Dataset containing missing values

Data Set	Task	Features	Training Examples	Missing Factor
Breast Cancer	Classification	10	559	1.97%
Intel-Sensor	Classification	11	1850945	4.05%
NFL	Regression	34	34302	9.04%
Water-Potability	Classification	9	2620	39.00%
Online Education	Classification	36	7026	35.48%
COVID	Regression	188	60229	53.67%
Air-Quality	Regression	12	7192	90.99%
Communities	Regression	1954	1595	93.67%

8.3 ACMs for Regression With Guarantees

For linear regression, if some conditions hold in the dataset, we can decompose the computation of the supremum for $h(\mathbf{w}', \mathbf{X}^e)$ in Theorem 8.2 to each example and compute ACMs in linear time.

Theorem 8.3. *For linear regression, if*

$$\forall i \in [1, ..., n], \forall \mathbf{x}_i^e \neq \mathbf{x}_i^{e*}, L(f(\mathbf{x}_i^{e*}, \mathbf{w}'), \mathbf{y}_i) - L(f(\mathbf{x}_i^e, \mathbf{w}'), \mathbf{y}_i) \geq \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{x}_{ic}, \mathbf{w}), \mathbf{y})$$

where x_{ic} is created by ignoring features with missing values in x_i, then

$$g(\mathbf{w}') = L(f(\mathbf{X}^{e*}, \mathbf{w}'), \mathbf{y}) - \min_{\mathbf{w} \in \mathcal{W}} L(f(\mathbf{X}^{e*}, \mathbf{w}), \mathbf{y})$$

where

$$\forall i \in [1, ..., n], \mathbf{x}_i^{e*} = arg \max_{\mathbf{x}_i^e \in \mathbf{x}_i^E} (\mathbf{w}'^T \mathbf{x}_i^e - y_i)^2$$

and $\mathbf{x}_i^E$ is the set of edge repairs for x_i.

Because training examples are independent, $\mathbf{X}^{e*}$ maximizes the overall training loss if and only if each training example in $\mathbf{X}^{e*}$ maximizes the squared error for the example. Further, when the latter condition in the theorem holds, $h(\mathbf{w}', \mathbf{X}^{e*})$ is also the supremum of $h(\mathbf{w}', \mathbf{X}^e)$. It is because this condition ensures that the training loss term is absolutely dominant in $h(\mathbf{w}', \mathbf{X}^e)$. This allows us to find the supremum edge repair for each training example individually.

Algorithm 5 uses this result to efficiently compute ACMs for linear regression. It uses the common gradient descent algorithm as $g(\mathbf{w})$ is convex. Its time complexity is $O(k \cdot d \cdot n)$. The latter condition in Theorem 8.3 is checked in linear time.

ACM for kernel SVM. Many properties in linear regression and linear SVM, such as the linearity that is used to prove Theorem 8.2, do not hold for kernel SVM. Therefore, it is very challenging to efficiently compute $g(\mathbf{w}')$ and check ACM for kernel SVM. We plan to put this line of research as the future work.

9 EXPERIMENTAL EVALUATION

We conduct experiments on a diverse set of real-world datasets and compare our algorithms with two natural baselines, a KNN imputation method, a deep learning-based imputation algorithm, and a benchmark method, ActiveClean. Our findings illustrate substantial savings in data cleaning costs and program running times when certain and approximately certain models exist. Moreover, our

Algorithm 5 Learning ACMs for Linear Regression

$\mathbf{w}^{(0)} \leftarrow \mathbf{w}^{init}$
$t \leftarrow 0$
$n \leftarrow$ the number of training examples
$e \leftarrow$ user-defined threshold for approximate optimality
while $||\nabla g(\mathbf{w}^{(t)})|| > \epsilon$ **do**
  $t \leftarrow t + 1$
  for $i = 1, 2, ..., n$ **do**
    $\mathbf{x}_i^{e^*} \leftarrow \text{argmax}_{\mathbf{x}_i^e} ||\mathbf{w}^{(t-1)T}\mathbf{x}_i^e - y_i||_2^2$
  end for
  $\nabla g(\mathbf{w}^{(t-1)}) \leftarrow \nabla L(f(\mathbf{X}^{e*}, \mathbf{w}^{(t-1)})$
  $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla g(\mathbf{w}^{(t-1)})$
end while
if $g(\mathbf{w}^{(t)}) \leq e$ **then**
  return $w^{(t)}$
else
  return "Approximately certain models do not exist"
end if

Table 3. Linear SVM: Comparing Performance on Randomly Corrupted Real-World Datasets

