Data Validation and Selection for Modern Machine Learning

by

Zifan Liu

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2024

Date of final oral examination: 04/03/2024

To my family.

ACKNOWLEDGMENTS

The journey through my PhD has been full of immense growth and learning, with both challenges and triumphs. This journey would not have been possible without the support and guidance of many incredible people who have contributed to my experience and success.

First and foremost, I extend my deepest gratitude to my advisor Theodoros Rekatsinas. Your expert guidance, patient mentorship, and unwavering support have been pivotal throughout my research. You have been not only a mentor but also an inspiration, and for that, I am profoundly grateful.

I would also like to thank the other members of my dissertation committee: Paraschos Koutris, Jignesh Patel, Shivaram Venkataraman, and Dimitris Papailiopoulos. Your insightful feedback and rigorous evaluations have significantly enhanced both my dissertation and my skills as a researcher. Your dedication to upholding high academic standards is truly appreciated.

A special thank you goes to my internship hosts Evan Rosen, Paul Suganthan, and Shaleen Deep. The opportunities you provided allowed me to apply my academic knowledge in real-world settings, enriching my research and broadening my perspective. These experiences were crucial in shaping my approach to my work.

My heartfelt thanks to the coauthors of my publications: Zhechun Zhou, Jongho Park, Christos Tzamos, Anna Fariha, Fotis Psallidas, Ashish Tiwari, and Avrilia Floratou. Collaborating with you has been an enriching experience. I am grateful for the shared efforts and the collective spirit that brought our ideas and work to fruition. Your partnership in research has been invaluable.

To my labmates Ankur Goswami, Akshata Bhat, Yunjia Zhang, Jason Mohoney, and Roger Waleffe, thank you for your camaraderie and support. Our discussions, brainstorming sessions, and the occasional much-needed distractions have made the lab a vibrant and enriching place to work. I could not have asked for a better group of colleagues.

I owe a great deal of thanks to my friends: Zhihan Guo, Kan Wu, Zhenmei Shi, Yingxin Jia, Yucheng Yang, Elisa Ou, and many others, who provided support, laughter, and a necessary escape from the world of academia. Your encouragement and presence have been a source of strength and joy throughout this demanding journey.

To my family, your unconditional love, and support have been my foundation. You have always believed in me, even when I doubted myself. Thank you for your endless patience and encouragement.

Lastly, I must acknowledge my cat Zelda Liu, whose quiet companionship during late-night writing sessions offered much-needed moments of comfort and amusement.

This thesis not only reflects my work but the support and contributions of each one of you. I am eternally grateful to have had such a wonderful group of people by my side. Thank you for being part of my PhD journey.

CONTENTS

ABSTRACT

Machine learning (ML) has revolutionized a wide range of fields with its capacity to learn from data and make informed decisions. Recognizing the critical role of well-curated data in the advancement of modern ML, the data-centric ML community emphasizes the importance of careful data preparation and strategic selection to ensure both data quality and their relevance to target tasks. However, the prevailing approach for data validation and selection employed in practice remains largely manual or ad-hoc, while automated methods often fall short in either effectiveness or efficiency, thus limiting their practical application. In this dissertation, we revisit the current methodologies for automated data validation and selection, aiming to propose new techniques with improved performance and practicality.

In the first part of this dissertation, we study the verification and discovery of denial constraints, a general formalism that can express a wide range of quality rules for tabular data. Verification entails detecting whether a given denial constraint holds on a specific dataset, while discovery focuses on the automated mining of valid constraints. The current state-of-the-art methods for denial constraint verification and discovery are inefficient on large-scale datasets due to their quadratic complexity relative to the dataset size. In addition, existing works on denial constraint discovery rely on a time-consuming blocking phase of building intermediate data structures, further limiting their practicality. To address the limitations of prior works, we make a dual contribution. First, we introduce a novel verification algorithm that demonstrates near-linear complexity relative to dataset size by connecting denial constraint verification to orthogonal range search, showing a theoretical improvement over prior works. Second, we present an anytime algorithm for denial constraint discovery by combining our verification algorithm with lattice searches, eliminating the need for the blocking structure-building phase in existing solutions. Our verification algorithm achieves up to 84×

faster compared to state-of-the-art approaches. In addition, our discovery algorithm is able to start providing valid constraints within the initial 10 minutes of execution, while existing methods are blocked for over 48 hours.

In the second part, we focus on defending against data corruption in ML pipelines. Data corruption is an impediment to modern ML applications as they can severely bias the learned model and also lead to invalid inferences. Data corruption in practice can be highly diverse, ranging from random noise, systematic errors to adversarial attacks, which are often beyond the scope of standard error detection methods. We present a simple framework to safeguard against data corruption during both training and deployment of ML models over tabular data. In the training stage, our framework identifies and removes corrupted data points from the training data to avoid obtaining a biased model. In the deployment stage, our framework flags, in an online manner, corrupted query points to a trained ML model that due to noise will result in incorrect predictions. To detect corrupted data, we develop a self-supervised deep learning model for mixed-type tabular data. To minimize the burden of deployment, learning the model does not require any human-labeled data. Our framework is designed as a plugin that can increase the robustness of any ML pipeline. We show that our framework consistently safeguards against corrupted data during both training and deployment of various models ranging from SVMs to neural networks, beating a diverse array of competing methods that span from data quality validation models to robust outlier-detection models. In addition, to promote the understanding of the worst-case effects that data corruption can have on learning performance, we present an information-theoretic analysis of robust mean estimation under coordinate-level corruption. Our analysis shows that leveraging the dependencies between features is the key to accurate mean estimation for corrupted data.

In the last part, our attention turns to task-specific data selection. The goal is to select training data for specific tasks from a massive and heterogeneous pool of candidate data, guided by a small set of representative examples from the target task. We highlight two critical properties for the selected data: distribution alignment and diversity, which are not adequately satisfied by previous works. Aligning the distribution of the training data with query data expected during service time ensures that the model is customized for the intended usage. On the other hand, sufficient diversity allows the model to learn more knowledge and avoid overfitting. We present a framework that formulates task-specific data selection as an optimization problem based on optimal transport, a notion that captures the distance between two distributions. We add a regularization term to the optimal transport formulation to provide a smooth tradeoff between distribution alignment and diversity. In addition, we incorporate kernel density estimation into the regularizer to reduce the negative effects of near-duplicates in the candidate pool. Finally. we connect our optimization problem to nearest neighbor search and design efficient algorithms to compute the optimal solution based on approximate nearest neighbor search techniques. Our approach achieves an improvement of up to 5 points in F1 scores for targeted instruction tuning compared to the state-of-the-art method and demonstrates robustness against near-duplicates.

1 INTRODUCTION

1.1 Motivation

Machine learning (ML) has achieved remarkable success across a range of fields, including computer vision (Dosovitskiy et al., 2021; He et al., 2016), natural language processing (Vaswani et al., 2017; OpenAI, 2023), recommender systems (Covington et al., 2016; Zhou et al., 2018), scientific discovery (Jumper et al., 2021), and so forth. The profound advancements of ML are attributed not only to innovative model designs but also to well-curated data. As the prominent role of data in ML becomes widely recognized, the concept of *data-centric ML* (Zha et al., 2023) has emerged, underscoring the importance of meticulous data preparation and the right choice of data to feed the models.

There are several key characteristics of well-curated data that are critical for the performance of ML models. First, well-curated data should be clean, without data quality issues that could mislead the model. Previous research (Breck et al., 2019; Koh et al., 2018; Li et al., 2023) has demonstrated that systematic errors or maliciously injected noise in the training data can lead to models with low accuracy or models that make systematic mistakes. Furthermore, near-duplicates in the training set can impair training efficiency and introduce bias into the model (Hernandez et al., 2022; Lee et al., 2022; Tirumala et al., 2023). For example, as shown by Lee et al. (2022), language models trained on datasets with near-duplicates are more likely to generate sequences that are copies of examples in the training set. Second, data fed to a model should have a distribution that matches the data of the intended application, as highlighted by Murphy (2012) and Gururangan et al. (2020). In other words, the patterns or characteristics that the model learns from the training data should closely mirror those it will encounter at service time. We show an example of distribution matching in Figure 1.1. As real-world data tend to be messy and heterogeneous, it is essential to conduct *data validation* to ensure data quality and *data selection* to choose data that aligns with the desired distribution.

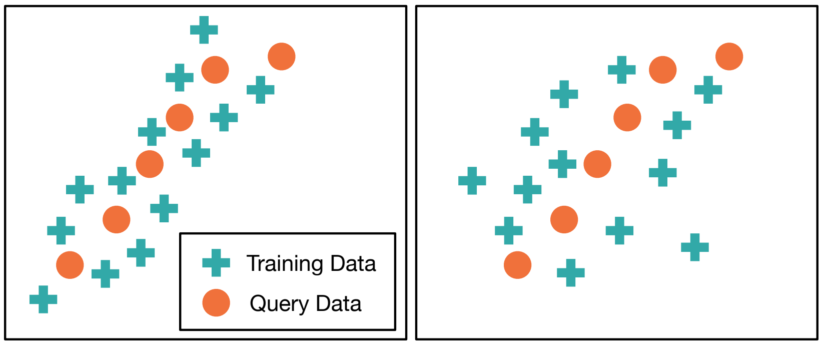


Figure 1.1: An illustration of distribution matching for 2-dimensional points. The training data in the left figure have better distribution alignment with the query examples at service time compared to the ones in the right figure.

In Figure 1.2, we show a simplified data pipeline in typical ML workflows. The raw data are cleaned before being forwarded to the validation module. If the data do not meet the validation criteria, they are returned for further cleaning. Once the data pass the validation, they will be selected and resampled to form the training dataset. In this dissertation, we focus on data validation and selection.

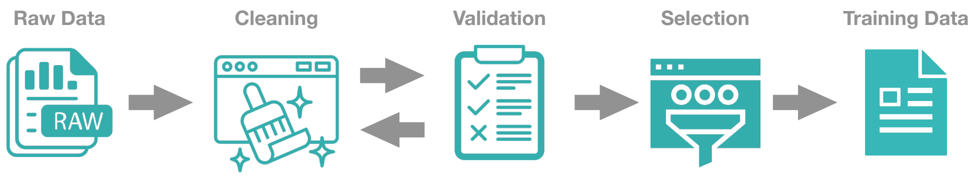


Figure 1.2: A simplified data pipeline in ML workflows.

Data validation is the process of assessing data quality with predefined standards. The problem of data validation has been extensively studied in the database community, where integrity constraints (Calì et al., 2002; Chu et al., 2013) such as key constraints, functional dependencies, and order dependencies are widely used to detect errors in tabular data. Additionally, ML platforms such as TFX (Polyzotis et al., 2019) and Deequ (Schelter et al., 2018) rely on assertions on feature statistics such as histograms, correlation, and mutual information to detect anomalies. Similarly, the natural language processing community (Laurençon et al., 2022; Rae et al., 2022) filters datasets based on text statistics such as word count, ratio of alphabetical characters, etc. Recently, researchers have proposed learning-based methods for data validation, using generative models to learn distributions of high-quality data (Eduardo et al., 2020; Wenzek et al., 2020) or discriminative models to distinguish between data from high-quality and noisy resources (Brown et al., 2020; Du et al., 2022).

Data selection aims to determine which data points to include in the training set to maximize a predefined utility that is often related to model performance. A common setting is that given a pool of candidate data points and a collection of sample queries that the model is expected to encounter at service time, select data from the candidates to form a training set. One line of data selection methods (Feng et al., 2022; Xie et al., 2023) stems from the one proposed by Moore and Lewis (2010), which evaluates the utility of a data point by taking the difference in probability scores given by two models: one trained on the data from the target task, and the other on general-purpose data. More recent approaches (Engstrom et al., 2024; Xia et al., 2024) select data points that have a high influence on the performance of the model for the target task.

Data validation and selection in real-world ML development rely heavily on human expertise and efforts, while automated methods have limitations that restrict their practical utility. Data validation processes in industrial ML platforms (Polyzotis et al., 2019; Schelter et al., 2018) require user-specified rule-based or schema-based quality assertions, where extensive data inspection by human experts is needed. Methods that discover constraints from data automatically (Zhang et al., 2020b; Bleifuß et al., 2017; Pena et al.,

2019) are either limited to specific types of constraints or computationally expensive. Learning-based quality filters (Wenzek et al., 2020; Brown et al., 2020; Du et al., 2022) make decisions automatically but require manual annotations on a set of data indicating whether the data come from a clean source or not when training the filters. Previous works on data selection relying on heuristics such as simple similarity search (Moore and Lewis, 2010; Xia et al., 2024) do not ensure that the selected data are distributed like the data from the target task. DSIR (Xie et al., 2023) introduces the concept of distribution matching but the focus on n-gram features of texts limits its application to other types of data and applications that require high-level semantics. Therefore, there is a need for principled and practical solutions for data validation and selection, which motivates this dissertation.

1.2 Dissertation Goal

This dissertation takes steps to improve the practicality of automated data validation and selection. Specifically, the goal is *to develop efficient, principled, and holistic frameworks for automated data validation and selection*. We move towards the goal by tackling the following problems.

First, *can we improve the efficiency of denial constraint discovery and verification to handle large-scale datasets in real-world applications?* Denial constraints (Chu et al., 2013) are a well-established formalism that captures a wide range of integrity constraints commonly encountered on tabular data, including candidate keys, functional dependencies, and ordering constraints, among others. Prior work exhibits notable limitations when confronted with large-scale datasets. The current state-of-the-art denial constraint verification algorithm (Pena et al., 2021) demonstrates a quadratic (worst-case) time complexity relative to the dataset's number of rows. In the context of denial constraint discovery, existing methodologies (Bleifuß et al., 2017; Pena et al., 2019) rely on a two-step algorithm that commences with an expensive data structure-building phase, often requiring hours to complete even for datasets containing only a few million rows. Consequently, users are left without any insights into the constraints on their dataset until the lengthy building phase concludes. Therefore, more efficient and flexible algorithms for denial constraint discovery and verification are desired for large-scale datasets in ML applications.

Second, *can we develop a data validation framework to guard against various types of data corruption on mixed-type tabular data while requiring minimal human efforts?* Data corruption in practical ML applications can be highly diverse, ranging from random noise, systematic mistakes to adversarial attacks (Koh et al., 2018; Li et al., 2023), which are often beyond the scope of constraint-based error detection (Pena et al., 2021) or simple statistical assertions (Polyzotis et al., 2019; Schelter et al., 2018). On the other hand, the mixed-type nature of tabular data renders adversarial attack detection methods (Steinhardt et al., 2017a) from the ML literature inapplicable, as these methods are primarily designed for real-valued data. In addition, as labeling data as clean or corrupted is excessively labor-intensive for the user, an effective data validation framework should avoid relying on such annotations. Therefore, there is a critical need to develop an innovative data validation framework that can effectively handle the complexity of mixed-type tabular data and diverse data corruption without the demand for manual labeling.

Third, *can we design a principled and generic framework to select target-specific data for ML training or finetuning?* The goal is to select training sets from a massive candidate pool automatically for a target task, using representative examples as a guide. Such a setting is common in the context of large language model pretraining or finetuning, where a massive and heterogeneous corpus crawled from the web is readily accessible, but task-specific data are needed to tailor a foundation model to the target distribution. As existing works (Moore and Lewis, 2010; Brown et al., 2020;

Gururangan et al., 2020) rely on simple heuristics or ad-hoc algorithms, their effectiveness across different settings is unclear. In addition, the lack of theoretical foundation casts doubts on the reliability of those methods. Instead, we seek a framework with solid theoretical foundations and the flexibility to be applied in different settings.

1.3 Contributions

In addressing the problems stated in the dissertation goal above, we make the following contributions.

Denial constraint discovery and verification. We present a general framework for efficient denial constraint discovery and verification. Our proposed algorithms for discovery and verification achieve $\tilde{O}(N)$ time and space complexity where N is the size of the dataset, a significant improvement over prior works (Pena et al., 2021; Bleifuß et al., 2017; Pena et al., 2019) whose complexity is $O(N^2)$. First, we introduce a near-optimal algorithm for validating any given denial constraint on the input dataset by making a connection between orthogonal range search and denial constraint verification. Second, we introduce the problem of anytime denial constraint discovery and propose an algorithm that leverages our novel verification algorithm to gradually output discovered constraints to users, eliminating the time-intensive blocking phase of intermediate data structure building that is employed in prior works (Bleifuß et al., 2017; Pena et al., 2019).

Safeguard against data corruption in ML pipelines We present a framework for safeguarding against corrupted data in ML pipelines. Our framework detects corrupted examples using a novel deep-learning model that captures the distribution of mixed-type tabular data, consistently achieving an AUROC score of above 80 points for corruption detection across different types of noise. Specifically, we design a new transformer-based model to capture the distribution of tabular data containing mixtures of numerical, categorical, and text-based entries. We employ a robust self-supervised training approach that operates directly on potentially corrupted data to train our model, without imposing any extra labeling burden on the user. We build an error detection mechanism based on the reconstruction loss from our model, which can be used to identify corruption in training data and validate inference queries. Furthermore, we conduct a theoretical analysis of mean estimation errors in the presence of coordinate-level corruption, providing insights into the worst-case effects that data corruption could impose on the learning process. Our analysis shows that one must exploit the structure of the data (i.e., dependencies between features) to achieve information-theoretically optimal errors for mean estimation.

Task-specific data selection We present a framework to select data for task-specific training or finetuning. Our framework samples task-specific data through regularized optimal transport for distribution alignment and diversity, achieving an improvement of up to 5 points in the F1 score over the state-of-the-art selection method (Xia et al., 2024) in instruction tuning for large language models on targeted tasks. Specifically, we formulate task-specific data selection as an optimization problem based on optimal transport, which is closely related to generalization error (Gálvez et al., 2021). The optimization problem allows a smooth trade-off between distribution alignment and diversity. We make our framework robust to near-duplicates in the candidate pool by incorporating kernel density estimation into the regularization term. We show the connection between the optimal solution to the optimization problem and nearest neighbor search. This connection allows us to develop efficient algorithms employing approximate nearest-neighbor search techniques (Douze et al., 2024).