(a) Data Sets Where Certain Models Exist

Data Set	MF	Examples Cleaned		Time (Sec)						Accuracy (%)				
		AC	MI/KI/DI	AC	CM	DI	KI	MI	NI	AC	DI	KI	MI	CM/NI
Gisette	0.1%	6.07	14	17.48	2.14	N/A	4.21	1.43	**0.90**	**97.94**	N/A	96.60	97.60	97.40
	1%	60.40	135	20.32	2.18	N/A	17.88	1.42	**0.88**	**97.89**	N/A	97.60	97.60	97.33
Malware	0.1%	1.0	**2**	4.56	0.73	N/A	0.96	0.74	**0.34**	96.09	N/A	**96.24**	**96.24**	**96.24**
	1%	14.3	20	3.93	0.86	N/A	1.56	0.73	**0.44**	96.10	N/A	**96.24**	**96.24**	**96.24**
	5%	44.93	200	3.17	0.78	N/A	3.99	0.72	**0.36**	96.54	N/A	96.24	96.24	**96.57**
Tuandromd	0.1%	3	5	3.71	0.04	62.17	0.17	0.05	**0.04**	98.73	**98.86**	98.09	98.09	98.76
	1%	30.9	45	3.72	**0.03**	78.81	0.29	0.04	**0.03**	98.88	**98.92**	98.80	98.76	98.58

(b) Data Set Where Certain Models Do Not Exist but Approximately Certain Models Exist

Data Set	MF	Examples Cleaned		Time (Sec)							Accuracy (%)					
		AC	MI/KI/DI	AC	CM	ACM	DI	KI	MI	NI	AC	DI	KI	MI	NI	ACM
Gisette	5%	248.93	675	17.62	1.82	10.37	N/A	53.97	1.45	**0.71**	97.03	N/A	**97.60**	97.43	97.30	97.35
	10%	393.33	1350	1.73	1.73	12.94	N/A	97.01	1.40	**0.65**	**99.93**	N/A	97.00	97.53	97.07	97.60
Tuandromd	5%	91.40	223	2.41	**0.04**	4.31	74.21	0.44	0.05	**0.04**	98.08	**98.76**	98.36	98.36	98.21	98.18

(c) Data Set Where Neither Certain Nor Approximately Certain Models Exist

Data Set	MF	Examples Cleaned		Time (Sec)							Accuracy (%)				
		AC	MI/KI/DI	AC	**CM**	ACM	DI	KI	MI	NI	AC	DI	KI	MI	NI
Malware	10%	66.0	200	1.97	0.70	7.16	N/A	6.78	0.73	**0.33**	88.05	N/A	**96.24**	**96.24**	83.95
Tuandromd	10%	121.33	446	1.64	**0.04**	4.98	82.45	0.74	0.05	**0.04**	97.36	**98.76**	**98.76**	98.20	98.54

study highlights the minimal computational overhead incurred by our algorithms when verifying certain and approximately certain model conditions, even when these models do not exist.

9.1 Experimental Setup

9.1.1 Hardware and Platform. We experiment on a configuration with two tasks, each utilizing two CPUs, and running on a cluster partition equipped with one 11GB GPU. The underlying hardware consists of Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz machines.

Table 4. p-SVM: Comparing Performance on Randomly Corrupted Real-World Datasets

(a) Data Sets Where Certain Models Exist

Data Set	MF	**Examples Cleaned**	Time (Sec)					Accuracy (%)			
		MI/KI/DI	CM	DI	KI	MI	NI	DI	KI	MI	CM/NI
Gisette	0.1%	14	55.69	N/A	56.61	56.01	**55.27**	N/A	**96.70**	**96.70**	**96.70**
Malware	0.1%	2	4.98	N/A	4.95	5.01	**4.66**	N/A	**92.98**	**92.98**	**92.98**
Tuandromd	0.1%	5	**0.20**	49.24	0.21	0.21	**0.20**	98.54	**98.54**	**98.54**	**98.54**

(b) Data Set Where Certain Models Do Not Exist

Data Set	MF	**Examples Cleaned**	Time (Sec)					Accuracy (%)			
		MI/KI/DI	**CM**	DI	KI	MI	NI	DI	KI	MI	**CM/NI**
Gisette	0.1%	14	**43.79**	N/A	52.33	51.69	51.32	N/A	**96.80**	96.70	96.70
	1%	135	**42.84**	N/A	60.04	51.20	50.04	N/A	**96.70**	**96.70**	**96.70**
	5%	675	**36.15**	N/A	85.98	42.95	37.12	N/A	96.75	96.70	**96.80**
	10%	1350	**36.94**	N/A	115.80	45.27	41.63	N/A	96.70	96.70	**97.00**
Malware	0.1%	**2**	**4.35**	N/A	5.64	5.18	4.35	N/A	**92.87**	**92.87**	92.75
	1%	20	**4.13**	N/A	6.76	4.80	4.53	N/A	**92.98**	**92.98**	**92.98**
	5%	100	4.84	N/A	7.90	4.81	**4.14**	N/A	**92.98**	**92.98**	**92.98**
	10%	200	**4.60**	N/A	9.79	5.81	4.84	N/A	**92.98**	92.74	**92.98**
Tuandromd	0.1%	5	0.20	70.22	0.21	0.21	**0.19**	**98.54**	**98.54**	**98.54**	**98.54**
	1%	45	**0.18**	44.81	0.30	0.21	0.19	**98.54**	**98.54**	**98.54**	**98.54**
	5%	223	0.18	42.39	0.54	0.20	**0.17**	**98.54**	**98.54**	**98.54**	98.31
	10%	446	0.19	42.45	0.75	0.21	**0.16**	98.54	**98.79**	98.54	98.54

9.1.2 Real-world Datasets with randomly generated missing values. In our certain model experiments with linear SVM, polynomial SVM, and DNN (arccosine SVM) we utilize three real-world datasets. These datasets originally do not contain any missing values, but we introduce corruption by randomly injecting missing values at missing factors of 0.1%, 1%, 5%, and 10%. Where **missing factor (MF) represents the ratio of incomplete examples (examples with at least 1 missing value) to the total number of examples**. It is important to note that certain models do not exist in all versions of the corrupted datasets. Therefore, we present experimental results for both scenarios, *when certain models exist and when they do not*. For each dataset and each missing factor, we present the average results based on three randomly corrupted versions of the dataset in which certain models exist. This is to reduce the variability in algorithm performance resulting from the randomness of missing value injection.

Malware Dataset. The Malware dataset aims to distinguish between malware and benign software through the analysis of JAR files [27]. It comprises 6825 features and 1996 training examples.

Gisette Dataset. The Gisette dataset addresses the problem of handwritten digit recognition, with a specific focus on distinguishing between the easily confused digits 4 and 9 [14]. It consists of 13500 training examples and 5000 features.

TUANDROMD Dataset. The TUANDROMD dataset is designed for the detection of Android malware software in contrast to benign or "goodware" applications [4]. It comprises 4464 training examples and incorporates 241 distinct features.

9.1.3 Real-world Datasets originally containing missing values. We also conduct experiments on 8 real-world datasets originally containing missing values. Our selection includes datasets from diverse domains and *missing factors* (Section 9.1.2). Table 2 presents a summary of the datasets. For preprocessing the dataset if the label is missing we drop all corresponding examples and utilize *sklearns* OneHotEncoder to featurize the categorical attributes.

Intel Sensor. This dataset contains temperature, humidity, and light readings collected from sensors deployed in the Intel Berkeley Research lab [3, 17]. The classification task is to predict whether the readings came from a particular sensor (sensor 49).

Water Potability . The dataset contains information about the properties and substances (sulfate, pH) in freshwater sources, the classification task is to predict if the water is potable or not [15].

COVID . This U.S. Department of Health and Human Services dataset provides data for hospital utilization dating back to January 1, 2020 [1]. The regression task is to predict the number of hospitals anticipating critical staffing shortages.

Air Quality. The dataset contains instances of hourly averaged responses from an array of chemical sensors embedded in an Air Quality Chemical Multisensor Device [32]. Given air-composition measurements, the regression task is to predict hourly Temperature.

NFL. This dataset contains play-by-play logs from US Football games. Given a play, the regression objective is to predict the score difference between the two teams [13]. We use a numeric version of this dataset since encoding string-valued attributes inflates the feature dimension by 80 times.

Breast Cancer. The dataset contains information about characteristics of potential cancerous tissue and the classification task is to predict if it is benign or malignant [34].

Online Education. This dataset contains responses to a survey on online education. The classification objective is to predict whether students prefer cellphones or laptops for online courses [31].

Communities-Crimes. The data contains socio-economic and crime data from the US Census and FBI. The regression task is to predict the total number of violent crimes per 100K population [26].