1.4 Organization

In Chapter 2, we provide background on data validation and selection with related notations and terminology. In Chapter 3, we introduce our framework for denial constraint discovery and verification. In Chapter 4, we present our framework for safeguarding corruption in ML pipelines, along with a theoretical analysis of the worst-case effects of data corruption on mean estimation. In Chapter 5, we introduce our task-specific data selection framework. Chapter 6 concludes this dissertation, and discusses potential directions for future exploration.

2 BACKGROUND

In this chapter, we provide the necessary background for data validation and task-specific data selection. We first introduce notations and terminology that will be used throughout this dissertation. Then we introduce the concept of data validation and selection, along with methods that have been proposed in the literature. Finally, we present works that are related to this dissertation in a broader sense.

2.1 Notations

We use $\mathbb{R}_{\geq 0}$ to represent the set of non-negative real numbers, and $\mathbb{R}_{>0}$ to represent the set of positive real numbers. Let N be a positive integer and we use $[N]$ to denote the set of integers from 1 to N. We use bold letters to denote matrices and the corresponding plain letters with subscripts to denote the entries in the matrix. For example, $\boldsymbol{\gamma} \in \mathbb{R}^{M \times N}$ is an $M \times N$ matrix, and γ_{ij} or $\gamma_{i,j}$ is the entry in the i^{th} row and the j^{th} column (1-indexed).

Dataset. A dataset $\mathcal{D} = \{x_1, \ldots, x_N\}$ is a multiset of examples. Depending on the context, an example $x \in \mathcal{D}$ can be a vector, a chunk of text, or a tuple in a relation database. We use the multiset semantic to allow the existence of duplicates, as duplicates or near-duplicates are common in web-crawled data (Fröbe et al., 2021) used for the pretraining of foundation models. Depending on the context, each example may contain its corresponding label. We use $|\mathcal{D}|$ to denote the cardinality of $\mathcal{D}$.

Tabular Data. For tabular data, a dataset $\mathcal{D}$ is a relation R with a finite set of attributes (i.e., columns in the table). We use $\mathsf{vars}(R)$ to denote the set of attributes in the relation, and $x.\mathsf{A}$ to denote the value of attribute A on example x.

Table 2.1: A table of demographic information. Erroneous entries are marked in red colors.

	`Zip`	`State`	`Age`
x_1	53705	WI	36
x_2	53705	CA	25
x_3	90011	CA	34
x_4	90011	CA	1996-11-4

2.2 Data Validation

Data validation is the process of assessing data with predefined quality checks before using them. Data validation takes a dataset as input and outputs true or false indicating whether the input passes the validation. In practice, additional information such as examples or values that fail the validation is provided by the data validation process to help the user debug and make improvements.

Example 2.1. *Table 2.1 is a relation containing demographic information with the following errors:*

- x_4.***`Age`*** *fails the check that the* ***`Age`*** *column of every example should contain an integer value.*
- x_1 *and* x_2 *fail the check that* ***`Zip`*** *determines* ***`State`****, i.e., two examples with the same* ***`Zip`*** *should also have the same* ***`State`****.*

Therefore, the data validation process will output false for Table 2.1, along with the information that x_4.***`Age`*** *fails the data type check and* x_1, x_2 *violate the rule that* ***`Zip`*** *determines* ***`State`****.*

In the rest of this section, we introduce common types of quality checks for data validation in more detail.

Table 2.2: Tax rates for people in different states in the USA.

	SSN	Zip	Salary	FedTaxRate	State	StateCode
x_1	100	10108	3000	20%	New York	01
x_2	101	53703	5000	15%	Wisconsin	02
x_3	102	53703	6000	20%	Wisconsin	02
x_4	103	53703	4000	22%	Wisconsin	02

2.2.1 Denial Constraint

Denial constraints (Chu et al., 2013) are a highly expressive formalism of integrity constraints for tabular data. Denial constraints generalize several other types of integrity constraints, including unique column combinations, functional dependencies, and order dependencies.

A denial constraint states a rule that denies the existence of conflicting combinations of column values, which are determined by the predicate conjunctions in the denial constraint. Formally, a denial constraint φ on relation R is a conjunction of predicates in the following form:

$$\forall x, x' \in R, x \neq x' : \neg(p_1 \wedge \cdots \wedge p_m)$$

$p_1, \ldots, p_m$ are predicates in the form of $x.\mathsf{A}\ \mathsf{op}\ x'.\mathsf{B}$ where $\mathsf{op} \in \{=, \neq, \geq, >, \leq, <\}$ is one of the six operators, and $\mathsf{A}, \mathsf{B} \in \mathsf{vars}(R)$ are two (possibly the same) attributes. A pair of examples (x, x') is said to be a violation if all the predicates in φ evaluate to true. In the rest of this dissertation, we omit the universal quantification ($\forall x, x' \in R, x \neq x'$) when presenting denial constraints for conciseness.

We use Table 2.2 containing tax information to show some examples of denial constraints.

Example 2.2. *We consider the following denial constraints:*

- $\varphi_1 : \neg(x.\texttt{SSN} = x'.\texttt{SSN})$. `SSN` *is a candidate key, i.e., every example should have a unique* `SSN`*.*

- $\varphi_2 : \neg(x.\texttt{Zip} = x'.\texttt{Zip} \wedge x.\texttt{State} \neq x'.\texttt{State})$. `Zip` *determines* `State`, *i.e., two examples sharing the same* `Zip` *should share the same* `State`.
- $\varphi_3 : \neg(x.\texttt{State} = x'.\texttt{State} \wedge x.\texttt{Salary} \leq x'.\texttt{Salary} \wedge x.\texttt{FedTaxRate} > x'.\texttt{FedTaxRate})$. *In the same* `State`, *examples with higher* `Salary` *should also have higher* `FedTaxRate`.

φ_1 *and* φ_2 *have no violation on Table 2.2. There are four example pairs violating* φ_4*:* $(x_2, x_4), (x_3, x_4), (x_4, x_2), (x_4, x_3)$.

Data validation based on denial constraints operates as follows: for the input dataset D and a collection of denial constraints Φ, output true if for any $\varphi \in \Phi$, the number of violations does not exceed a pre-specified threshold, and output false otherwise. The threshold decides how much inconsistency we allow for the dataset. When the threshold is set to 0, the constraints need to be fully satisfied.

2.2.2 Statistical Assertions

Assertions on data statistics are employed by ML platforms in the industry (Polyzotis et al., 2019; Schelter et al., 2018) and some works on large language model pretraining (Raffel et al., 2020; Rae et al., 2022; Chen et al., 2021; Laurençon et al., 2022) to validate data quality. For the input dataset $\mathcal{D}$, an assertion is a rule in one of the following forms: (1) $\mathcal{T}(\mathcal{D}) \in \mathcal{R}$ and (2) $\forall x \in \mathcal{D}, \tau(x) \in \mathcal{R}$, where $\mathcal{T}$ computes a statistic on a dataset, and τ computes a statistic on a single example. $\mathcal{R}$ is the range that a statistic should lie in. Data validation based on such assertions outputs true if all the assertions hold on the input dataset, and outputs false otherwise.

In TFX (Polyzotis et al., 2019), an ML platform from Google, single-attribute statistics (e.g., minimum, maximum, mean, or median for numerical attributes, number of unique values, the most frequent value for categorical attributes) and cross-attribute statistics (e.g., correlations between attributes, mutual information of an attribute with the label) on the dataset level are supported for tabular data. In the natural language processing literature, assertions are made on statistics of individual examples, including word count (Raffel et al., 2020), the maximum number of consecutive repetitions of the same word (Laurençon et al., 2022), symbol-to-word ratio (Rae et al., 2022) and maximum line length (Chen et al., 2021), etc.

Example 2.3. *Xie et al. (2023) introduce a series of quality checks for their text dataset. To pass their quality checks, each example must satisfy the following assertions:*

1. *The minimum length of a word is at least* 40.
2. *The maximum length of a word is at most* 500.
3. *The repeat ratio is between* 0.02 *and* 0.2*, where the repeated ratio is defined as the maximum number of occurrences of the same word in an example divided by the word length of that example.*
4. *The informativeness ratio is between* 0.3 *and* 0.7*, where the informativeness ratio is the number of non-stop and non-punctuation words in an example divided by the word length of that example.*
5. *The numeric ratio is less than* 0.2*, where the numeric ratio is the number of numerical words in an example divided by the word length of that example.*

The words are tokens based on the NLTK word tokenizer (Bird et al., 2009).

2.2.3 Learning-Based Data Validation

Learning-based data validation methods rely on generative models to capture the distribution of high-quality data, or discriminative models to distinguish between high-quality and low-quality data.

The first line of works (Wenzek et al., 2020) assume access to a high-quality dataset $\mathcal{D}_{\text{high}}$ and use a generative model to learn the distribution of high-quality data from $\mathcal{D}_{\text{high}}$. The second line of works (Brown et al., 2020; Du et al., 2022; Gao et al., 2020) require a high-quality dataset $\mathcal{D}_{\text{high}}$ and a low-quality dataset $\mathcal{D}_{\text{low}}$ to train a classifier that distinguish between the two distributions. In the data validation process, the trained model assigns a confidence score to each example in the candidate dataset indicating how likely the example comes from the high-quality distribution. In Figure 2.1, we show the workflow of learning-based data validation. In stage 1, either a generative or binary classifier is trained, which will be used to assign scores to the candidate examples to be validated in stage 2. For a dataset to pass the validation, all the examples should have a confidence score above some threshold. Whenever a dataset fails the validation, a simple filtering of the examples with low confidence scores is often applied.

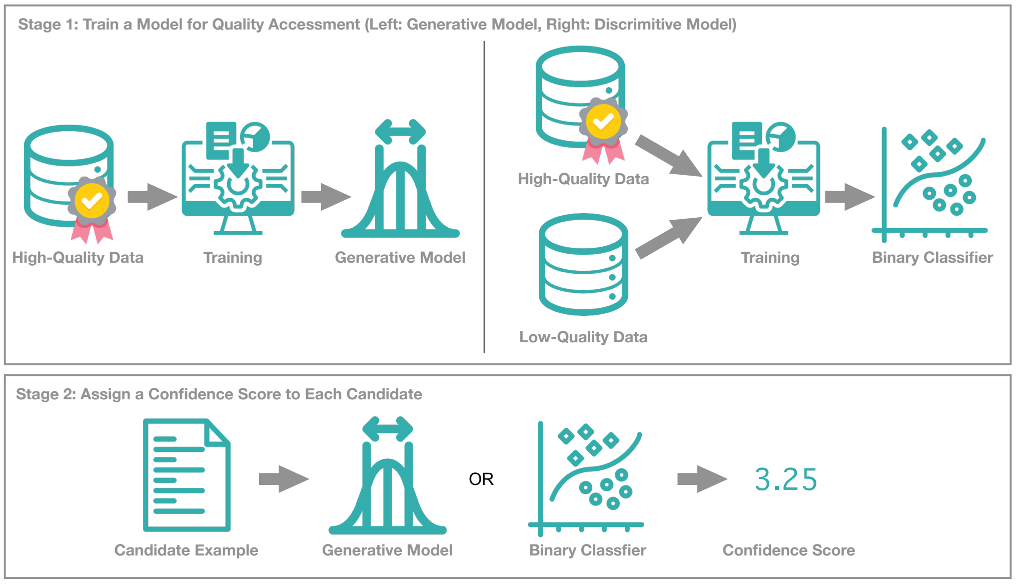


Figure 2.1: Workflow of learning-based data validation.

There are also outlier detection methods (Chen et al., 2001; Liu et al., 2008; Eduardo et al., 2020) in the statistic literature that do not need access to $\mathcal{D}_{\text{high}}$ and $\mathcal{D}_{\text{low}}$. Instead, they learn the distribution of high-quality data directly from the dataset to be validated, which is possibly clean or noisy.

2.3 Task-Specific Data Selection

Task-specific data selection aims at selecting proper training data for a target task from a candidate data pool. This dissertation focuses on *automated selection*, where a small set of samples from the target task is provided to guide the selection. We show an illustration of the setting in Figure 2.2. Formally, task-specific data selection takes a set of representatives $\mathcal{Q} = \{q_i\}_{i=1}^{M}$ from the target task and a candidate pool $\mathcal{D} = \{x_j\}_{j=1}^{N}$ as inputs, and outputs a subset or a resample of $\mathcal{D}$. Such a setting is common in practice, especially in the context of large language model finetuning. For example, recently there has been a trend in the industry to build customized chatbots for specific tasks (Conover et al., 2023) such as question-answering on user data and domain-specific information retrieval. To build such chatbots, a practitioner may manually craft a set of examples to showcase the desired data and use them to guide the selection of relevant data from a massive web-crawled corpus or an internal data lake.

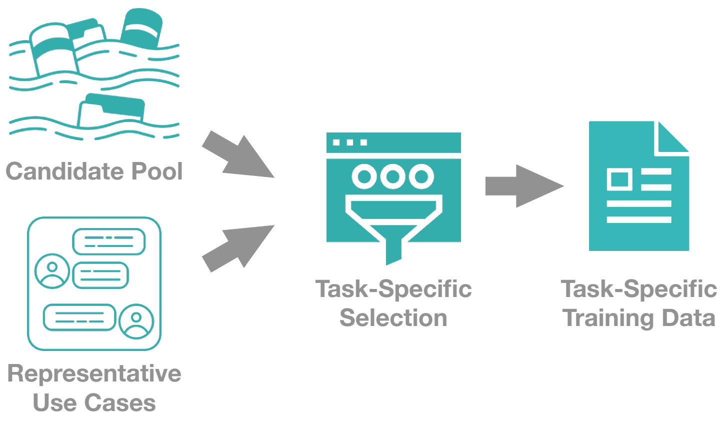


Figure 2.2: Target-specific data selection.

Task-specific data selection methods can be model-agnostic or model-specific, depending on whether the selection process considers downstream training. In the rest of this section, we will introduce model-agnostic and model-specific methods proposed by previous works.

2.3.1 Model-Agnostic Selection

Model-agnostic data selection is decoupled from the downstream training process. Similarity-based methods (Ruder and Plank, 2017a; Gururangan et al., 2020; Aharoni and Goldberg, 2020; Yao et al., 2022) retrieves the top examples from the candidate pool, ranked by their similarity to the representative from the target task. The features used for similarity computation can be embeddings or ngrams for texts. Another line of works (Moore and Lewis, 2010; Xie et al., 2023) use two generative models where one of them learns the distribution of the data from the target task and the other learns the distribution of general-purpose data. Moore and Lewis (2010) rank the candidate examples by the differences in log probability given by the two models. Xie et al. (2023) perform importance resampling to match the distribution of the resampled candidate pool and the target distribution.

2.3.2 Model-Specific Selection

Model-specific data selection methods (Engstrom et al., 2024; Xia et al., 2024) choose data to maximize the model performance on the target task. Given the high cost of actually training a model and evaluating it on the target task, these methods often estimate the model performance by approximation. Engstrom et al. (2024) approximate the model performance using datamodels (Ilyas et al., 2022), a function that maps the training data membership (the information of whether each candidate is included in the training set or not) to the model performance. Formally, let $\mathcal{S} \subseteq \mathcal{D}$ be a selected training set and $\mathbf{1}_{\mathcal{S}} \in \{0, 1\}^N$ be a binary vector where the j^{th}

element is 1 if and only if $x_j \in \mathcal{S}$. Datamodels predicate the loss on query example q_i as $\boldsymbol{\theta}_i^T \mathbf{1}_{\mathcal{S}}$ when the model is trained on $\mathcal{S}$, where $\boldsymbol{\theta}_i \in \mathbb{R}^N$ is the parameters learned from randomly selected subsets and the corresponding evaluation loss on q_i. Engstrom et al. (2024) select x_j's with the top-lowest influences on the loss of the query examples, i.e., $\sum_{i=1}^{M} (\boldsymbol{\theta}_i)_j$. Xia et al. (2024) employ the influence function (Koh and Liang, 2017) to approximate the marginal gain on the model performance when including a candidate into the training set. Specifically, they first train a model on a random subset of the candidate pool and then approximate the change of loss on the query examples (i.e., the influence) when including a new candidate into the training set by performing a single Newton step. Candidates with the top-lowest influences will be selected.

Note that these model-specific methods can be made model-agnostic if we use a surrogate model in the selection process. A surrogate model is a simpler model used to approximate the behavior of the actual model. Surrogate models are particularly useful in predicting the change in model performance in response to changes in training data or hyperparameters. When we use a fixed surrogate model to replace the actual model in the data selection process, any model-specific selection method is decoupled from the actual model.

2.4 Related Works

Denial Constraint Verification. Denial constraint verification is the problem of verifying whether a constraint holds on a given input or not. Facet (Pena et al., 2021) is the state-of-the-art algorithm for denial constraint verification. Facet follows the design of VioFinder (Pena et al., 2020) which processes the predicates in a denial constraint one by one to rule out potential violations. Previous works on data cleaning (Rekatsinas et al., 2017; Geerts et al., 2020; Fan et al., 2021) rely on relational database management systems

(DBMS) to detect denial constraint violations, where denial constraints are translated into SQL queries. Those DBMS-based methods often fall short when the given denial constraint contains inequalities, and they are slower than denial constraint-specific methods by orders of magnitude as shown in previous works (Pena et al., 2020, 2021).