9.1.4 Algorithms for Comparison. In our experiments, we include two natural imputation baselines, a deep learning-based imputation algorithm, and a benchmark algorithm for comparison.
Active Clean(AC): ActiveClean [18, 19] aims at minimizing the number of repaired examples to achieve an accurate model. We use ActiveClean for linear regression and linear SVM, but not for kernel SVM because ActiveClean's implementation relies on sklearn's SGDClassifier module, which does not support non-linear models such as kernel SVM [19]. Simply switching to sklearn's Support Vector Classification (SVC) module cannot resolve the issue since SVC does not support 'partial fit', an essential function in ActiveClean.
KNN-Imputer(KI): This method predicts the values of missing items based on observed examples using a KNN classifier [23, 29].
Deep-learning based Imputation (DI): We utilize MIWAE [23] as a sophisticated state-of-the-art imputation algorithm for comparison. This approach uses deep latent variable models to predict the values of missing data items based on the value of observed examples. Specifically, MIWAE adapts the objective of importance-weighted autoencoders [5] and maximizes a potentially tight lower bound of the log-likelihood of the observed data.
Mean Imputation(MI): MI is a widely used method for handling missing values in practice. Each missing feature value is imputed with the mean value of that feature [23].
No Imputation(NI): NI naively drops all missing values and trains the model on the complete training set [18]. *When certain model exists* the model trained with NI is equivalent to Certain Model algorithms.

9.1.5 Metrics. We evaluate all algorithms on a held-out test set with complete examples. We use accuracy as a metric for classification tasks. Accuracy reflects the percentage of total correct

class predictions therefore *higher accuracy is preferred*. For regression tasks, we use mean squared error (MSE). MSE measures the average squared difference between actual and predicted values. Therefore, a *lower MSE is preferred*. We also study the algorithms' data-cleaning efforts in terms of program execution time and the number of examples cleaned.

9.2 Results on Real-world Datasets with Random Corruption

In this section, we present results for all scenarios, *when certain models exist, when certain models do not exist but approximately certain models exist, and when neither exists* for 3 datasets with different degrees of random corruption (Section 9.1.2).

When certain models exist : We begin by focusing on scenarios where certain models are known to exist. Tables 3a and 4a present a performance comparison between certain model algorithms and baselines for linear SVM and p-SVM, respectively. We observe that certain models exist when the missing factor is small. This is because certain SVM models require that incomplete examples should not be support vectors in any repair (Section 5). When the missing factor is small, the number of incomplete examples is also small. Therefore, the likelihood of such examples being support vectors is also small. Since certain models exist, certain model algorithms by definition (Section 3) clean zero missing data. In contrast, baseline methods spend substantial effort on data cleaning, represented by *examples cleaned* column in Tables 3a and 4a. In terms of program execution time, certain model (CM) algorithms are slower than simple methods such as Mean Imputation (MI) and No Imputation (NI). However, MI and NI are both heuristics without any guarantees of the optimality of the trained model. Moreover, to find the appropriate imputation method for a specific incomplete dataset, users may want to check missing data mechanisms, which often take longer time than implementing certain model algorithms. Also, MI *still requires a large number of imputations*. Compared with ActiveClean (AC) and the advanced imputation methods, Deep-learning based Imputer (DI) and KNN-Imputer (KI), certain model algorithms run much faster and also guarantee an optimal model. In the tables, DI has "N/A" results for some datasets in this section when it runs for more than one hour but fails to return a result. To emphasize, *when certain models exist, we do not need to check for approximately certain models since they exist by definition* (Section 8).

When certain models do not exist but approximately certain models exist: When certain models do not exist, we may resort to approximately certain models (ACM). Table 3b shows the datasets where certain models do not exist due to the strict conditions, but approximately certain models exist and clean zero missing data. The prediction accuracy of approximately certain models is very close to the results from all baseline methods. This is because when approximately certain models exist, their approximate optimality is guaranteed. In terms of program execution time, approximately certain model algorithms run faster than DI and KI, but slower than AC, MI, and NI. However still in this scenario, checking and learning approximately certain models saves imputation costs with minimal compromise on the model's accuracy.

When neither certain nor approximately certain models exist: Sometimes, neither certain nor approximately certain models may exist, as shown in Table 3c for linear SVM with relatively large missing factor. Since approximately certain model algorithms are not available for kernel SVMs, in Tables 4b and 5 we present the datasets where certain models do not exist for polynomial SVM (p-SVM), and our DNN approximation with arccosine SVM, respectively. However, even if our algorithms do not find certain models for DNN, *certain models may still exist*. This is because the related theorem for DNN is *necessary but not sufficient* as described in Section 7. We also investigate certain model existence for RBF-SVM. We observe similar patterns and results to that of arccos-SVM. Due to the limited space, we exclude the results for RBF-SVM from the paper. In scenarios when neither a certain model nor an approximately certain model exists, checking for them incurs some computational overhead. Nonetheless, *paying for this overhead is worthwhile*

Table 5. DNN: Comparing Performance on Randomly Corrupted Real-World Datasets: CMs are not found

Data Set	MF	**Examples Cleaned**	Time (Sec)					Accuracy (%)			
		MI/KI/DI	**CM**	DI	KI	MI	NI	DI	KI	MI	**NI**
Gisette	0.1%	14	**31.27**	N/A	40.79	40.62	39.67	N/A	52.30	52.30	52.10
	1%	135	**35.46**	N/A	40.69	40.63	39.57	N/A	52.30	52.30	52.10
	5%	675	**33.28**	N/A	39.91	39.67	36.06	N/A	52.30	52.30	52.00
	10%	1350	**29.98**	N/A	39.94	39.84	34.10	N/A	52.30	52.30	51.70
Malware	0.1%	**2**	**3.27**	N/A	6.02	6.06	5.93	N/A	48.37	48.37	48.37
	1%	20	**3.15**	N/A	5.99	6.78	5.76	N/A	48.37	48.37	48.37
	5%	100	**4.03**	N/A	5.98	6.00	5.39	N/A	48.37	48.37	48.37
	10%	200	**4.12**	N/A	5.98	5.96	4.92	N/A	48.37	48.37	34.09
Tuandromd	0.1%	5	**0.22**	47.56	0.21	0.28	0.20	61.81	61.81	61.81	61.81
	1%	45	**0.25**	69.24	0.21	0.20	0.20	61.81	61.81	61.81	61.81
	5%	223	**0.20**	67.94	0.22	0.21	0.18	61.81	61.81	61.81	61.81
	10%	446	**0.18**	70.13	0.25	0.26	0.17	61.81	61.81	61.81	66.52

Table 6. Linear Regression: Performance Comparison on Real-World Datasets with Missing Values

(a) Data Sets Where Certain Models Exist

Data Set	**Examples Cleaned**		Time (Sec)						MSE				
	AC	MI/KI/DI	AC	CM	DI	KI	MI	NI	AC	DI	KI	MI	CM/NI
NFL	12.0	3101	49.91	7.11	394.18	12.96	0.14	**0.13**	0.02	**0.00**	**0.00**	**0.00**	**0.00**
COVID	33.6	32325	100.59	438.79	1944.10	587.87	0.68	**0.28**	2.07	**0.00**	**0.00**	**0.00**	**0.00**

(b) Data Set Where Certain Models Do Not Exist but Approximately Certain Models Exist

Data Set	**Examples Cleaned**		Time (Sec)								MSE					
	AC	MI/KI/DI	AC	CM	ACM5	ACM6	DI	KI	MI	NI	AC	DI	KI	MI	NI	ACM
Communities	319.6	1494	2.10	1.45	4.15	3.74	4088.46	15.82	1.39	**0.08**	0.06	0.35	0.63	1.30	2.30	**0.03**

(c) Data Set Where Neither Certain Nor Approximately Certain Models Exist

Data Set	**Examples Cleaned**		Time (Sec)								MSE				
	AC	MI/KI/DI	AC	CM	ACM5	ACM6	DI	KI	MI	NI	AC	DI	KI	MI	NI
Air-Quality	48.80	6544	0.87	**0.01**	4.62	N/A	111.08	3.67	0.02	**0.01**	28.22	2.11	3.05	**1.07**	3.47

Table 7. Linear SVM: Performance Comparison on Real-World Datasets with Missing Values

(a) Data Set Where Certain Models Do Not Exist but Approximately Certain Models Exist

Data Set	**Examples Cleaned**		Time (Sec)							Accuracy (%)					
	AC	MI/KI/DI	AC	CM	ACM	DI	KI	MI	NI	AC	DI	KI	MI	NI	ACM
Intel	30.0	75080	355.58	276.01	2428.76	N/A	13775.93	275.49	**273.24**	98.80	N/A	**98.90**	97.50	98.39	98.43

(b) Data Set Where Neither Certain Nor Approximately Certain Models Exist

Data Set	**Examples Cleaned**		Time (Sec)							Accuracy(%)				
	AC	MI/KI/DI	AC	CM	ACM	DI	KI	MI	NI	AC	DI	KI	MI	NI
Water Potability	29.0	1022	0.34	**0.01**	0.69	56.14	0.30	0.04	**0.01**	49.15	**56.33**	39.95	39.70	41.89
Online Education	16.4	2493	3.88	0.03	5.03	88.87	2.09	0.14	**0.02**	63.06	**63.85**	62.85	61.37	36.33
Breast Cancer	9.8	14	0.06	0.01	0.15	7.37	0.33	0.01	**0.00**	50.44	**65.94**	**65.94**	**65.94**	34.05

for two reasons. First, substantial data-cleaning savings are realized when certain models exist (as we discuss in the previous paragraphs). Second, the time costs associated with checking certain models are minimal (confirmed by the small program running times in Tables 3c, 4b, and 5).