Denial Constraint Discovery. Denial constraint discovery is the problem of identifying denial constraints holding on a given dataset. Previous works (Bleifuß et al., 2017; Pena et al., 2019, 2022; Xiao et al., 2022) on denial constraint discovery operate in a two-step framework originally proposed by Chu et al. (2013), where an intermediate data structure called the evidence set is built before denial constraint enumeration. In particular, Hydra (Bleifuß et al., 2017) first builds an approximate evidence set on a small sample of the input data, and then refines it on the full dataset. Hydra only enumerates exact constraints that have no violations on the dataset. DCFinder (Pena et al., 2019) employs a different enumeration algorithm that supports approximate denial constraint discovery, which allows a specified number of violations. Subsequent works (Pena et al., 2022; Xiao et al., 2022) further accelerate the discovery process by parallel computation and data structure compression, but they still follow the two-step framework. The building phase of the intermediate data structure is often the bottleneck of the methods following this framework, which have quadratic time complexity with respect to the number of examples in the input dataset.

Relation between Data Quality and ML Performance. Li et al. (2019) study the effects of data cleaning on the performance of ML models. Their findings suggest that data cleaning tends to have positive or insignificant effects on ML performance, and the impact varies across different datasets. Karlaš et al. (2020) present an algorithm to decide whether the prediction for a given query is affected by the missing values in the training set for nearest neighbor classifiers. Bian et al. (2023) propose an algorithm that identifies data points for which cleaning has no effect on the learning results with a setting limited to Naïve Bayes classifiers and missing values. ActiveClean (Krishnan et al., 2016) treats data cleaning for ML as an active learning problem, suggesting a subset for cleaning in each iteration. They achieve provable convergence for convex models. However, modeling the connection between data cleaning and ML performance is challenging in more general settings.

Adversarial Attacks and Defenses. Adversarial attacks inject malicious noisy to mislead ML models. Training time attacks (Koh et al., 2018; Muñoz-González et al., 2017; Biggio et al., 2012) add poisoned samples to corrupt the target model. Filtering-based defenses (Steinhardt et al., 2017b; Diakonikolas et al., 2019b) remove suspicious samples during training based on training statistics. Inference time attacks (Madry et al., 2018a; Carlini and Wagner, 2017; Moosavi-Dezfooli et al., 2016; Gao et al., 2022a) add small perturbations to test samples to fool the classifier. Efforts have been made to improve the robustness of the model by training data augmentation (Goodfellow et al., 2014; Madry et al., 2018b; Gao et al., 2022b; Zhang et al., 2020a) or making modifications to the model (Xiao et al., 2019; Pang et al., 2020, 2019; Zhang et al., 2021). Those works focus on robustness from the model perspective without assessment of data quality. Another group of defenses tries to detect adversarial samples at inference time. Roth et al. (2019) and Hu et al. (2019) add random noise to input samples and detect suspicious ones based on the changes in the logit values. Grosse et al. (2017) assume that adversarial samples have different distributions from benign samples and add another class to the classifier to detect them.

Data Deduplication Data deduplication removes duplicates or near-duplicates from a dataset. The problem of data deduplication has been studied for decades (Christen, 2012), while we focus on deduplication in the context of data preparation for ML training. Exact duplicates can be detected using hash functions (Elazar et al., 2024; Wenzek et al., 2020), while the detection of near-duplicates is more challenging. Rae et al. (2022)

and Gao et al. (2020) identify near-duplicates utilizing locality-sensitive hashing (Gionis et al., 1999). Lee et al. (2022) and Chowdhery et al. (2024) compute edit distances between examples to find near-duplicates. Another line of works (Abbas et al., 2023; Tirumala et al., 2023) relies on learned embeddings of the examples to detect near-duplicates.

3 EFFICIENT VERIFICATION AND DISCOVERY OF DENIAL CONSTRAINTS

3.1 Introduction

Integrity constraints play a pivotal role in a wide range of data analysis tasks such as data exploration (Abedjan et al., 2016; Fariha et al., 2021), data cleaning and repair (Rekatsinas et al., 2017; Giannakopoulou et al., 2020), data synthesis (Ge et al., 2021), and query optimization (Kossmann et al., 2022). By enforcing integrity constraints, users can ensure the reliability, consistency, and accuracy of their data, enabling them to make informed decisions, derive meaningful insights, and extract maximum value from their datasets. One class of constraints that is of particular interest is known as *Denial Constraints* (DCs) (Chu et al., 2013). DCs are appealing since they are expressive enough to capture many integrity constraints that are useful in practice such as functional dependencies, ordering constraints, and unique column combinations among others.

In this chapter, we study the problem of efficient DC *verification* and its application in DC *discovery*. DC verification involves deciding whether a given DC is satisfied on a specific dataset and is particularly valuable during data exploration, where analysts aim to ascertain the presence or absence of specific patterns within the dataset. Additionally, it serves as a valuable tool in assessing dataset quality, enabling the identification of noisy or inconsistent data instances (Fariha et al., 2021). DC discovery involves the automatic discovery of DCs from a given dataset, which holds significant appeal due to the inherent challenges associated with the manual identification of DCs. The manual approach not only requires expertise and

The work described in this chapter is done while interning at Microsoft.

significant time investment but also suffers from a higher likelihood of errors, given the intricate and ever-evolving nature of datasets.

In recent years, substantial advancements happened in the field of DC verification and discovery (Pena et al., 2019; Bleifuß et al., 2017; Pena et al., 2020, 2021, 2022). However, existing methods expose noteworthy limitations on real-world production datasets in practical use. First, in the context of DC verification, the best-known algorithm, FACET (Pena et al., 2021), has a worst-case time and space complexity $\Omega(|R|^2)$ on a given relation R with cardinality $|R|$ (number of rows). In the context of DC discovery, prior works (Chu et al., 2013; Bleifuß et al., 2017; Pena et al., 2019, 2022; Xiao et al., 2022) follow a two-step process: (1) building an intermediate data structure called *evidence set* from the input, which is the most computationally demanding aspect of the DC discovery process (Pena et al., 2022), and (2) mining the DCs from the evidence set using various set-covering algorithms, which could also be costly depending on the number of DCs and the size of the evidence set. The time required to construct the evidence set is often prohibitive. Even for medium-sized datasets (e.g., 5 million rows and 30 columns), evidence set construction can take up to several hours. In addition, the two-phase approach employed by previous works has a fundamental limitation that can lead to poor user experience. In particular, the approach is "all or none", i.e., to produce any DC, it needs to complete the full evidence set construction (which is time-consuming), at the end of which it reports all DCs. An alternative method for DC discovery is to conduct a lattice search (Huhtala et al., 1999), enumerating and verifying DCs from simple to more complex ones. This method provides the flexibility of terminating the discovery process prematurely, allowing the user to interrupt the execution at any time when they are satisfied with the set of DCs already mined (i.e., the anytime property). However, the substantial expense associated with existing solutions for DC verification largely limits the efficiency of lattice-search-based DC discovery.

In this chapter, we propose an efficient algorithm for DC verification with near-linear time and space complexity relative to dataset size. We make a connection between the problem of DC verification and *orthogonal range search* (Bentley and Friedman, 1979; Bentley and Saxe, 1980), a celebrated line of work in computational geometry, which studies the problem of determining which k-dimensional objects in a set intersect with a given query object. We show that it is possible to design a near-optimal algorithm for verifying a given DC over a specific dataset by leveraging techniques employed for orthogonal range search. Our proposed algorithm has a time complexity of $O(|R| \log^{f(\varphi)} |R|)$, where $f(\varphi)$ is a parameter that is dependent only on the characteristics of the DC φ and not on the input dataset R. This represents a theoretical improvement over prior work and translates into an order of magnitude better performance in practice.

In addition, we show that integrating our DC verification algorithm with a lattice search results in an efficient anytime (Zilberstein, 1996) algorithm for DC discovery, allowing for progressive constraint discovery and early termination. At a high level, we perform a lattice-based traversal of the space of candidate DCs and invoke our novel DC verification algorithm to confirm whether a given constraint holds. Valid DCs are outputted immediately upon verification.

Our contributions Our key contribution is a framework, RAPIDASH, that relies on a novel approach for DC verification leveraging the connection to orthogonal range search. Specifically, we make the following contributions:

1. **A novel DC verification algorithm.** We present a near-optimal algorithm for verifying a given DC on a dataset R. We prove that our proposed algorithm can achieve a near-linear time and space complexity relative to dataset size. This represents a significant improvement over the best-known verification algorithm (Pena et al., 2021), which has a worst-case quadratic complexity both in time and space. Further, we show that in certain scenarios, our algorithm can run in only linear

space while still achieving provably sub-quadratic running time.

2. **Efficient DC discovery.** We plug our novel DC verification algorithm into lattice-search-based DC discovery, resulting in an efficient DC discovery method that satisfies the anytime property. We also present several pruning criteria to further reduce the cost of lattice searches.

3. **Experimental evaluation.** We conduct an extensive empirical evaluation over both open-source and real-world production datasets, notably the latter being an order of magnitude larger than the datasets employed in prior studies. We show that Rapidash achieves up to 84× speedup over the state-of-the-art (Pena et al., 2021) for DC verification. For DC discovery, our anytime algorithm can produce all single-column DCs (e.g., single-column candidate keys, columns with identical values, etc.) within the first 10 minutes, while prior work (Bleifuß et al., 2017; Pena et al., 2019) fails to produce any output within the first 48 hours.

3.2 Preliminaries and Problem Statement

In this section, we introduce the formulation of DCs and the concept of search space for DC discovery. Then we make our problem statement.

3.2.1 Definition and Classification of DCs

A DC φ on a relation R is a conjunction of predicates in the following form:

$$\forall x, x' \in R, x \neq x' : \neg(p_1 \wedge \cdots \wedge p_m)$$

$p_1, \ldots, p_m$ are predicates in the form of $x.\mathsf{A}\ \mathsf{op}\ x'.\mathsf{B}$ where $\mathsf{op} \in \{=, \neq, \geq, >, \leq, <\}$ is one of the six operators, and $\mathsf{A}, \mathsf{B} \in \mathsf{vars}(R)$ are two (possibly the same) attributes in the relation. We will refer to $\neq$ as disequality and $\geq, >, \leq, <$ as inequalities. All operators except equality will be collectively referred to as non-equality operators. A pair of examples (x, x') is said to be a violation to φ if all the predicates in φ evaluate to true. We will say that a φ holds on R if there are no violations, i.e., the DC is *exact*. An exact DC is said to be minimal if no proper subset of its predicates forms another exact DC.

A predicate is said to be *homogeneous* if it is of the form $x.\mathsf{A}$ `op` $x'.\mathsf{A}$, i.e. it is defined over a single column, and *heterogeneous* if it is of the form $x.\mathsf{A}$ `op` $x'.\mathsf{B}$ where $A \neq B$. We use the term homogeneous DC to refer to a DC that contains only homogeneous predicates. A heterogeneous DC can contain both types of predicates. Without loss of generality, we will assume that each column of R participates in at most one predicate of a homogeneous DC. We will use $\mathsf{vars}_{\mathsf{op}}(\varphi)$ to denote the set of columns in a homogeneous DC that appear in some predicate with the operator `op`.

3.2.2 Predicate Space

The space of DCs is governed by the *predicate space*, the set of all predicates that are allowed on R. As noted in Bleifuß et al. (2017); Pena et al. (2019), a predicate is meaningful when a proper comparison operator is applied to a pair of comparable attributes. Specifically, all the six operators can be used on numerical attributes (i.e. they are continuous), e.g., age and salary, but only $=$ and $\neq$ can be used on categorical attributes such as name and address. Two attributes are said to be comparable if: (i) they have the same type; (ii) the active domain overlap is at least 30% Bleifuß et al. (2017); Pena et al. (2019). For example, in Table 2.2, column `Salary` and `State` are not comparable since they have different types, and `SSN` and `Zip` are not comparable since the values do not have any overlap.

3.2.3 Problem Statement

We use the term *DC verification* for the process of determining whether a DC holds on a relation R and *DC discovery* to refer to the process of finding (some or all) minimal DCs over R. In the rest of this chapter, we focus on the following two problems.

1. (DC verification) Given a relation R and a DC φ, determine whether φ holds on R and enumerate the tuple pairs that violate the DC if there are any.
2. (DC discovery) Given a relation R and a predicate space, design an efficient, anytime DC discovery algorithm to find DCs that have no violations.

An anytime algorithm is required to produce an increasing number of valid DCs as time progresses in a way that we have some DCs even if the algorithm is interrupted before it terminates.

3.3 Rapidash for Denial Constraint Verification

In this section, we describe the general ideas underlying RAPIDASH followed by specific improvements and optimizations. Our algorithm builds appropriate data structures to store the input data (leveraging existing work on orthogonal range search), and issues appropriate queries to detect violations of a given DC on the data. For ease of exposition and aiding readability, we will focus on the problem of DC verification that decides whether the given DC holds and outputs a boolean answer. At the end of the section, we point out how our main algorithm can be readily modified in a minimal way to also support violation enumeration.

3.3.1 Foundation

Before we delve into the details of our proposed algorithm and its relationship to orthogonal range search, we will provide some intuition behind the design of the algorithm taking as an example the simplest scenario: homogeneous constraints with equality predicates only. The constraint $\varphi_1 : \neg(x.\texttt{SSN} = x'.\texttt{SSN})$ presented in the previous section is a qualifying constraint. The verification algorithm should return `True` over a relation R, when `SSN` is a candidate key. If at least one pair of tuples in R shares the same `SSN` value, then the algorithm should return `False`.

To evaluate whether such a tuple exists, we can incrementally populate a hash table tuple-by-tuple. The intuition is that if two tuples fall in the same hash partition then they have the same `SSN` value, and thus, we have identified a violation. The steps are presented in Algorithm 1. For each new tuple, we extract the `SSN` value 3 and check whether it already exists in the hash table. If not, then we create a new entry with an associated count of 1. If there is already an entry then we increase the count by 1. If the count becomes greater than 1, we have identified two tuples with the same `SSN` value, and we return `False`. For example, in Table 2.2, the algorithm would create 4 hash table entries (one per `SSN` value), each with count 1, and thus, the constraint would evaluate to `True`. The algorithm works similarly with constraints having more than one equality predicate. This algorithm is straightforward and easy to understand, yet it grows more complex with the inclusion of non-equality predicates. It is at this juncture that orthogonal range search becomes relevant. Before we dive deeper into these scenarios, we will first provide some background on orthogonal range search.

3.3.2 Orthogonal Range Search

In this subsection, we present some background on orthogonal range search (Bentley and Friedman, 1979; Bentley and Saxe, 1980). Given a totally ordered

Algorithm 1: DC verification for homogeneous constraints with equality predicates

```
   Input  : Relation $R$, Homogeneous DC $\varphi$ with only equality predicates.
   Output : True/False
1  $H \leftarrow$ empty hash table
2  foreach $x \in R$ do
       /* Project tuple $x$ on the columns participating in
          equality predicates                                   */
3      $v \leftarrow \pi_{\mathsf{vars}_{=}(\varphi)}(x)$
4      if $v \notin H$ then
5          $H[v] \leftarrow 0$
6      $H[v] \leftarrow H[v] + 1$
7      if $H[v] > 1$ then
           /* Hash collision -- violation detected              */
8          return False
9  return True
```

domain $\mathbb{N}$, let $A \subseteq \mathbb{N}^k$, for some $k \geq 1$, be a set of size N. Let $\mathbf{L} = (l_1, \ldots, l_k)$ and $\mathbf{U} = (u_1, \ldots, u_k)$ be such that $\mathbf{L}, \mathbf{U} \in \mathbb{N}^k$ and $l_i \leq u_i$ for all $i \in [k]$.

Definition 3.1. *An orthogonal range search query is denoted by* $(\mathbf{L}, \mathbf{U})$*, and its evaluation over* A *consists of enumerating the set*

$$Q(A) = \{a \in A \mid \bigwedge_{i \in [k]} l_i \ \boldsymbol{op}_1 \ a_i \ \boldsymbol{op}_2 \ u_i\}$$

where $\boldsymbol{op}_1, \boldsymbol{op}_2 \in \{=, <, \leq\}$.

In other words, $\mathbf{L}$ and $\mathbf{U}$ form an axis-aligned hypercube in k dimensions, and $Q(A)$ reports all points in A that lie on/within the hypercube. The Boolean version of the orthogonal range search problem consists of determining if $Q(A)$ is empty or not.

Example 3.2. *Consider Table 2.2. Let* A *be the set of two-dimensional points obtained by projecting the table on* `Salary` *and* `FedTaxRate`*. Let* $\mathbf{L} = (3500, 5)$ *and* $\mathbf{U} = (4500, 25)$*. Then, the orthogonal range query* $(\mathbf{L}, \mathbf{U})$

is asking for all points such that the `Salary` is between 3500 *and* 4500*, and the `FedTaxRate` is between* 5 *and* 25*. In Table 2.2, only x_4 satisfies the criteria (its values of `Salary` and `FedTaxRate` are* 4000 *and* 22 *respectively). Thus, the result of the orthogonal range search query* $(\mathbf{L}, \mathbf{U})$ *is* $\{(4000, 22)\}$*.*

The two most celebrated data structures for orthogonal range search are range trees and kd-trees (Bentley and Friedman, 1979). We will review their complexity and trade-offs when analyzing the complexity of our algorithms.

3.3.3 Verification Algorithm

In this subsection, we present our verification algorithm that leverages prior work on orthogonal range search. The algorithm builds on top of the ideas behind Algorithm 1 but extends them to cover homogeneous constraints that contain both equalities and inequalities. Without loss of generality, we will assume that none of the predicates contain disequality. This assumption will be removed later. Finally, we assume that the categorical columns in R have been dictionary-encoded to integers, a standard assumption in line with prior work Pena et al. (2019, 2021).

The algorithm preserves the core concepts of Algorithm 1, namely the use of a hash table and early termination in case of violations. The primary modification involves incorporating orthogonal range search indexes to identify violations stemming from inequality predicates. The algorithm is presented in Algorithm 2.