9.3 Results on Real-world Dataset with Inherent Missingness

In our certain model experiments with linear regression, linear SVM, and SVM with kernels, we utilize 8 real-world datasets (Section 9.1.3). These datasets originally contain missing values.

Table 8. p-SVM: Performance Comparison on Real-World Datasets with Missing Values-CMs do not exist

Data Set	**Number of Examples Cleaned**	Time (Sec)					Accuracy(%)			
	MI/KI/DI	CM	DI	KI	MI	NI	DI	KI	MI	NI
Intel Sensor	1022	3321.85	N/A	3498.51	3128.54	**3007.65**	N/A	53.78	**58.37**	51.98
Water Potability	1022	0.05	41.27	1.03	0.16	**0.06**	63.94	62.17	**62.96**	61.19
Online Education	354	**0.18**	57.13	0.98	0.22	0.20	97.13	95.52	**97.26**	93.29
Breast Cancer	11	**0.01**	42.35	0.25	0.01	0.01	**71.24**	70.86	70.71	69.63

Table 9. DNN: Performance Comparison on Real-World Datasets with Missing Values-CMs are not found

Data Set	**Number of Examples Cleaned**	Time (Sec)					Accuracy(%)			
	MI/KI/DI	CM	DI	KI	MI	NI	DI	KI	MI	NI
Online Education	354	**0.16**	51.37	0.79	0.22	0.20	**42.75**	40.96	41.04	41.06
Breast Cancer	11	0.02	48.96	0.02	**0.01**	**0.01**	65.20	64.26	**67.86**	66.67

When certain models exist: We present the result for the first scenario in Table 6a. Certain models exist for NFL and COVID datasets. By checking and learning certain models with zero imputation, we save substantial energy in data cleaning compared to all 4 baselines. This imputation cost saving also comes with guarantees on the optimal model, experimentally proved by almost the same performance between certain models and the models from baseline methods. There is one exception in the COVID dataset where ActiveClean has a regression error slightly different from other baselines. This may be because partial-fit is used to proxy a complete-fit in ActiveClean's implementation [19], which in some cases may converge early but with errors. We further investigate the data scenarios that entail certain models and verify that features irrelevant to the label are the ones with missing values. For instance, the COVID dataset receives regular data updates from three different sources. We observe that the newly added features are the ones with missing values.
When certain models do not exist but approximately certain models exist: Table 6b and 7a present the results for linear regression and linear SVM, respectively. Two datasets (Communities and Intel-Sensor) do not have certain models but have approximately certain models. Approximately certain models result in similar MSE/accuracy compared to all baseline methods, supported by the theoretical guarantee from approximately certain models (Section 8). In this scenario, checking and learning approximately certain models also eliminates the need for any form of data imputations. In terms of program execution time, we observe similar patterns as the results of randomly corrupted datasets. To understand the *influence of data characteristics on certain model existence*, we study the results for both certain models (CM and ACM). We find that certain and approximately certain models are more likely to exist in regression tasks when the number of features is large (e.g., Communities and COVID), and in classification tasks when the number of examples is large (e.g., Intel-Sensor). This is because the uncertainty from incomplete features and examples is diluted in model training when the number of features and examples is large.
When neither certain nor approximately certain models exist: Tables 6c, 7b, 8, and 9 show the cases where neither a CM nor an ACM exists (or is not found) for linear regression, linear SVM, p-SVM, and DNN, respectively. We report DNN's result (Table 9) only on two out of four classification datasets because the DNN's theorem in Section 7 only applies to datasets where each example has at most one missing value. For p-SVM and DNN, the prediction accuracy is almost the same from different baseline imputations, which often empirically suggests the existence of certain or approximately certain models. However, certain models do not exist (or are not found). To explain, when certain models do not exist, different imputations may lead to different models. Nonetheless, different models sometimes can still make identical predictions on the testing set. e. In terms of program execution time, checking certain and approximately certain models is worthwhile even if we do not find any upon checking, based on the same reasons discussed in Section 9.2.

10 RELATED WORK

Stochastic and Robust Optimization. Researchers have proposed stochastic optimization to find a model by optimizing the *expected loss function over the probability distributions of missing data items* in the training examples [11]. This approach avoids imputing missing values by redefining the loss function to include the uncertainty due to missing values in the training data. Similarly, in robust optimization, researchers minimize the loss function of a model for the worst-case repair to an incomplete dataset, i.e., the repair that brings the highest training loss, given distributions of the missing values. However, the distributions of missing data items are not often available. Thus, users may spend significant time and effort to discover or train these distributions. Additionally, for a given type of model, users must solve various and possibly challenging optimization problems for many possible (combinations of) distributions of missing values. In our approach, users do not need to find the probability distribution of the missing data. Moreover, our algorithm for each type of model generalizes for all types and distributions of missing values in the training data.