For a DC to be false, all predicates must be true for some tuple pair. As soon as we have identified such a tuple pair, we can safely terminate and return `False` to the user.

We first compute the number k of columns in inequality predicates (we assume $k > 0$ as Algorithm 1 covers the case where $k = 0$). We then proceed similarly as before: project each tuple on the columns participating in equality predicates (v on line 4) and evaluate whether the resulting tuple

Algorithm 2: DC verification for homogeneous constraints with inequality predicates

```
Input  : Relation $R$, Homogeneous DC $\varphi$ with inequalities.
Output : True/False
1  $k \leftarrow$ #number of columns appearing in inequality predicates
2  $H \leftarrow$ empty hash table
3  foreach $x \in R$ do
       /* Project tuple $x$ on the columns participating in
          equality predicates                                  */
4      $v \leftarrow \pi_{\mathsf{vars}_=(\varphi)}(x)$
5      if $v \notin H$ then
6          $H[v] \leftarrow$ new OrthogonalRangeSearchTree($k$)
       /* Project tuple $x$ on the columns participating in
          inequality predicates                                */
7      $z \leftarrow \pi_{\mathsf{vars}(\varphi)\setminus\mathsf{vars}_=(\varphi)}(x)$
8      if $\neg H[v]$.isEmpty() then
           /* Evaluate violations through two range search
              queries                                          */
9          $\mathbf{L}, \mathbf{U}, \mathbf{L}', \mathbf{U}' \leftarrow$ CreateRangeSearchQueries($z, \varphi$)
10         if $H[v]$.booleanRangeSearch($\mathbf{L}, \mathbf{U}$) $\vee$
            $H[v]$.booleanRangeSearch($\mathbf{L}', \mathbf{U}'$) then
               /* Violation detected                           */
11             return False
       /* Insert $z$ into the range tree                       */
12     $H[v]$.insert($z$)
13 return True
14 procedure CreateRangeSearchQueries($z$, $\varphi$)
       /* $\mathbf{L}$ and $\mathbf{U}$ are indexed by the non-equality predicates $p_i$.
          Both are of size k                                   */
15     $\mathbf{L} \leftarrow (-\infty, \ldots, -\infty), \mathbf{U} \leftarrow (\infty, \ldots, \infty)$
       /* Create range search query ($\mathbf{L}$, $\mathbf{U}$)                */
16     foreach inequality predicate $p_i \in \varphi$ do
17         if $p_i$.op is $<$ or $\leq$ then
18             $\mathbf{U}_i \leftarrow \pi_{p_i.\mathsf{col}}(z)$
19         if $p_i$.op is $>$ or $\geq$ then
20             $\mathbf{L}_i \leftarrow \pi_{p_i.\mathsf{col}}(z)$
       /* Create inverted range search query ($\mathbf{L}'$, $\mathbf{U}'$)     */
21     $\mathbf{U}' \leftarrow \mathbf{L}, \mathbf{L}' \leftarrow \mathbf{U}$
22     flip $-\infty$ to $\infty$ and $\infty$ to $-\infty$ in $\mathbf{U}'$ and $\mathbf{L}'$ respectively
23     return $\mathbf{L}, \mathbf{U}, \mathbf{L}', \mathbf{U}'$
```

has been seen before (line 5). If not, we create a new hash table entry whose value now, instead of being an integer counter, is a range search tree of k dimensions (line 6). This tree will be used to index the k-dimensional tuples containing the columns participating in inequality predicates and identify violations. Before we further delve into the pseudo-code, we explain the main intuition through an example.

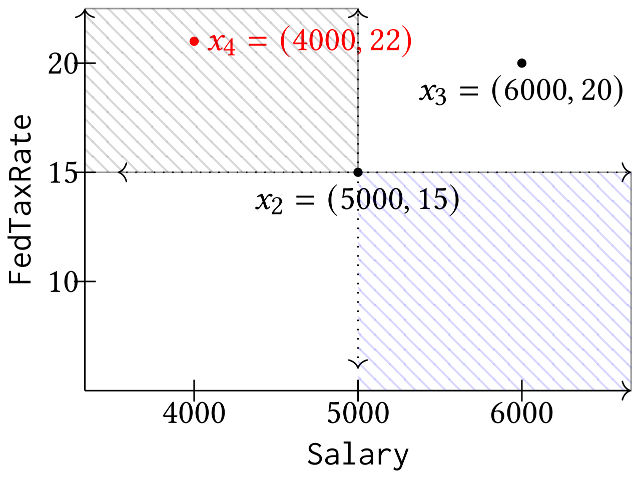


Figure 3.1: `Salary` and `FedTaxRate` for each tuple in `Tax`. The grey (upper left quadrant centered at x_2) and blue shaded areas (lower right quadrant centered at x_2) show the regions where the tuples that could form a violation with x_2 lies.

Example 3.3. *Consider the relation* ***Tax*** *from in Table 2.2 and the DC* $\varphi_3 : \neg(x.\texttt{State} = x'.\texttt{State} \wedge x.\texttt{Salary} \leq x'.\texttt{Salary} \wedge x.\texttt{FedTaxRate} > x'.\texttt{FedTaxRate})$*, which contains one equality and two inequality predicates* $(k = 2)$*. For simplicity, we omit the details of how range search works in this example but present it later. Algorithm 2 will first start with the equality predicate, and place* x_1 *in a hash partition by hashing* $x_1.\texttt{State} =$ *New York and initialize a 2-dimensional range search tree for that partition (line 6). Since this tree is empty, we do not perform any violation detection (line 8) and we insert* $(x_1.\texttt{Salary}, x_1.\texttt{FedTaxRate}) = (3000, 20)$ *in the tree (line 12).*

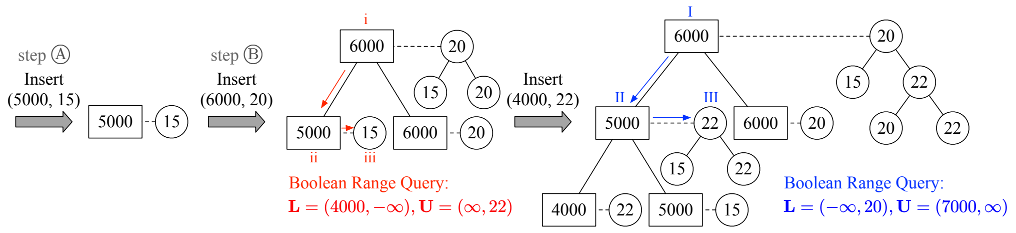


Figure 3.2: An illustration of insertion and search in a 2D range tree that stores (`Salary`, `FedTaxRate`). The primary tree with rectangle nodes stores `Salary`, and the secondary trees with circle nodes store `FedTaxRate`. The Roman numbers denote the visiting order during the search.

Next, we process x_2, which is placed in a different hash partition since x_2.`State` $=$ *Wisconsin and we initialize a new range tree for this partition. As in the previous step, we then insert* (x_1.`Salary`, x_1.`FedTaxRate`) $=$ $(5000, 15)$ *in the tree (call this step* Ⓐ*). When x_3 is processed, it is placed in the same partition as x_2 since they have the same* `State` *value.*

At this point, we have two tuples in the same hash partition, and thus we need to consider the inequality predicates in the DC to establish whether there is a DC violation. This is where the orthogonal range search tree is leveraged. Such a violation would occur in two scenarios: (1) if any tuple in the tree (x_2 in this case) has ***`Salary`*** *less than x_3.****`Salary`*** $= 6000$ *but* ***`FedTaxRate`*** *more than x_3.****`FedTaxRate`*** $= 20$*, or (2) if any tuple in the tree has* ***`Salary`*** *more than x_3.****`Salary`*** *but* ***`FedTaxRate`*** *less than x_3.****`FedTaxRate`****.*

To identify whether any of the two scenarios above is true, we perform two orthogonal range search queries using the values of x_3 to probe the index. More specifically, we perform a search with $\mathbf{L} = (-\infty, 20)$ *and* $\mathbf{U} = (6000, \infty)$ *(scenario 1). Then, we also search in the* inverted *range* $\mathbf{L}' = (6000, -\infty)$ *and* $\mathbf{U}' = (\infty, 20)$ *(scenario 2). Since x_2 does not lie in the desired range, both range searches return false. Figure 3.1 visualizes this result. Since there is no violation, we insert* $(6000, 20)$ *in the tree (call this step* Ⓑ*)*

Finally, x_4 (highlighted in red in Figure 3.1) is processed and placed in the same partition as x_2 and x_3 because of the same value in ***`State`*** *column.*

We again initiate two range search queries based on x_4 `Salary` $= 4000$ and `FedTaxRate` $= 22$. The queries would be $\mathbf{L} = (-\infty, 22), \mathbf{U} = (4000, \infty)$ and $\mathbf{L}' = (4000, -\infty), \mathbf{U}' = (\infty, 22)$. Then, x_2 and x_3 form a violation wrt. x_4 since both the points represent a higher salary than 4000 but a lower tax rate than 22, and Line 11 returns False.

Algorithm 2 formalizes the process described in the example above. When it comes to evaluating inequality predicates, we generate two range queries based on the values of the current tuple and the operator type in the inequality predicates (line 9) and then search the range tree (line 10). If any of the range search queries return `True`, a DC violation is detected and the algorithm terminates (line 11). Otherwise, the tuple is inserted in the tree (line 12), and the algorithm continues with the next tuple in the relation.

Table 3.1: Data structure parameter on input of size n (Overmars, 1983). k is the number of dimensions of the points inserted in the tree.

DS	Insertion $I(n)$	Answering $T(n)$	Space $S(n)$
Range tree	$O(\log^k n)$	$O(\log^k n)$	$O(n \cdot \log^{k-1} n)$
kd-tree	$O(\log n)$	$O(n^{1-\frac{1}{k}})$	$O(n)$

Seminal work by Overmars (1983) showed that using range trees and kd-trees, one can design an algorithm with the parameters as shown in Table 3.1. We now demonstrate an example of how range trees are used by Algorithm 2 when performing step Ⓐ and step Ⓑ in Example 3.3.

Example 3.4. *In Figure 3.2, we illustrate insertion and search in a range tree. The tree is two-dimensional and stores (`Salary`, `FedTaxRate`) points. We show the process of inserting x_2, x_3 (i.e. performing step Ⓐ and step Ⓑ from Example 3.3) and a range search for points whose `Salary` ≥ 4000 and `FedTaxRate` < 22. In the range tree, both the primary tree (which stores `Salary`) and the secondary tree (which stores `FedTaxRate`) are binary search trees. Leaf nodes store the inserted data, and each internal node stores the* *smallest value in its right subtree. Each node in the primary tree is linked to a secondary tree, which stores the `FedTaxRate` value of all the points present in the subtree rooted at this node. When we insert each point, we find the insert position in the primary tree, create a leaf node to store the inserted value, and an internal node to connect the new leaf node and the leaf node at the insert position. We also update all the secondary trees for nodes in the path from the root to the insert position. When we perform the range search, we look for nodes whose values lie in the range by preorder traversal. For instance, the search* $\mathbf{L} = (4000, -\infty), \mathbf{U} = (\infty, 22)$ *is performed by going from root node i to the left child node ii. Since the `Salary` value stored in node ii is in the range, we go to the secondary tree linked to it. Finally, we find node iii whose `FedTaxRate` value is less than* 22 *and return true.*

As another example, suppose we also insert x_4 *in the tree and search for a point where* `Salary` ≤ 7000 *and* `FedTaxRate` > 20. *At node II, we go to its secondary tree instead of traversing its descendants since all the points stored in the subtree rooted at node II have* `Salary` ≤ 5000, *which are within the search range for* `Salary`. *At node III, we return true since the minimum* `FedTaxRate` *stored in its right subtree is* 22, *which is greater than* 20.

3.3.4 Correctness and Complexity

Correctness Next, we show the correctness of Algorithm 2, as stated by the following lemma.

Lemma 3.5. *Algorithm 2 correctly determines whether a homogeneous DC* φ *is satisfied.*

Proof. We first show that Algorithm 2 is correct when φ only contains equality predicates. In this case, it is sufficient to determine whether there exist two distinct tuples x_1 and x_2 such that $\pi_{\mathsf{vars}_=(\varphi)}(x_1) = \pi_{\mathsf{vars}_=(\varphi)}(x_2)$. The hash table H stores a counter for each distinct $\pi_{\mathsf{vars}_=(\varphi)}(x)$ and increments it for each tuple $x \in R$ (Line 6). Thus, the algorithm will correctly return false as soon as some counter becomes greater than one and return true only if no such x_1, x_2 exists. Next, we consider the case when there exists at least one predicate with inequality. We show the proof for the case when all inequality predicate operators are $<$, i.e., all predicates in the DC are of the form $(x.A = x'.A)$ or $(x.A < x'.A)$. The proof for other operators is similar. We first state the following claim.

Claim 3.6. *Let w be the set of attributes that appear in the predicates with inequalities. Two tuples x_1 and x_2 in the same partition can form a violation iff $x_1(w) \prec x_2(w)$ or $x_2(w) \prec x_1(w)$, where the notation $x(w)$ denotes the projection, $\pi_w(x)$, of tuple x on attributes w.*

Here, $\prec$ is the standard coordinate-wise strict dominance checking operator. Claim 3.6 follows directly from the semantics of the operator under consideration and the definition of a violation. Suppose x is the tuple being inserted in the tree. Line 10 will query the range tree with $\mathbf{L} = (-\infty, \ldots, -\infty), \mathbf{U} = (x(v_1), \ldots, x(v_k))$ and $\mathbf{L}' = (x(v_1), \ldots, x(v_k))$, $\mathbf{U}' = (\infty, \ldots, \infty)$. In other words, the algorithm searches for a point in the tree such that x is strictly smaller or larger for all k coordinates. The existence of such a point implies there exists a pair that forms a violation.

If the orthogonal range search finds no point, Claim 3.6 tells us that x cannot form a violation with any tuple x' already present in the range tree. In each iteration of the loop, we insert one tuple into the range tree. Therefore, if x_1 and x_2 form a violation, it will be discovered when one of them (say x_2) is already inserted in the range tree and x_1 is being processed in the loop. This completes the proof. $\square$

Time and Space Complexity We next establish the running time of the algorithm. First, observe that if $k = 0$, then the algorithm takes $O(|R|)$ time since the for loop only performs a constant number of hash table operations. If $k \geq 1$, the algorithm performs one insertion and two Boolean orthogonal range search queries in each iteration of the for loop. Suppose the insertion time complexity, denoted by $I(n)$, is of the form[1] $\log^\alpha n$ and search time complexity is $T(n)$ when the data structure has n points in it. The running time can be bounded as:

$$\sum_{i=1}^{|R|} (\underbrace{\log^\alpha i}_{\text{insertion time}} + \underbrace{2 \cdot T(i)}_{\text{query time}}) < \int_1^{|R|+1} \log^\alpha i \, di + \int_1^{|R|+1} 2 \cdot T(i) \, di$$

$$= O(|R| \cdot \log^\alpha |R|) + \int_1^{|R|+1} 2 \cdot T(i) \, di$$

The integral in the second term in the equation above can be bounded by setting $T(i) = \log^k i$ or $T(i) = i^{1-1/k}$. In both cases, the second term evaluates to $O(|R| \cdot T(|R|))$. For space usage, note that the hash table takes a linear amount of space in the worst case. Thus, the space requirement of the tree data structure determines the space complexity. The main result can be stated as follows.

Theorem 3.7. *Algorithm 2 runs in time $O(|R| \cdot (I(|R|) + T(|R|)))$ and uses space $S(|R|)$ when using range tree or* kd*-tree with parameters as shown in Table 3.1.*

With range trees, the running time is $O(|R| \cdot \log^k |R|)$ and space usage is $O(|R| \log^{k-1} |R|)$; for kd-trees, the running time is $O(|R|^{2-\frac{1}{k}})$ and space requirement is $O(|R|)$.

3.3.5 Heterogeneous Predicates

In this subsection, we present adjustments to Algorithm 2 to handle heterogeneous predicates.

Example 3.8. *Continuing our study of **Tax** from Table 2.2, consider the following constraint: $\varphi_4 : \neg(x.\texttt{Salary} < x'.\texttt{FedTaxRate})$ which says that all values of the **Salary** column must be greater than or equal to any value of **FedTaxRate** column.*

[1]We use $\log^k N$ to mean $(\log N)^k$ and not iterated logarithms.

This is an example of a heterogeneous predicate that cannot be handled by Algorithm 2 (notice that both x and x' are referenced and the columns are different). Along the same lines, heterogeneous comparison constraints over date-related columns have also been found to be useful in our production settings. For example, over the TPC-H schema, Pena et al. (2022) identify the heterogeneous constraint $\neg(x.\texttt{Receiptdate} \geq x'.\texttt{Shipdate} \wedge x.\texttt{Shipdate} \leq x'.\texttt{Receiptdate})$ which represents the business logic that a new order is shipped only after all the previously shipped orders are received, i.e., the $[\texttt{Shipdate}, \texttt{Receiptdate}]$ intervals of the orders never overlap. These real-world scenarios underscore the need for going beyond simple homogeneous constraints considered in the previous section.

We now discuss how our algorithm can be extended to handle heterogeneous predicates of the form $x.\mathsf{A}$ op $x'.\mathsf{B}$, where op is $=, <, \leq, >$ or $\geq$. We note that equality predicate $x.\mathsf{A} = x'.\mathsf{B}$ is equivalent to $x.\mathsf{A} \leq x'.\mathsf{B} \wedge x.\mathsf{A} \geq x'.\mathsf{B}$ and by supporting inequalities, we can support heterogeneous equality predicates as well.