Subset Selection over Incomplete Data. To save data cleaning costs, researchers propose to select a representative subset of training data and impute the missing values in the subset [6, 33]. Then, a model is trained with the clean version of this subset. This approach still cleans data items. One still needs to spend time constructing a model to select a proper subset. Also, the trained model is often not the same as the model trained with the whole dataset.

ML Poisoning Attacks. Researchers have proposed methods to build ML models that are robust to malicious modifications of training data to induce unwanted behavior in the model [9]. We, however, focus on robustness against missing values in the data.

11 CONCLUSION AND FUTURE WORK

In this paper, we present the conditions where data repair is not needed for training optimal and approximately optimal models over incomplete data, i.e., certain or approximately certain models exist. We also offer efficient algorithms for checking and learning certain and approximately certain models for linear regression, linear SVM, kernel SVMs, and DNN. Our experiments with real-world datasets demonstrate significant cost savings in data cleaning compared to five popular benchmark methods, without introducing significant overhead to the running time.

11.1 Suggesting a subset of incomplete examples to impute

Sometimes certain models do not exist and users cannot get accurate models without imputation. Nonetheless, Theorem 5.4 can suggest a subset of incomplete examples to impute. Specifically, Algorithm 2 identifies a subset of incomplete examples that violate certain model conditions. Users can choose to impute the examples in this subset instead of all missing values to get a certain model. Cleaning costs are saved by partial cleaning, although one may not need to clean all examples in this subset to achieve a certain model. Interesting future work is how to efficiently suggest the minimal number of examples to impute to get a certain model.

11.2 Handling dirty data beyond missing values

Although this paper focuses on missing values, CM and ACM methods can be applied to other types of dirty data that require imputations to fix. This is because CM and ACM algorithms check if a model is optimal over all possible imputations. For example, a value x_{ij} is an outlier if it substantially deviates from the distribution of the corresponding feature. An outlier can be fixed by imputing it. Certain types of data inconsistency also require imputations such as those caused by constraint violations over individual tuples. When CM or ACM exists, cleaning outliers and certain types of inconsistency is unnecessary. However, the applicability of our methods to complex data

inconsistencies, such as violations of functional dependencies across multiple attributes, remains unclear. Addressing these challenges is an important direction for future research.

ACKNOWLEDGMENTS

This research was partially funded through NSF grant CNS-1941892 and the Industry-University Cooperative Research Center on Pervasive Personalized Intelligence.

REFERENCES

[1] 2023. COVID-19 Reported Patient Impact and Hospital Capacity. https://catalog.data.gov/dataset/covid-19-reported-patient-impact-and-hospital-capacity-by-state-timeseries-cf58c. Accessed on 01-01-2024.

[2] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan. 2021. Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions. *Journal of Big Data* 8, 1 (2021), 53. https://doi.org/10.1186/s40537-021-00444-8

[3] Peter Bodik, Wei Hong, Carlos Guestrin, Sam Madden, Mark Paskin, and Romain Thibaux. 2004. Intel Berkley Research Lab Data. https://db.csail.mit.edu/labdata/labdata.html

[4] Parthajit Borah, DK Bhattacharyya, and JK Kalita. 2020. Malware Dataset Generation and Evaluation. In *2020 IEEE 4th Conference on Information and Communication Technology (CICT)*. IEEE, 1–6.

[5] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. 2016. Importance Weighted Autoencoders. arXiv:1509.00519 [cs.LG]

[6] Chengliang Chai, Jiabin Liu, Nan Tang, Ju Fan, Dongjing Miao, Jiayi Wang, Yuyu Luo, and Guoliang Li. 2023. GoodCore: Data-effective and Data-efficient Machine Learning through Coreset Selection over Incomplete Data. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[7] Youngmin Cho and Lawrence Saul. 2009. Kernel methods for deep learning. *Advances in neural information processing systems* 22 (2009).

[8] Youngmin Cho and Lawrence K Saul. 2011. Analysis and extension of arc-cosine kernels for large margin classification. *arXiv preprint arXiv:1112.3712* (2011).

[9] Samuel Drews, Aws Albarghouthi, and Loris D'Antoni. 2020. Proving data-poisoning robustness in decision trees. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 1083–1097. https://doi.org/10.1145/3385412.3385975

[10] Austen Z. Fan and Paraschos Koutris. 2022. Certifiable Robustness for Nearest Neighbor Classifiers. In *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference) (LIPIcs, Vol. 220)*, Dan Olteanu and Nils Vortmeier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:20. https://doi.org/10.4230/LIPICS.ICDT.2022.6

[11] Ravi Ganti and Rebecca M Willett. 2015. Sparse Linear regression with missing data. *arXiv preprint arXiv:1503.08348* (2015).