Now, let's look at how to handle heterogeneous inequality predicates. Suppose that φ has a predicate $x.\mathsf{C}$ op $x'.\mathsf{D}$. If $\mathsf{C} = \mathsf{D}$, then the predicate is homogeneous, and building a 1-dimensional range search data structure is enough. However, when $\mathsf{C} \neq \mathsf{D}$, we need to index in 2 dimensions (C and D). Additionally, we need to adjust our procedure for computing the two search range queries $(\mathbf{L}, \mathbf{U})$, $(\mathbf{L}', \mathbf{U}')$ to consider the different attributes present in the predicate. Let us look at an example.

Example 3.9. *Consider the DC φ_4 from Example 3.8 that checks that all the **Salary** values must be greater than any **FedTaxRate** value. We will create one 2-dimensional range search data structure in which we will store values of* (Salary, FedTaxRate)*. Suppose we are processing tuple x_2 with* $\mathsf{Salary} = 5000$ *and* $\mathsf{FedTaxRate} = 15$*. We first do a range search to check if there is a tuple with **Salary** that is less than* $x_2.\mathsf{FedTaxRate}$ *denoted as* $\mathbf{L} = (-\infty, -\infty), \mathbf{U} = (15, \infty)$*. Additionally, we check whether* *there is a tuple with* `FedTaxRate` *that is larger than* x_2.`Salary` *denoted as* $\mathbf{L}' = (-\infty, 5000), \mathbf{U}' = (\infty, \infty)$. *If any of the two range search queries return True, we have found a violation.*

Given this example, it becomes clear that we can easily extend Algorithm 2 to account for heterogeneous predicates by simply adjusting the procedure to create the two range search queries that are used to detect violations. Algorithm 3 shows the updated query generation procedure. The main idea is that if φ has a predicate $x.\mathsf{C}$ `op` $x'.\mathsf{D}$, then when we process a new tuple r, the upper-bound for attribute C is set to $r.\mathsf{D}$ in the forward check, and the lower-bound for attribute D is set to $r.\mathsf{C}$ in the inverted check (because we are comparing attribute C of x with attribute D of x' in the predicate). It is worth noting that this algorithm can be applied for homogeneous constraints as well. Note that, when $\mathsf{C} = \mathsf{D}$, we recover our original procedure presented in Algorithm 2. Thus, we can safely handle both types of constraints by simply replacing the `CreateRangeSearchQueries` function in Algorithm 2 with the one presented here. Finally, we also note that the new generalization also extends our algorithm to handle the case when attributes occur in more than one predicate.

3.3.6 Supporting Predicates with Disequalities

So far, we have discussed how to support homogeneous and heterogeneous predicates with equalities or inequalities. In this subsection, we show that it is possible to apply orthogonal range search techniques to support disequalities as well. This type of operator is quite important as it is required for specifying functional dependencies. We demonstrate that with the below example.

Example 3.10. *Consider the DC:* $\varphi_5 : \neg(x.\texttt{Zip} = x'.\texttt{Zip} \wedge x.\texttt{StateCode} \neq x'.\texttt{StateCode})$ *that represents the functional dependency* `Zip`$\rightarrow$`StateCode`.

Algorithm 3: Generalized range search query generation that covers both homogeneous and heterogeneous predicates with inequalities.

```
   Input  : A tuple $r$ from relation $R$, DC $\varphi$
   Output : Two range search queries (normal and inverted)
1  procedure CreateRangeSearchQueries($r$, $\varphi$)
2      $\mathbf{L}, \mathbf{L}' \leftarrow (-\infty, \ldots, -\infty), \mathbf{U}, \mathbf{U}' \leftarrow (\infty, \ldots, \infty)$    /* $\mathbf{L}, \mathbf{U}, \mathbf{L}', \mathbf{U}'$ are
        indexed by attributes of $R$ that appear in inequality
        predicates */
3      foreach inequality predicate $x.\mathsf{C}$ op $x'.\mathsf{D}$ in $\varphi$ do
4          if op is $<$ or $\leq$ then
5              $\mathbf{U}.\mathsf{C} \leftarrow r.\mathsf{D}$
6              $\mathbf{L}'.\mathsf{D} \leftarrow r.\mathsf{C}$
7          if op is $>$ or $\geq$ then
8              $\mathbf{L}.\mathsf{C} \leftarrow r.\mathsf{D}$
9              $\mathbf{U}'.\mathsf{D} \leftarrow r.\mathsf{C}$
10     return $\mathbf{L}, \mathbf{U}, \mathbf{L}', \mathbf{U}'$
```

The constraint contains a disequality predicate that cannot be handled by any of our previous algorithms.

We now discuss how we can support the verification of such predicates. Any predicate $x.\mathsf{A} \neq x'.\mathsf{B}$ can be written as a union of two predicates: $(x.\mathsf{A} < x'.\mathsf{B}) \vee (x.\mathsf{A} > x'.\mathsf{B})$. Therefore, a DC containing l predicates with op as $\neq$ can be equivalently written as a conjunction of 2^l DCs containing no disequality operator.

If the original homogeneous DC contains no inequality predicate, then it is possible to reduce the number of equivalent DCs from 2^l to 2^{l-1} (as is shown by Proposition 1). The idea is that a violation (x, x') is symmetric (i.e. (x', x) is also a violation) if the DC contains only equality and disequality predicates. Therefore, when converting a DC to an equivalent one that only have inequalities, it suffices to expand $(x.\mathsf{A} \neq x'.\mathsf{A})$ to just $(x.\mathsf{A} < x'.\mathsf{A})$ for one last disequality predicate instead of $(x.\mathsf{A} < x'.\mathsf{A}) \vee (x.\mathsf{A} > x'.\mathsf{A})$.

Proposition 1. *Given a homogeneous DC φ with only equality and l disequality predicates, there exists an equivalent conjunction of 2^{l-1} DCs that contain only equality and inequality predicates.*

Proof. Consider the constraint $\varphi : \neg(\phi \wedge x.\mathsf{A} \neq x'.\mathsf{A})$, where ϕ is a conjunction of homogeneous equality and disequality predicates. Let (q, r) be a violation to φ. Without loss of generality, we assume that $q.\mathsf{A} < r.\mathsf{A}$, and then (q, r) is also a violation to $\varphi' : \neg(\phi \wedge x.\mathsf{A} < x'.\mathsf{A})$. Since ϕ only contains equality and disequality predicates, (r, q) also satisfies ϕ by symmetricity, and therefore (r, q) is a violation to φ and $\varphi'' : \neg(\phi \wedge x.\mathsf{A} > x'.\mathsf{A})$. In fact, for any violation (r, q) to φ, one of (r, q) and (q, r) must violate φ' while the other violates φ''. Thus, we only need to check φ' for violations, which contains $l - 1$ disequality predicates and can be written as a conjunction of 2^{l-1} DCs containing no disequality predicates by logical equivalence. $\square$

3.3.7 Optimization for Single-Inequality Constraints

If a DC has homogeneous equality predicates and at most one predicate (homogeneous or heterogeneous) containing an inequality, then the verification can be done in linear time. The key idea is that for predicate containing inequality $x.\mathsf{A}$ `op` $x'.\mathsf{B}$, it is enough to keep track of the running minimum and maximum values for values seen in columns A and B respectively as we process the relation. To illustrate the idea, we use an example.

Example 3.11. *Consider the functional dependency φ_4 from Example 3.10 and the **`Tax`** table. Based on proposition 1, we can convert the disequality predicate into inequality to get the DC $\neg(x.\texttt{Zip} = x'.\texttt{Zip} \wedge x.\texttt{StateCode} < x'.\texttt{StateCode})$. Since both columns in the inequality predicate are the same (**`StateCode`**), we have $\mathsf{A} = \mathsf{B}$. Let us focus on rows of **`Tax`** with **`Zipcode`**=53703. We will keep track of the* min *and* max *value of **`StateCode`** for each of these rows. When x_2 is processed, we set* $\min = \max = x_2.\texttt{Zipcode} = 02$. *For x_3, we observe that since x_3 also has **`StateCode`**=02,* *the predicate* $x_2.\texttt{StateCode} < x_3.\texttt{StateCode}$ *is false and thus no violation is found. The values of* min *and* max *remain unchanged. Finally, for* x_4*, we also do not find a violation since* $x_4.\texttt{StateCode} = \texttt{02}$*. However, if there was a different row* x_4' *such that* $x_4'.\texttt{Zip} = 53703$ *and* $x_4'.\texttt{StateCode} = 03$*,* $x_2.\texttt{StateCode} < x_4'.\texttt{StateCode}$ *would become true and thus, we would have found a violation since all predicates are true for a tuple pair.*

Algorithm 4 shows the algorithm. Like the previous algorithms, we begin by partitioning the input into a hash table based on the equality predicates. Let the inequality predicate be $x.\mathsf{A}\ \texttt{op}\ x'.\mathsf{B}$. The main idea is to maintain the running minimum and maximum values for the Columns A and B for each partition of the input. Since the comparison is one-dimensional, it is sufficient to compare against the minimum (or maximum) value. The algorithm makes only one pass over the entire dataset and the overall time complexity is $O(|R|)$. While this optimization is simple, it has important implications. In particular, popular constraints such as functional dependencies (FD) are DCs that contain exactly one inequality predicate. Algorithm 4 recovers the standard linear time algorithm to verify FDs (Ibaraki et al., 1999).

3.3.8 Enumerating Violations

In this subsection, we discuss violation enumeration in our problem statement). To enumerate violations, we can simply replace the boolean range searches in Algorithm 2 with a function that outputs all points in the range. In addition, we describe an optimization that improves the time and space complexity of enumerating the DC violations. The observation is that, unlike verification, enumeration usually requires examining every tuple of the relation. Therefore, we can make use of sort-based optimizations. We demonstrate with an example.

Example 3.12. *Consider* $\varphi_3 : \neg(x\texttt{State} = x'\texttt{State} \wedge x\texttt{Salary} \leq x'\texttt{Salary} \wedge x\texttt{FedTaxRate} > x'\texttt{FedTaxRate})$ *for Table 2.2 with* ***State*=*Wisconsin*.** *The*

Algorithm 4: DC verification for DCs with row-homogeneous equality and one inequality predicate

```
Input  : Relation $R$, DC $\varphi$ containing equality predicates of form
         $x.\mathsf{C} = x'.\mathsf{C}$ (for some $C$) and one inequality predicate $p$ of form
         $x.\mathsf{A}$ op $x'.\mathsf{B}$
Output : True/False
1  $H \leftarrow$ empty hash table
2  foreach $x \in R$ do
3      $v \leftarrow \pi_{\mathsf{vars}_=(\varphi)}(x)$
4      if $v \notin H$ then
5          $H[v] \leftarrow (+\infty, +\infty, -\infty, -\infty)$        /* four-tuple represents
           $(\mathsf{min_A}, \mathsf{min_B}, \mathsf{max_A}, \mathsf{max_B})$ for $v$ */
6      if $(p.\mathsf{op} \in \{<, \leq\} \wedge H[v].\mathsf{min_A}$ op $x.\mathsf{B}) \vee (p.\mathsf{op} \in \{>, \geq\}$
        $\wedge H[v].\mathsf{max_A}$ op $x.\mathsf{B})$ then
7          return False
8      if $(p.\mathsf{op} \in \{<, \leq\} \wedge x.\mathsf{A}$ op $H[v].\mathsf{max_B}) \vee (p.\mathsf{op} \in \{>, \geq\}$
        $\wedge x.\mathsf{A}$ op $H[v].\mathsf{min_B})$ then
9          return False
10     $H[v].\mathsf{min_A} \leftarrow \min\{H[v].\mathsf{min_A}, x[\mathsf{A}]\}$             /* modify $\mathsf{min_A}$ */
11     $H[v].\mathsf{min_B} \leftarrow \min\{H[v].\mathsf{min_B}, x[\mathsf{B}]\}$             /* modify $\mathsf{min_B}$ */
12     $H[v].\mathsf{max_A} \leftarrow \max\{H[v].\mathsf{max_A}, x[\mathsf{A}]\}$             /* modify $\mathsf{max_A}$ */
13     $H[v].\mathsf{max_B} \leftarrow \max\{H[v].\mathsf{max_B}, x[\mathsf{B}]\}$             /* modify $\mathsf{max_B}$ */
14     return True
```

first step is to sort the three rows in increasing order of ***Salary****, which creates the ordering x_4, x_2, x_3. The key observation here is that once sorting has been performed, it is guaranteed that any tuple can only form a violation (wrt. the* ***Salary*** *predicate) with a tuple that appears after it in the sort order. At this point, the* ***Salary*** *attribute can be completely removed from consideration as the corresponding predicate will always be satisfied. We can now resume our usual processing by constructing a one-dimensional binary search tree on* ***FedTaxRate*** *attribute. We first insert x_4.*FedTaxRate $= 22$ *into the tree. For the next tuple in the order x_2, which has x_2.*FedTaxRate $= 15$*, we ask the tree to enumerate all tuples that have a value greater than* 15*, a standard operation on a binary search tree. We find the x_4 is such a tuple and thus* *report (x_4, x_2) as a violation to the user. Then, we insert x_2.*`FedTaxRate` *in the tree. Finally, x_3 is processed and we search for all values in the tree that with* ***`FedTaxRate`*** *greater than 20 and report (x_4, x_3) as a violation.*

Algorithm 5 shows the detailed steps for enumerating violations for a homogeneous DC. We begin by sorting the relation on a column that participates in some predicate that has an inequality operator (line 2). We can safely assume that the operator is either $<$ or $>$ since $\leq, \geq$ operator can be decomposed into two constraints: one containing only $=$ as the operator for the predicate and the other containing $<$ or $>$. The algorithm iterates over the dataset and uses the hash table H or range search data structure similarly to Algorithm 2. There are two important differences to note. First, we can omit the inverted range search. This is because the sort order guarantees that for any tuple x, a violation can be formed only with tuples that appear before x in the sort order. Second, since the dataset has already been sorted on one of the columns with inequality predicates, the number of dimensions of the point inserted in the range search data structure is reduced by one compared to Algorithm 2. The same idea can also be applied to any DC containing at least one homogeneous predicate and using the column referenced in the predicate for sorting. For DCs containing only heterogeneous predicates, we use a sort-merge style approach. We also have the following complexity result with this optimization.

Theorem 3.13. *Let k be the number of columns that occur in predicates containing an inequality. Then, our enumeration algorithm runs in time $O(|R| \cdot (I(|R|) + T(|R|) + \log |R|) + K)$ to enumerate K violations and uses space $S(|R|)$ when using range tree or* **kd***-tree with parameters as shown in Table 3.1 with the number of dimensions as $k - 1$.*

Algorithm 5: Violation enumeration for DCs.

```
Input  : Relation $R$, Homogeneous DC $\varphi$ containing at least one row
         homogeneous inequality predicate
Output : DC violations
1  $H \leftarrow$ empty hash table
2  sort $R$ on a column (say $\mathsf{C}$) that participates in an inequality predicate in
     ascending order if the predicate operator is $<$ (and descending for $>$)
3  $\ell \leftarrow |\mathsf{vars}(\varphi) \setminus \{\mathsf{C} \cup \mathsf{vars}_=(\varphi)\}|$
4  $\mathsf{Temp} \leftarrow \emptyset$
5  foreach $x_i \in R$ do
6      $v \leftarrow \pi_{\mathsf{vars}_=(\varphi)}(x_i)$
7      if $v \notin H$ then
8          if $\ell \neq 0$ then
9              $H[v] \leftarrow$ new OrthogonalRangeSearch()
10         else
11             $H[v] \leftarrow \emptyset$
12     if $\ell \neq 0$ then
           /* Two rows can only form a violation if they satisfy
              the predicate containing C.  Therefore, no
              violation can be formed for tuples that have the
              same value for attribute C.                        */
13         if $x_i.\mathsf{C} = x_{i+1}.\mathsf{C}$ then
14             $\mathsf{Temp} \leftarrow \mathsf{Temp} \cup x_i$
15         else
16             if $\mathsf{Temp} \neq \emptyset$ then
17                 foreach $r \in \mathsf{Temp}$ do
18                     $H[v]$.INSERT($\pi_{\mathsf{vars}(\varphi)\setminus\{\mathsf{C}\cup\mathsf{vars}_=(\varphi)\}}(r)$)
19                 $\mathsf{Temp} \leftarrow \emptyset$
20             $H[v]$.INSERT($\pi_{\mathsf{vars}(\varphi)\setminus\{\mathsf{C}\cup\mathsf{vars}_=(\varphi)\}}(x_i)$)
21         $\mathbf{L}, \mathbf{U} \leftarrow$ SeachRange($x_i$)            /* From Algorithm 2 */
22         $\mathcal{L} \leftarrow H[v]$.ENUMERATE($\mathbf{L}, \mathbf{U}$)
23         output $(x, x_i)$ for each $x \in \mathcal{L}$
24     else
25         output $(x, x_i)$ for each $x \in H[v]$
26         $H[v] \leftarrow H[v] \cup x_i$
```

3.4 Denial Constraint Discovery Based on Rapidash

Algorithm 6: DC discovery

```
   Input  : Relation $R$
   Output : List of exact, minimal DCs
1  $\mathcal{L} \leftarrow$ NEW LIST() /* stores exact, minimal DCs */
2  $k \leftarrow 0$
3  while $k \leq |vars(R)|$ do
4      $k \leftarrow k + 1$
5      foreach candidate $\varphi$ formed using size $k$ subset of $vars(R)$ do
           /* MINIMAL borrowed from Chu et al. (2013) */
6          if MINIMAL($\mathcal{L}, \varphi$) and NOTPRUNED($\mathcal{L}, \varphi$) and VERIFY($R, \varphi$) then
7              output $\varphi$
8              $\mathcal{L}$.APPEND($\varphi$)
9  return processed $\mathcal{L}$ using implication test from Chu et al. (2013)
   procedure NOTPRUNED($\mathcal{L}, \varphi$)
10     foreach $\varphi' \in \mathcal{L}$ do
11         $p_1, \ldots, p_m \leftarrow$ predicates in $\varphi'$
12         foreach $j \in [m]$ do
13             if $\varphi$ contains $\{p_i\}_{i \neq j}$ and $\neg p_j$ then
14                 return False
15     return True
```

In this section, we introduce our DC discovery method, a combination of our algorithm for DC verification and lattice searches. To find all exact, minimal DCs, we use a lattice-based approach where we generate candidate DCs and leverage the verification algorithm to verify whether the DC holds.

Similar to prior works on functional dependency discovery (Huhtala et al., 1999), we start with singleton sets of attributes and traverse larger sets in a level-by-level fashion. For each set of attributes, we generate candidate DCs by generating all possible predicates. Then, we apply the verification algorithm to check whether the DC holds over the input. If the DC is true,

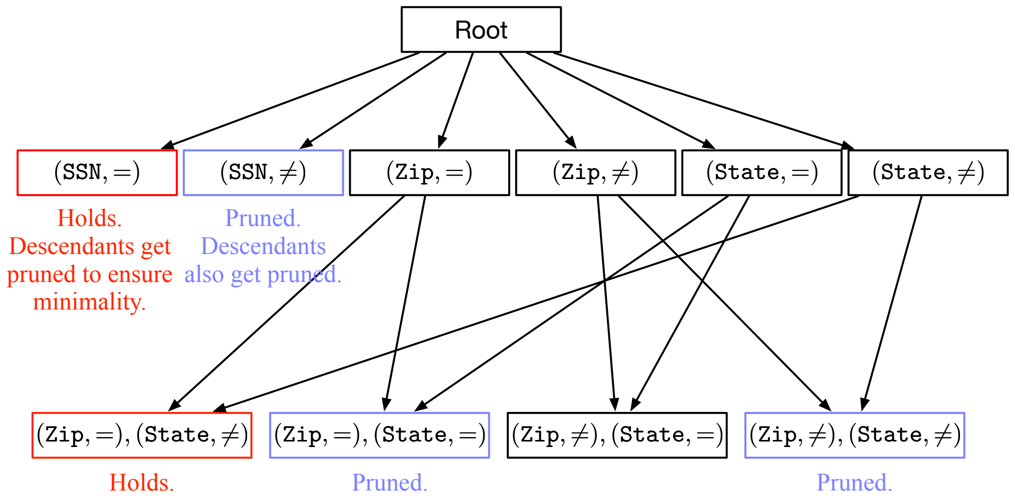


Figure 3.3: The search space of homogeneous DC discovery on Table 2.2 when we only consider SSN, Zip, State and limit the number of predicates to two. To save space, we only show (attribute, operator) of the predicates in the DCs. For example, (SSN, =) represents $\neg(x.\texttt{SSN} = x'.\texttt{SSN})$ and (Zip, =), (State, ≠) represents $\neg(x.\texttt{Zip} = x'.\texttt{Zip} \wedge x.\texttt{State} = x'.\texttt{State})$. DCs marked as "Holds" have been verified to be true and DCs marked as "Pruned" are candidates implied by DCs that are true.

we output the DC to the user and store it in a list $\mathcal{L}$, which is used to check the minimality of a candidate DC. We also prune candidate DCs whose validity is implied by others. When $\neg(\bigwedge_{i\in[m]} p_i)$ is verified, for any $j \in [m]$, we remove all DCs containing $\{p_i\}_{i\neq j} \cup \{\neg p_j\}$ from the search space.

Example 3.14. *Figure 3.3 shows an example search sub-space showing the first two levels of the lattice. Level one (with incoming arrows from the* ***Root****) contains all DCs over a single column and level two contains candidates generated from predicates in level one. Consider the DC* $\varphi_1 : \neg(x.\texttt{SSN} = x'.\texttt{SSN})$ *and suppose that it holds. Once this DC is verified, it is added to* $\mathcal{L}$ *and does not contribute any new candidates in the search space. The next candidate* $\varphi_1' : \neg(x.\texttt{SSN} = x'.\texttt{SSN})$ *is superfluous as it is guaranteed to be false due to logical implication. We also remove all* *descendants of φ'_1 because they will be equivalent to other DCs. For instance, $\neg(x.\texttt{SSN} \neq x'.\texttt{SSN} \wedge x.\texttt{Zip} = x'.\texttt{Zip})$ is equivalent to $\neg(x.\texttt{Zip} = x'.\texttt{Zip})$, and the latter has already been checked on level one. On level two, the first candidate $\neg(x.\texttt{Zip} = x'.\texttt{Zip} \wedge x.\texttt{State} \neq x'.\texttt{State})$ holds and added to $\mathcal{L}$, which helps us prune the two other candidates in the second level which are marked in the figure.* $\square$

3.5 Experimental Evaluation

In this section, we report the results of our experimental evaluation. In particular, we seek to answer the following questions:

1. What is the performance improvement (time and space) of the RAPIDASH verification and enumeration algorithm compared to FACET on open-source datasets used in prior work (Pena et al., 2021)?

2. What is the performance and scalability of RAPIDASH and FACET over large-scale real-world production datasets with complex constraints?

3. How does the performance and scalability of RAPIDASH-based DC discovery compare to existing solutions?

Table 3.2: Size of the datasets.

Dataset	Number of Rows	Number of Columns
Tax	1M	12
TPC-H	1M	12
D_1	50M	28
D_2	25M	28

Table 3.3: List of denial constraints used in experiments for each dataset.

	Denial Constraint
Tax	c_1: $\neg(x.\texttt{AreaCode} = x'.\texttt{AreaCode} \wedge x.\texttt{Phone} = x'.\texttt{Phone})$
Tax	c_2: $\neg(x.\texttt{ZipCode} = x'.\texttt{ZipCode} \wedge x.\texttt{City} \neq x'.\texttt{City})$
Tax	c_3: $\neg(x.\texttt{State} = x'.\texttt{State} \wedge x.\texttt{HasChild} = x'.\texttt{HasChild} \wedge x.\texttt{ChildExemp} \neq x'.\texttt{ChildExemp})$
Tax	c_4: $\neg(x.\texttt{State} = x'.\texttt{State} \wedge x.\texttt{Salary} > x'.\texttt{Salary} \wedge x.\texttt{Rate} < x'.\texttt{Rate})$
TPC-H	c_5: $\neg(x.\texttt{Customer} = x'.\texttt{Supplier} \wedge x.\texttt{Supplier} = x'.\texttt{Customer})$
TPC-H	c_6: $\neg(x.\texttt{Receiptdate} \geq x'.\texttt{Shipdate} \wedge x.\texttt{Shipdate} \leq x'.\texttt{Receiptdate})$
TPC-H	c_7: $\neg(x.\texttt{ExtPrice} > x'.\texttt{ExtPrice} \wedge x.\texttt{Discount} < x'.\texttt{Discount})$
TPC-H	c_8: $\neg(x.\texttt{Qty} = x'.\texttt{Qty} \wedge x.\texttt{Tax} = x'.\texttt{Tax} \wedge x.\texttt{ExtPrice} > x'.\texttt{ExtPrice} \wedge x.\texttt{Discount} < x'.\texttt{Discount})$
D_1	$\varphi_{1,1}$:$\neg(x.\mathsf{A} = x'.\mathsf{A} \wedge x.\mathsf{B} = x'.\mathsf{B} \wedge x.\mathsf{C} \neq x'.\mathsf{C} \wedge x.\mathsf{D} \neq x'.\mathsf{D})$
D_1	$\varphi_{1,2}$:$\neg(x.\mathsf{C} = x'.\mathsf{C} \wedge x.\mathsf{E} = x'.\mathsf{E} \wedge x.\mathsf{F} = x'.\mathsf{F} \wedge x.\mathsf{G} \neq x'.\mathsf{G} \wedge x.\mathsf{H} \neq x'.\mathsf{H})$
D_1	$\varphi_{1,3}$:$\neg(x.\mathsf{B} = x'.\mathsf{B} \wedge x.\mathsf{I} = x'.\mathsf{I} \wedge x.\mathsf{J} = x'.\mathsf{J} \wedge x.\mathsf{K} \neq x'.\mathsf{K} \wedge x.\mathsf{L} \neq x'.\mathsf{L})$
D_1	$\varphi_{1,4}$:$\neg(x.\mathsf{A} = x'.\mathsf{A} \wedge x.\mathsf{I} = x'.\mathsf{I} \wedge x.\mathsf{M} > x'.\mathsf{M} \wedge x.\mathsf{N} \neq x'.\mathsf{N})$
D_2	$\varphi_{2,1}$:$\neg(x.\mathsf{A} = x'.\mathsf{A} \wedge x.\mathsf{B} = x'.\mathsf{B} \wedge x.\mathsf{C} \geq x'.\mathsf{C} \wedge x.\mathsf{D} \leq x'.\mathsf{D} \wedge x.\mathsf{E} \leq x'.\mathsf{E} \wedge x.\mathsf{F} \geq x'.\mathsf{F} \wedge x.\mathsf{G} > x'.\mathsf{G})$
D_2	$\varphi_{2,2}$:$\neg(x.\mathsf{A} \neq x'.\mathsf{A} \wedge x.\mathsf{B} = x'.\mathsf{B} \wedge x.\mathsf{H} \leq x'.\mathsf{H} \wedge x.\mathsf{F} \geq x'.\mathsf{F} \wedge x.\mathsf{G} \geq x'.\mathsf{G})$
D_2	$\varphi_{2,3}$:$\neg(x.\mathsf{A} = x'.\mathsf{A} \wedge x.\mathsf{I} \neq x'.\mathsf{I} \wedge x.\mathsf{D} \leq x'.\mathsf{D} \wedge x.\mathsf{G} \geq x'.\mathsf{G} \wedge x.\mathsf{J} = x'.\mathsf{J})$
D_2	$\varphi_{2,4}$:$\neg(x.\mathsf{C} \leq x'.\mathsf{C} \wedge x.\mathsf{D} \leq x'.\mathsf{D} \wedge x.\mathsf{K} = x'.\mathsf{K})$

3.5.1 Experimental Setting

Datasets, DCs, and Hardware. We perform experiments on both open-source datasets and production datasets. For open-source datasets, we use the Tax, TPC-H with a total of 8 DCs that were identified by prior works (Bleifuß et al., 2017; Pena et al., 2019, 2021) to be representative of what is usually seen in production settings (as defined by experts or discovered from data). We also use two production datasets (related to banking records and document shipping) from Microsoft customers interested in applying DC verification and discovery on their data. Each dataset contains a mix of categorical, numeric, and datetime columns. For both production datasets, we pick 3 DCs by taking a 10% sample of each dataset and discover DCs that are true over the sample. The fourth DC (denoted by $\varphi_{i,4}$ for dataset D_i) holds over the full dataset. Table 3.2 shows the size of the datasets and Table 3.3 lists a total of 16 DCs over all datasets that we use in our experiments[2]. Note that constraints c_5, c_6 are examples of heterogeneous constraints. We ran all experiments on an Intel(R) Xeon(R) W-2255 CPU @ 3.70GHz machine with 128GB RAM running Windows 10 Enterprise (version 22H2). All of our experiments are executed over a single core and in the main memory setting.

Evaluation Metrics. We perform experiments for both DC verification and enumeration using Rapidash and Facet and report the end-to-end running time and space consumption. For enumeration performance, we report the total time to count and return the number of all the violations (same approach as Facet (Pena et al., 2021)) to avoid output materialization cost and focus on understanding the intrinsic hardness. For DC discovery, we compare against Hydra (Bleifuß et al., 2017) and DCFinder (Pena et al., 2019) in runtime. We use the same predicate space as defined by Pena et al. (2019). All reported running times are the trimmed mean of five independent executions after the dataset has been loaded in memory.

Implementation. Similar to prior work, Rapidash is implemented in Java. We use a standard implementation of orthogonal range trees (referred to as Rapidash($\perp$)) and kd-trees (referred to as Rapidash(kd)). Since we were unable to obtain the original Facet source code from Pena et al. (2021), we implemented Facet ourselves in Java using the Metanome infrastructure from Bleifuß et al. (2017) and Pena et al. (2019), with all optimizations enabled as described in the Facet paper (Pena et al., 2021). We manually verified that the performance of our implementation is in line with the numbers reported by Pena et al. (2021) accounting for hardware differences[3]. Further, to ensure a fair comparison, for DC verification, we ensure that Facet execution terminates as soon as the first violation is found.

[2]The column names in the DCs have been omitted due to security and privacy concerns.

[3]There is some deviation to be expected since the Tax dataset used by Pena et al. (2021) has not been publicly released. We could only obtain a different version of the dataset (available at met (2023)) generated for a different publication.

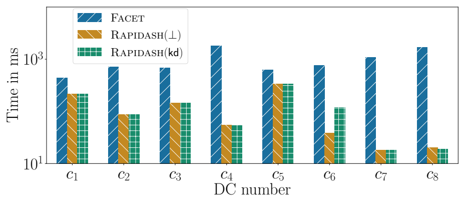
3.5.2 DC Verification on Open-Source Datasets

In this subsection, we compare against FACET on open-source datasets for DC verification. Figure 3.4 shows the running time for FACET and RAPIDASH. Let us fix our attention to Figure 3.4a. Our first observation is that for c_1 and c_5 DCs (which contain only equality and thus, both FACET and RAPIDASH take a provably linear amount of time in theory), RAPIDASH is faster by 2× for verification. This is because FACET requires cardinality estimation for all columns involved in the predicates, followed by creating the refinements which require iterating over the dataset again. RAPIDASH requires no statistics and iterates over the dataset only once. Constraints c_6, c_7, and c_8 all have a large number of violations (on the order of several hundred million). For these constraints, both versions of RAPIDASH are up to 84× faster than FACET since RAPIDASH can find a violation after only looking at a few tuples in the dataset while FACET requires expensive computation. In fact, for c_6 and c_7, we observed that the size of all ordered pairs[4] after just the first refinement (which are inequality predicates) is 1.2B and 3.6B respectively.

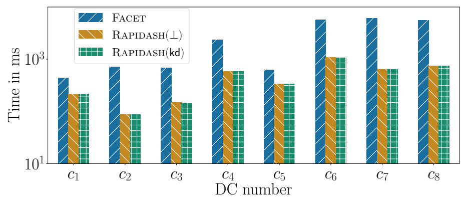
The speedup improvement obtained by RAPIDASH also extends to the violation enumeration problem (Figure 3.4b). Although there is no early termination possible for counting, RAPIDASH still performs up to an order of magnitude better due to our improved algorithms. FACET performance, on the other hand, degrades further since the last refinement cannot be stopped early as FACET requires *all* refinements to be complete in order to begin counting. Note that both RAPIDASH($\perp$) and RAPIDASH(kd) have the same performance numbers since all constraints contain at most two inequality predicates and thus, both range trees and kd-trees degenerate into a simple binary search tree.

Figure 3.5 shows the space usage for both verification and violation enumeration. For FACET, space usage is calculated using the cardinality

[4]Given an ordered pair $(\mathsf{tids}_1, \mathsf{tids}_2)$, its size is defined as $|\mathsf{tids}_1| + |\mathsf{tids}_2|$

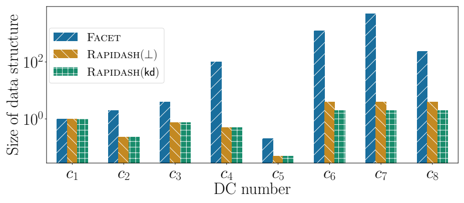


(a) Running time (in ms) for DC verification on Tax and TPC-H.

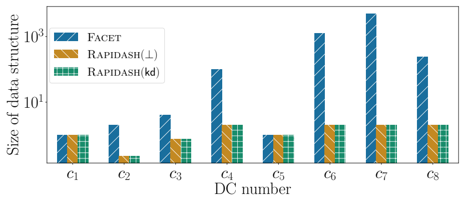


(b) Running time (in ms) for violation enumeration on Tax and TPC-H.

Figure 3.4: Running time for DC verification on open-source datasets.



(a) Space usage for DC verification on Tax and TPC-H.



(b) Space usage for violation enumeration on Tax and TPC-H.

Figure 3.5: Space usage for DC verification on open-source datasets.

(in millions) of cluster pairs constructed and we use the number of nodes in the tree constructed for Rapidash(⊥) and Rapidash(kd). For every DC, Rapidash uses significantly lower space compared to Facet. The high space usage of Facet is directly attributed to the size of ordered pairs generated after refining predicates. The largest gap is observed for c_6, c_7, and c_8, which is expected since the DCs have a lot of violations, making refinement computation and storage expensive. On the other hand, for each constraint, Rapidash(kd) requires only (provably) linear amount of memory and this behavior can be directly observed in practice as well.

Scalability Figure 3.6 shows the behavior of the algorithms on TPC-H with varying the cardinality. Let us take constraint c_8 as an example. As the cardinality increases, the speedup obtained by Rapidash compared to Facet increases from 7.5× for 1M to 13.86× when the dataset size is 4M. This suggests that the running time of Facet grows at a non-constant rate

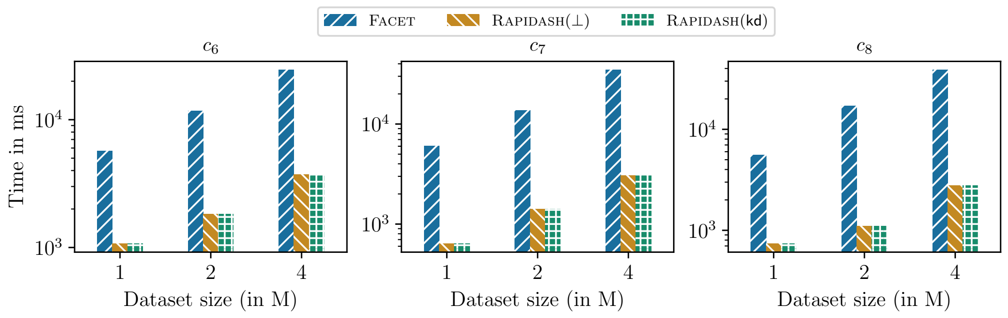


Figure 3.6: Running time (in milliseconds) for violation enumeration on the TPC-H dataset with varying cardinality.