[12] Claudio Gentile and Manfred K. K Warmuth. 1998. Linear Hinge Loss and Average Margin. In *Advances in Neural Information Processing Systems*, M. Kearns, S. Solla, and D. Cohn (Eds.), Vol. 11. MIT Press.

[13] Max Horowitz. 2015. Detailed NFL Play-by-Play Data 2015. Kaggle. https://www.kaggle.com/datasets/maxhorowitz/nflplaybyplay2015

[14] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, Gideon Dror. 2003. Gisette. https://doi.org/10.24432/C5HP5B

[15] Aditya Kadiwal. 2021. Water Potability. Kaggle. https://www.kaggle.com/datasets/adityakadiwal/water-potability

[16] Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. 2020. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117* (2020).

[17] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. 2017. BoostClean: Automated Error Detection and Repair for Machine Learning. arXiv:1711.01299 [cs.DB]

[18] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.

[19] Krishnan, Sanjay and Wang, Jiannan and Wu, Eugene and Franklin, Michael J and Goldberg, Ken. 2018. Cleaning for Data Science. https://activeclean.github.io/

[20] Marine Le Morvan, Julie Josse, Erwan Scornet, and Gael Varoquaux. 2021. What's a good imputation to predict with missing values?. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 11530–11540. https://proceedings.neurips.cc/paper_files/paper/2021/file/5fe8fdc79ce292c39c5f209d734b7206-Paper.pdf

[21] R.J.A. Little and D.B. Rubin. 2002. *Statistical analysis with missing data*. Wiley. http://books.google.com/books?id=aYPwAAAAMAAJ
[22] Tongyu Liu, Ju Fan, Yinqing Luo, Nan Tang, Guoliang Li, and Xiaoyong Du. 2021. Adaptive Data Augmentation for Supervised Learning over Missing Data. *Proc. VLDB Endow.* 14, 7 (mar 2021), 1202–1214. https://doi.org/10.14778/3450980.3450989
[23] Pierre-Alexandre Mattei and Jes Frellsen. 2019. MIWAE: Deep Generative Modelling and Imputation of Incomplete Data Sets. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 4413–4423. https://proceedings.mlr.press/v97/mattei19a.html
[24] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From Cleaning before ML to Cleaning for ML. *IEEE Data Eng. Bull.* 44, 1 (2021), 24–41.
[25] Jose Picado, John Davis, Arash Termehchy, and Ga Young Lee. 2020. Learning over dirty data without cleaning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1301–1316.
[26] Michael Redmond. 2009. Communities and Crime. UCI Machine Learning Repository.
[27] Ricardo P Pinheiro, Sidney M. L. Lima, Sérgio M. M. Fernandes, E. D. Q. Albuquerque, S. Medeiros, Danilo Souza, T. Monteiro, Petrônio Lopes, Rafael Lima, Jemerson Oliveira, Sthéfano Silva. 2019. REJAFADA. https://doi.org/10.24432/C5HG8D
[28] DONALD B. RUBIN. 1976. Inference and missing data. *Biometrika* 63, 3 (12 1976), 581–592. https://doi.org/10.1093/biomet/63.3.581 arXiv:https://academic.oup.com/biomet/article-pdf/63/3/581/756166/63-3-581.pdf
[29] Olga Troyanskaya, Mike Cantor, Gavin Sherlock, Trevor Hastie, Rob Tibshirani, David Botstein, and Russ Altman. 2001. Missing Value Estimation Methods for DNA Microarrays. *Bioinformatics* 17 (07 2001), 520–525. https://doi.org/10.1093/bioinformatics/17.6.520
[30] Stef Van Buuren. 2018. *Flexible imputation of missing data*. CRC press.
[31] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198
[32] Saverio Vito. 2016. Air Quality. UCI Machine Learning Repository.
[33] Yining Wang and Aarti Singh. 2015. Column subset selection with missing data via active sampling. In *Artificial Intelligence and Statistics*. PMLR, 1033–1041.
[34] William Wolberg. 1992. Breast Cancer Wisconsin (Original). UCI Machine Learning Repository.
[35] Cheng Zhen, Nischal Aryal, Arash Termehchy, and Amandeep Singh Chabada. 2024. Certain and Approximately Certain Models for Statistical Learning. *arXiv preprint arXiv:2402.17926* (2024).

Received October 2023; revised January 2024; accepted February 2024