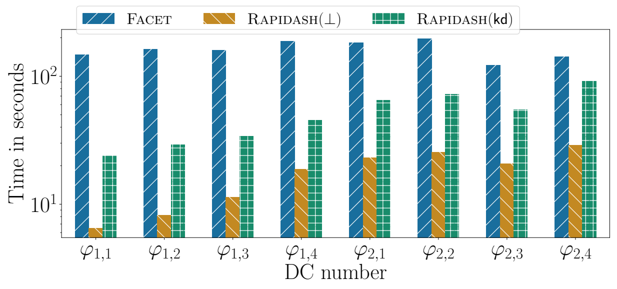


Figure 3.7: Running time (in seconds) for DC verification on production datasets.

compared to Rapidash which grows in line with expectation.

3.5.3 DC Verification on Production Datasets

In this subsection, we evaluate on production datasets for DC verification. Figure 3.7 shows the running time (in log scale) of Rapidash for all production datasets and DCs. The speedup obtained by our algorithm is close to an order of magnitude and up to 40×. Compared to Facet, both algorithms perform significantly better on all DCs. Rapidash(⊥) performs better than Rapidash(kd) on all DCs. This is not surprising since using kd-trees for orthogonal range search is more expensive as shown by complexity analysis. However, we can see that Rapidash(kd) is still faster

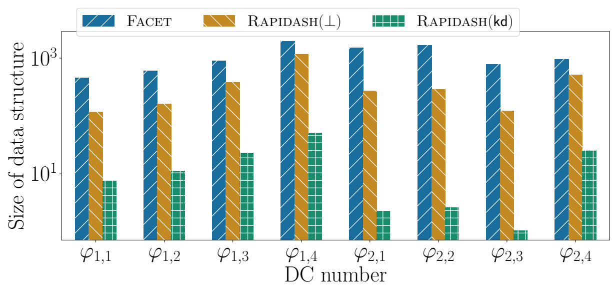


Figure 3.8: Space requirement of different algorithms for DC verification on production datasets

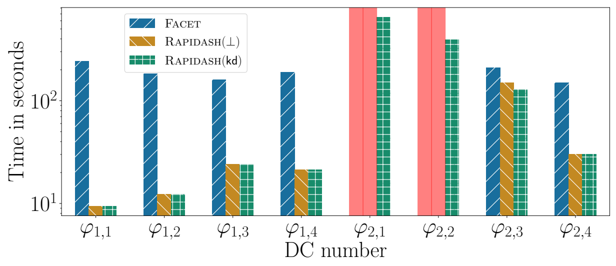


Figure 3.9: Running time (in seconds) for violation enumeration on production datasets. Red (solid) bars for $\varphi_{2,1}$ and $\varphi_{2,2}$ denote an out-of-memory error.

than Facet by up to 20×.

The speedup obtained by Rapidash can be attributed to two reasons. First, Rapidash can terminate as soon as a violation is discovered, as opposed to Facet, which cannot do early termination in general. The second reason is that Rapidash does not require any expensive materialization as opposed to the ordered pair generation that is done by Facet. We also measure the space usage of all the systems (as is shown in Figure 3.8). For all DCs, Facet requires $1.4 - 8\times$ more space than Rapidash(⊥). In addition, Rapidash(kd) was a further order of magnitude lower in its space requirement compared to Rapidash(⊥).

Figure 3.9 shows the running time for violation enumeration. For most constraints, we observe a similar trend in the running time as we saw for verification. The most interesting observation is that for constraints $\varphi_{2,1}$ and $\varphi_{2,2}$, both FACET and RAPIDASH($\perp$) fail to complete due to Java out-of-memory error, as both constraints contain a large number of inequality predicates. However, RAPIDASH(kd) can finish the computation in about 10 minutes, thanks to its linear memory use guarantee.

3.5.4 DC Discovery

Performance We evaluate the performance of our DC discovery algorithm (RAPIDASH(disc)) in comparison to HYDRA and DCFINDER using the production datasets. We'd like to note that our datasets are much larger than those used in prior work (HYDRA and DCFINDER were evaluated on datasets consisting of up to 1M rows). We run the experiments with a time limit of 48 hours. For all datasets, both HYDRA and DCFINDER could not finish the computation of the evidence set within the time limit for any dataset. This is not surprising since the evidence-set can be super linear in the size of the dataset and has an exponential dependency on the number of columns. Therefore, even after spending a lot of computing resources, the user does not get any information at all about whether there even exists a DC or not. In contrast, RAPIDASH(disc) was able to discover all constraints over a single attribute (i.e. $k = 1$) within 10 minutes for all datasets. Constraints over single attributes are already interesting since it includes single-column primary keys, finding whether columns that are empty, or sorted in a particular order. All constraints are generated over pairs of attributes ($k = 2$) within one hour of starting the discovery process. Further, since RAPIDASH(disc) continuously outputs DCs, the user can still get useful information even if the algorithm isn't allowed to run to completion.

Scalability micro-benchmark To understand the scaling behavior, we create a micro-benchmark where we vary the number of rows and columns in dataset D_1 and run DCFINDER[5] to generate all constraints over at most

[5]We omit a microbenchmark with HYDRA since DCFINDER is known to be faster Pena

three columns. We intentionally keep the dataset size small to ensure that DCFinder can actually terminate. Figure 3.10 shows the scalability wrt. to varying the cardinality of D_1. DCFinder is faster than our solution when the dataset size is 10^5 rows but its running time grows very quickly as the dataset size increases, demonstrating the super-linear running time empirically. This growth is entirely due to the evidence-set computation step. Rapidash(disc) on the other hand has a much slower growth in running time. The same behavior was also observed when varying the number of columns but keeping the cardinality as 5×10^5, as shown in Figure 3.11. Even for only 25 columns in a small dataset, the evidence set computation becomes a blocker. Note that the jump in DCFinder running time when going from 10 to 15 columns is larger than when going from 15 to 20 columns. This is because the evidence set computation is sensitive to column cardinality and whether it is numeric or categorical. Recall that categorical columns only admit $=, \neq$ as operators but numerical columns can have any operator in the predicate. When numerical columns are added, not only do they generate 6 row-level homogeneous predicates but also column-level and heterogenous predicates. This leads to a blowup in predicate space which in-turn makes the evidence set larger.

3.6 Conclusion and Future Work

In this chapter, we studied the problem of DC verification and discovery. We presented Rapidash, a DC verification algorithm with near-linear time complexity with respect to the dataset size that leverages prior work on orthogonal range search. We also developed an anytime DC discovery algorithm that does lattice search based on our verification algorithm. Unlike previous works, our discovery algorithm eliminates the reliance on the

et al. (2019)

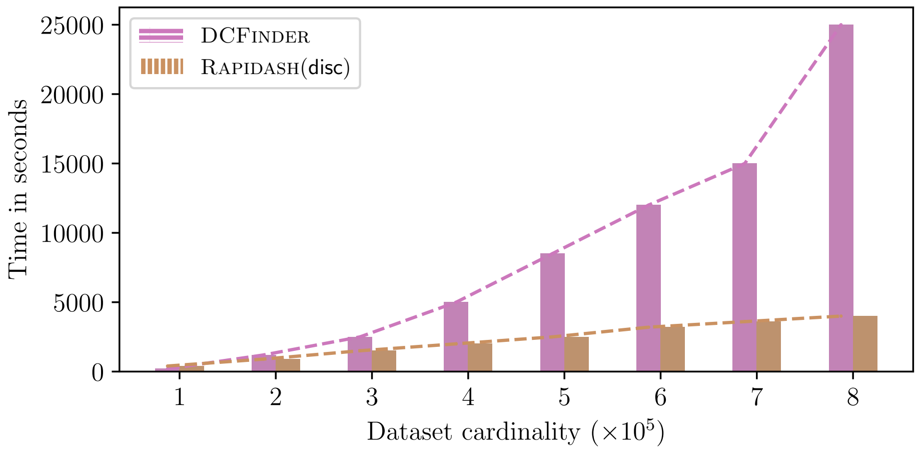


Figure 3.10: Running time of DCFinder vs. Rapidash(disc) for varying cardinality of D_1 with 15 columns.

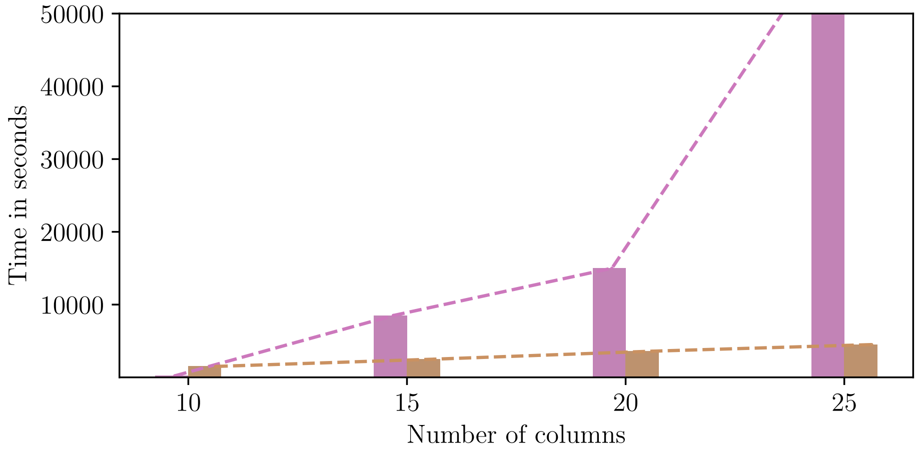


Figure 3.11: Running time of DCFinder vs. Rapidash(disc) for varying number of columns with $|D_1| = 5 \cdot 10^5$. Criss-cross hashed bar means experiment could not complete in 24 hours.

construction of evidence sets, which can be computationally expensive. Through empirical evaluation, we demonstrated that our DC verification algorithm is faster than the state of the art by an order of magnitude on large-scale production datasets. Our DC discovery algorithm is able to output valid DCs incrementally whereas existing methods fail to provide any useful information.

4 GUARDING AGAINST CORRUPTED DATA IN TABULAR DATA DURING LEARNING AND INFERENCE

4.1 Introduction

Data quality assessment is critical in all phases of the machine learning (ML) life cycle. Both in the training and deployment (inference) stages of ML models, erroneous data can have devastating effects. In the training stage, errors in the data can lead to biased ML models (Koh et al., 2018; Schelter et al., 2018; Breck et al.; Polyzotis et al., 2019), i.e., models that learn wrong decision boundaries. In the deployment stage, errors in the inference queries can result in wrong predictions, which in turn can be harmful for critical decision making systems (Breck et al.; Steinhardt et al., 2017b). ML pipelines need reliable data quality assessment during both training and inference to be robust to data errors.

We focus on tabular data and seek to develop a simple, plug-and-play approach to guard against corrupted data (including adversarially corrupted data) during both training and inference in ML pipelines. During training, our goal is to identify and filter corrupted examples from the data used to train a model, while during deployment, our goal is to flag erroneous query points to a pre-trained ML model, i.e., points that due to noise will result in incorrect predictions of the ML model. This chapter introduces a unified solution to guard against corrupted data for both the training and deployment stages of ML models.

Guarding against corrupted data in ML pipelines exhibits many challenges. First, detecting corrupted examples in the training data can be a hard exercise that requires developing methods that go beyond standard outlier detection mechanisms (Xue et al., 2010). Data poisoning techniques (Steinhardt et al., 2017b; Koh et al., 2018; Muñoz-González et al., 2017; Biggio et al., 2012) attack models by adding a small fraction of adversarially crafted poisoned data to the training set. Any reliable mechanism that filters corruptions from a training data set should not only remove easy to detect outliers but also hard to detect poisoned data.

Second, online-detection of inference queries that yield a model misprediction due to corruption requires not only knowledge of the data quality, but also knowledge of the tolerance of the trained ML model to corruptions. The reason is that not all corruptions will flip the prediction of a trained ML model and different models exhibit different degrees of robustness to corruption. Moreover, adversarial noise may target specific subsets of the data or classes in the ML pipeline (Koh et al., 2018). For this reason, online-filtering of corrupted inference queries requires a method that takes both the downstream model and data quality into account.

The above challenges require rethinking the current solutions for identifying errors in data. The majority of outlier detection methods in the statistical literature (Xue et al., 2010; Liu et al., 2008; Chen et al., 2001) and error detection methods in the database literature (Heidari et al., 2019; Mahdavi et al., 2019) are not effective against adversarial corruptions (Koh et al., 2018). More advanced methods are required to defend against adversarial corruptions (Steinhardt et al., 2017b). However, current methods are typically limited to real-valued data (Diakonikolas et al., 2017) and focus either on training (Diakonikolas et al., 2019b) or inference (Roth et al., 2019; Grosse et al., 2017) but not both. Finally, recent techniques for data validation in ML pipelines that are deployed in industrial settings (Schelter et al., 2018; Breck et al.) rely on user-specified rule- or schema-based quality assertions evaluated over batches of data and it is unclear if they can support on-the-fly, single point validation, which is required during inference.

We present Picket, a framework for safeguarding against corrupted data during both the training and deployment stages of ML pipelines. Picket can be used in an offline manner to validate data that will be used for training but can also be used in an online manner to safeguard against corruptions for on-the-fly queries at inference time. We empirically demonstrate that PICKET outperforms both state-of-the-art outlier detection mechanisms such as Robust Variational Autoencoders (Eduardo et al., 2020), and state-of-the-art methods for detection of adversarial corruption attacks during inference (Roth et al., 2019; Grosse et al., 2017). Our work makes the following technical contributions:

Self-Attention for Tabular Data PICKET is built around PICKETNET, a new deep learning-based encoder for mixed-type tabular data. PICKETNET can model mixtures over numerical, categorical, and even text-based entries of limited length (e.g., descriptions). The goal of PICKETNET is to learn the characteristics of the distribution governing the non-corrupted data on which the ML pipeline operates and it is used in PICKET to distinguish between clean data points and corrupted ones. The architecture of PICKETNET builds upon the general family of Transformer networks (Vaswani et al., 2017) and introduces a new multi-head self-attention module (Vaswani et al., 2017) over tabular data. This module follows a stream-based architecture that is able to capture not only the dependencies between attributes at the schema-level but also the statistical relations between cell values—it follows a schema stream and value stream architecture. We find that compared to schema-only models, PICKETNET's two-stream architecture is critical for obtaining accurate predictions across diverse data sets.

Robust Training over Arbitrary Corruptions We show how to learn a PICKETNET model without imposing any extra labeling burden to the user and by operating directly on potentially corrupted data (i.e., we do not not require access to clean data to learn the non-corrupted data distribution). We achieve that by using a robust self-supervised training approach that is robust to corrupted data points (including adversarial points). As with standard self-supervision, the context captured in the data is used as the supervision signal. The training procedure for PicketNet monitors the reconstruction loss of tuples in the input data over early training iterations and uses related statistics to identify suspicious data points. These points are then excluded from subsequent iterations during training.

A Plugin to ML Pipelines We demonstrate how Picket can serve as a "plugin" that safeguards against corrupted data in different ML pipelines during both training and inference. We evaluate Picket over multiple data sets with different distributional characteristics and consider different types and magnitudes of corruption, ranging from simple random noise to adversarial attacks that explicitly aim to harm the performance of downstream ML models. We find that Picket provides a reliable mechanism for detecting data corruptions in ML pipelines: Picket consistently achieves an area under the receiver operating characteristic curve (AUROC) score of above or close to 80 points for detecting corrupted data across different types of noise and ML models.

In addition, we analyze the effects of data corruption on mean estimation theoretically to promote the understanding of the relationship between data corruption and learning. We consider *robust mean estimation* under *coordinate-level* corruptions (either missing entries or value replacements). We assume an *adversarial corruption model* for which a given data set generated from an unknown distribution can have up to α-fraction of its coordinates corrupted adversarially, i.e., the adversary can strategically hide or modify individual coordinates of samples. The goal is to find an estimate $\hat{\mu}$ of the true mean μ of the data set that is accurate even in the worst case. We present an information-theoretic analysis of corruption under coordinate-level adversaries. Our results show that one must exploit the structure of the data (i.e., dependencies between features) to achieve information-theoretically optimal error for mean estimation.

4.2 Preliminaries

Data Corruption Models We consider data corruption due to random, systematic, and adversarial noise.

1. *Random noise* is drawn from some unknown distribution that does not depend on the data. Random noise is not predictable and cannot be replicated in a repeatable manner. While many ML models are robust to purely random noise during training, high-magnitude random noise can still lead to false predictions, and hence is of interest to our study.

2. *Systematic noise* depends on values in the data and leads to repeated errors in data samples. This type of noise biases the distribution of the data. Systematic noise can skew the distribution of the data, and this bias can potentially harm the performance of an ML model depending on the importance of the corrupted features to the downstream prediction task.

3. *Adversarial noise* contaminates the data to explicitly mislead ML models and harm their performance. At training time, adversarial noise corrupts the training points to force a model to learn a bad decision boundary; at test time, adversarial noise corrupts the input queries in a manner that will lead to a false prediction by the model. It usually depends on the data and the target model, although some types of adversarial noise may work well across different models.

Dealing with Corrupted Data in ML The most common approach to deal with corrupted data during training is to identify corrupted samples and remove them from the training set. This process is referred to as *filtering*. Given a training data set $\mathcal{D}$, filtering identifies a set of clean data points $\mathcal{C} \subseteq \mathcal{D}$ to be used for training. Common filtering mechanisms rely on outlier detection methods (Liu et al., 2008; Chen et al., 2001; Eduardo et al., 2020). In addition, recent filtering methods focus on adversarial corruptions over real-valued data (Steinhardt et al., 2017b; Diakonikolas et al., 2017). Finally, there are data validation modules for ML platforms (Polyzotis et al., 2019;

Schelter et al., 2018; Breck et al.) that rely on user-defined rules and simple statistics to check the quality of data batches. The statistical tests used by these methods are subsumed by outlier detection methods and user-defined quality rules are out of the scope of this work. For inference, apart from outlier detection methods, there are methods that accept or reject inference queries by using statistical tests that compare the query to clean data (Grosse et al., 2017) or by considering variations in a model's internal data representation (Roth et al., 2019). We also consider the online detection of inference queries that result in wrong predictions due to corruption.

Self-Supervision In self-supervised learning systems (Devlin et al., 2018; Su et al., 2020), the learning objective is to predict part of the input from the rest of it. A typical approach to self-supervision is to mask a portion of the input, and then let the model reconstruct the masked portion based on the unmasked parts. Through self-supervised learning, a model learns to capture dependencies between different parts of the data. Self-supervised learning is a subset of unsupervised learning in a broad sense since it does not need human supervision.

Multi-Head Self-Attention Models with multi-head self-attention mechanism learn representations for structured inputs e.g., a tuple or a text sequence, by capturing the dependencies between different parts of the inputs (Vaswani et al., 2017). One part can pay different levels of attention to other parts of the same structured input. For example, consider the text sequence "the dog wears a white hat", the token "wears" pays more attention to "hat" than "white" although "white" is closer in the sequence. The attention mechanism can also be applied to tuples that consist of different attributes (Wu et al., 2020). Multi-head self-attention takes an ensemble of different attention functions, with each head learning one.

We provide a brief review of the multi-head self-attention model (Vaswani et al., 2017). Let $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$ be the embedding of a structured input with T tokens. Each token $x^{(i)}$ is transformed into a query-key-value triplet

($q_i = \boldsymbol{W_Q} x^{(i)}$, $k_i = \boldsymbol{W_K} x^{(i)}$, $v_i = \boldsymbol{W_V} x^{(i)}$) by three learnable matrices $\boldsymbol{W_Q}$, $\boldsymbol{W_K}$ and $\boldsymbol{W_V}$. The query q_i, key k_i, and value v_i are real-valued vectors with the same dimension d. The output of a single head for the i^{th} token is $\sum_{j=1}^{T} w_{ij} v_j$, a weighted sum of all the values in the sequence, where $w_{ij} = \text{softmax}((q_i^T k_1, q_i^T k_2, \ldots, q_i^T k_T)/\sqrt{d})_j$. The attention $x^{(i)}$ pays to $x^{(j)}$ is determined by the inner product between q_i and k_j. Multiple heads share the same mechanism but have different transformation matrices. The outputs of all the heads are concatenated and transformed into the final output by an output matrix W_O, which is also learnable.

4.3 Overview of Picket

We review PICKET's functionalities during the training and inference phases of a ML pipeline. An overview diagram of PICKET's core components and functionalities is shown in Figure 4.1. The corresponding pseudo-code is shown in Algorithm 7.

Guarding against Corrupted Data in Training

We consider a tabular data set $\mathcal{D}$ with N training examples. Let x be a sample (tuple) in $\mathcal{D}$ with T attributes. These attributes correspond to the features that are used by the downstream model. For each x we denote x^* its clean version; if x is not corrupted then $x = x^*$.

We assume that $\mathcal{D}$ contains clean and corrupted samples and that the fraction of corrupted samples is always less than half. The goal of PICKET is to filter out the corrupted samples in $\mathcal{D}$ and construct a *clean* set of examples $\mathcal{C} \subseteq \mathcal{D}$ to be used for training a downstream model. Without loss of generality, we assume that PICKET performs filtering over $\mathcal{D}$ once. This process can be repeated for data batches over time. We next describe how we construct $\mathcal{C}$ in PICKET.

PICKET follows the next steps: First, PICKET learns a self-supervised PICKETNET model that captures how data features are distributed for

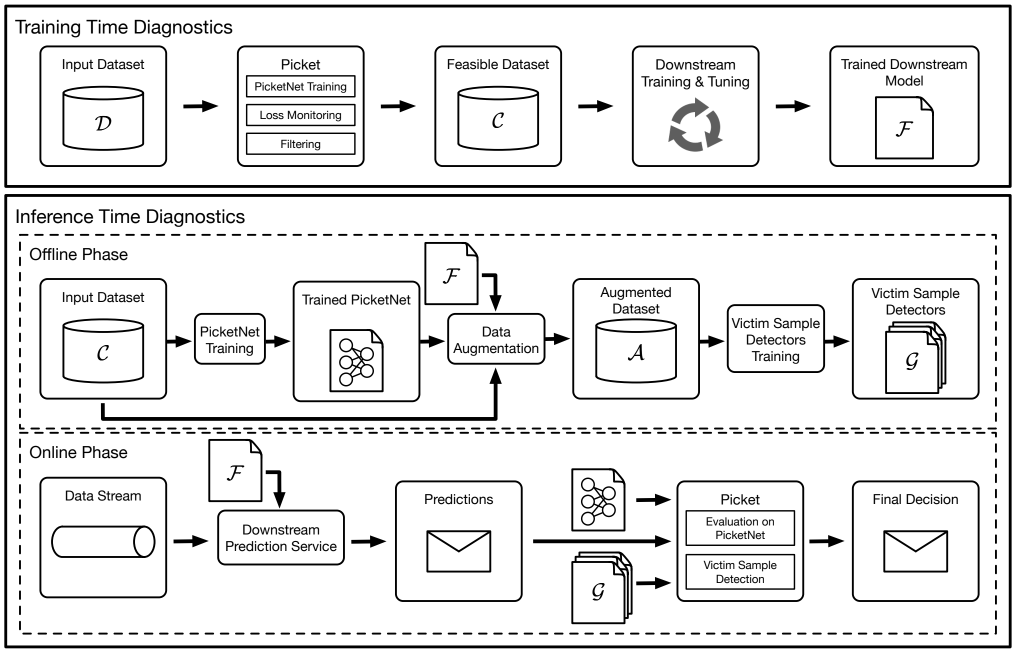


Figure 4.1: The key components of a typical machine learning pipeline with Picket.

the clean samples. Picket does not require human-labeled examples of corrupted or clean data. During training, Picket records the reconstruction loss across training epochs for all points in $\mathcal{D}$. After training of PicketNet, we analyze the reconstruction loss progression over the first few training epochs to identify points in $\mathcal{D}$ that are corrupted. Set $\mathcal{C}$ is constructed by removing these corrupted points from $\mathcal{D}$. We also proceed with training PicketNet on $\mathcal{C}$. The the pre-trained PicketNet model is used to detect corruptions during inference.

Guarding against Corrupted Data in Inference

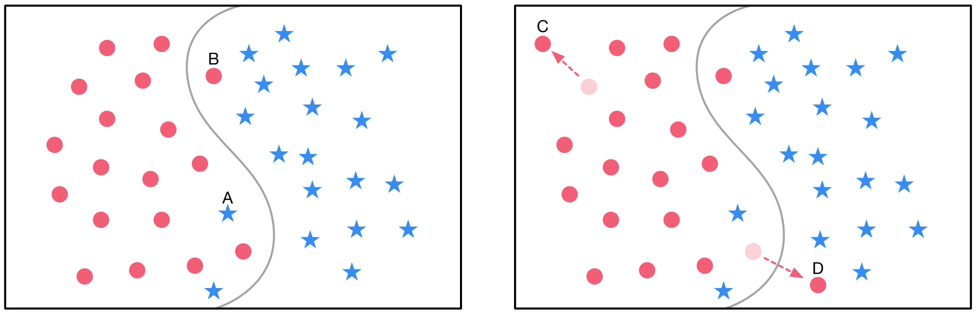
We consider a trained model $\mathcal{F}$ that serves inference queries over data points with the same T attributes as in the training phase of the ML pipeline. We define a *victim sample* to be a point x such that $\mathcal{F}(x^*) = y$ but $\mathcal{F}(x) \neq y$, i.e., the input sample is corrupted and it gets misclassified due to corruption. We show an example that illustrates the difference between non-victim and

Algorithm 7: Picket in a typical ML pipeline

```
1  Training Time:
2  Input: dataset $\mathcal{D}$, downstream model type and configuration $\mathcal{I}_{\text{config}}$;
3  Output: filtered dataset $\mathcal{C}$, trained downstream model $\mathcal{F}$;
4  $\mathcal{C}$ = PicketNetTrainingAndEarlyFiltering($\mathcal{D}$);
5  $\mathcal{F}$ = DownstreamModelTraining($\mathcal{C}, \mathcal{I}_{\text{config}}$);
6  ____________________________________________
7  Inference Time (Offline Phase):
8  Input: filtered dataset $\mathcal{C}$, trained downstream model $\mathcal{F}$;
9  Output: trained PicketNet $\mathcal{M}$, victim sample detectors $\mathcal{G}$;
10 $\mathcal{M}$ = PicketNetTraining($\mathcal{C}$);
11 augmented dataset $\mathcal{A}$ = DataAugmentation($\mathcal{M}, \mathcal{F}$);
12 $\mathcal{G}$ = VictimSampleDetectorTraining($\mathcal{A}$);
13 ____________________________________________
14 Inference Time (Online Phase):
15 Input: data stream $\mathcal{D}_{\text{stream}}$, trained downstream model $\mathcal{F}$, trained
     PicketNet $\mathcal{M}$, victim sample detectors $\mathcal{G}$;
16 Output: final prediction $y_{\text{prediction}}$;
17 raw prediction $y_{\text{raw}}$ = DownstreamPrediction($\mathcal{D}_{\text{stream}}, \mathcal{F}$);
18 $y_{\text{prediction}}$ = PicketVictimDetection($\mathcal{D}_{\text{stream}}, y_{\text{raw}}, \mathcal{M}, \mathcal{G}$)
```

victim samples according to our definition in Figure 4.2. The goal of Picket is to solve the following problem: Given an already-trained classifier $\mathcal{F}$, for each sample x that comes on the fly, we want to tell if it is a good sample or it is a victim sample and will be misclassified due to corruption, i.e., we want to detect if $\mathcal{F}(x) \neq \mathcal{F}(x^*)$. We assume access to data set $\mathcal{C}$ and model $\mathcal{M}$ output by Picket for safeguarding during the training phase of the ML pipeline in hand. We then adopt a two-phase approach, *offline* and *online* phase, to solve the aforementioned problem.

We now focus on the offline phase. Given the trained model $\mathcal{F}$, data set $\mathcal{C}$, and model $\mathcal{M}$, we learn a victim-sample detector for each class in the prediction task in hand. Each victim-sample detector is a binary classifier that detects if an input sample x will be misclassified by $\mathcal{F}$ due to corruption.



(a) Before Corruption (b) After Corruption

Figure 4.2: An example of non-victim and victim samples. The grey line is the decision boundary of a binary classifier that separates the red circles and the blue stars in the two-dimensional space. (a) Before corruption, some samples (e.g. point A and B) get misclassified, but they are not victim samples because they are clean, and the misclassification is due to the limitation of the model. Those samples should be handled by model analytics, and are out of the scope of our framework. (b) After corruption, two samples are shifted by the noise (point C and D). C is not a victim sample since the noise injected does not affect the correctness of classification. D is a victim sample because it gets misclassified due to noise.

The victim-sample detectors operate on an extended feature set: Beyond the original T features of the inference query x we add T additional features corresponding to the reconstruction loss obtained by masking each feature in turn and applying model $\mathcal{M}$ to predict it back.

During the online phase, we use model $\mathcal{M}$ and the victim-sample detectors over a stream of incoming inference queries to identify victim samples. Picket performs the following: for each incoming point x, Picket evaluates classifier $\mathcal{F}$ on x to obtain an initial prediction $\mathcal{F}(x)$. Picket also uses $\mathcal{M}$ to compute the reconstruction-loss vector for the features of x. The extended feature vector containing the original features of x and the reconstruction loss features are given as input to the victim sample detector for the class that corresponds to the prediction $\mathcal{F}(x)$. Using this input, the detector identifies if point x corresponds to a victim sample. If the point is not marked as suspicious the final prediction is revealed downstream, otherwise the inference query is flagged.

4.4 The PicketNet Model

PICKET uses a new two-stream multi-head self-attention model to learn the distribution of tabular data. We refer to this model as PICKETNET. The term *stream* refers to a path in a neural network that focuses on a specific view of the input data. For example, standard attention mechanism is one stream that learns value-based dependencies between the parts of the input data (see Section 4.2). Combining multiple streams, where each stream focuses on learning a different view of the data, has been shown to achieve state-of-the-art results in natural language processing tasks (Yang et al., 2019) and computer vision tasks (Simonyan and Zisserman, 2014) but has not been applied on tabular data. PICKETNET introduces a new two-stream model for tabular data and proposes a robust, self-supervised training procedure for learning this model.

4.4.1 Model Architecture

PICKETNET contains two streams: a *schema stream* and a *value stream*. The schema stream captures schema-level dependencies between attributes of the data, while the value stream captures dependencies between specific data values. A design overview of PICKETNET is shown in Figure 4.3 with details of the two streams. The input to the network is a mixed-type data tuple x with T attributes $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$.

The first level of PICKET obtains a numerical representation of tuple x. To capture the schema- and value-level information for x, we consider two numerical representations for each attribute i: (1) a real-valued vector that encodes the information in value $x^{(i)}$, denoted by $I_i^{(0)}$, and (2) a real-valued vector that encodes schema-level information of attribute i, denoted by $P_i^{(0)}$.

For example, a tuple with two attributes is represented as $I_1^{(0)}P_1^{(0)}I_2^{(0)}P_2^{(0)}$. To convert $x^{(i)}$ to $I_i^{(0)}$, PICKETNET uses the following process: The encoding for each attribute value $x^{(i)}$ is computed independently. We consider (1) categorical, (2) numerical, and (3) textual (short-text) attributes. For categorical attributes, we use a learnable lookup table to get the embedding for each value in the domain. This lookup table is learned jointly with all other components of PICKETNET. For numerical attributes, we keep the raw value as one dimension and pad the other dimensions with zeros. For text attributes, we use fastText (Bojanowski et al., 2017) encoding and apply SIF (Arora et al., 2017) to aggregate the embedding of the words in a cell. The initial embedding vectors $I_i^{(0)}$ are inputs to the value-level stream.

Each vector $P_i^{(0)}$ serves as a *positional encoding* of the attribute associated with index i. Positional encodings are used to capture high-level dependencies between attributes. $P_i^{(0)}$ is consistent for attribute i in all examples, i.e., it does not change as the values in different examples vary. Hence, it captures common dependencies at the schema level. Each $P_i^{(0)}$ corresponds to a trainable vector that is initialized randomly and is fed as input to the schema stream in every self-attention layer.

We now describe subsequent layers of our model. These layers consider the two attention streams and form a stack of n self-attention layers. The output of the previous layer serves as the input to the next layer. Self-attention layer l takes the value vector $I_i^{(l)}$ and positional encoding $P_i^{(l)}$ to learn a further representation for attribute i and its value $x^{(i)}$. After each attention layer, the outputs of the two streams are aggregated and fed as input to the value-level stream of the next layer, while the schema stream still takes as input the positional encoding. The output of the value stream $H_i^{(l)}$ and that of the schema stream $G_i^{(l)}$ are computed as:

$$H_i^{(l)} = \mathbf{MHS}(\boldsymbol{Q} = L_Q(I_i^{(l)}), \boldsymbol{K} = L_K(I_{j=1,\ldots,T}^{(l)}), \boldsymbol{V} = L_V(I_{j=1,\ldots,T}^{(l)}))$$

$$G_i^{(l)} = \mathbf{MHS}(\boldsymbol{Q} = L_Q(P_i^{(l)}), \boldsymbol{K} = L_K(P_{j=1,\dots,T}^{(l)}), \boldsymbol{V} = L_V(I_{j=1,\dots,T}^{(l)}))$$

where **MHS** represents the multi-head attention function followed by a feed-forward network and L_Q, L_K, L_V are linear transformations that transform the input into query, key, or value vectors by the corresponding weight matrices for $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$. Finally, $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are matrices formed by packing the query, key and value vectors from their inputs.

The difference between the two streams is that the query in the schema stream corresponds to the positional encoding, therefore it learns higher-level dependencies. For the value stream the input to the next level is the sum of the outputs from the two streams: $I_i^{(l+1)} = H_i^{(l)} + G_i^{(l)}$; for the schema stream the input to the next level $P_i^{(l+1)}$ corresponds to a new positional encoding that does not depend on the previous layers. If layer l is the last layer, $O_i = I_i^{(l+1)}$ is the final representation for attribute value $x^{(i)}$.

4.4.2 Training Process

We learn PicketNet using the noisy data set $\mathcal{D}$ without any human-labeled examples of corrupted or clean data. The training of PicketNet follows a self-supervised learning objective.

Self-Supervised Training For each point in $\mathcal{D}$, we mask one of the attributes and then try to reconstruct it based on the values of the other attributes in the same tuple. Other attributes may still contain noisy data or missing values. The attributes are masked in turn following an arbitrary order. The training is also multi-task since the reconstruction of each attribute forms one learning task.

We use different types of losses for the three types of attributes to quantify the quality of reconstruction. Consider a sample x whose original value of attribute i is $x^{(i)}$. If $x^{(i)}$ is numerical, its a one-dimensional value, and hence, the reconstruction of the input value is a *regression task*: We apply a simple neural network on the output O_i to get an one-dimensional reconstruction $\hat{x}^{(i)}$,

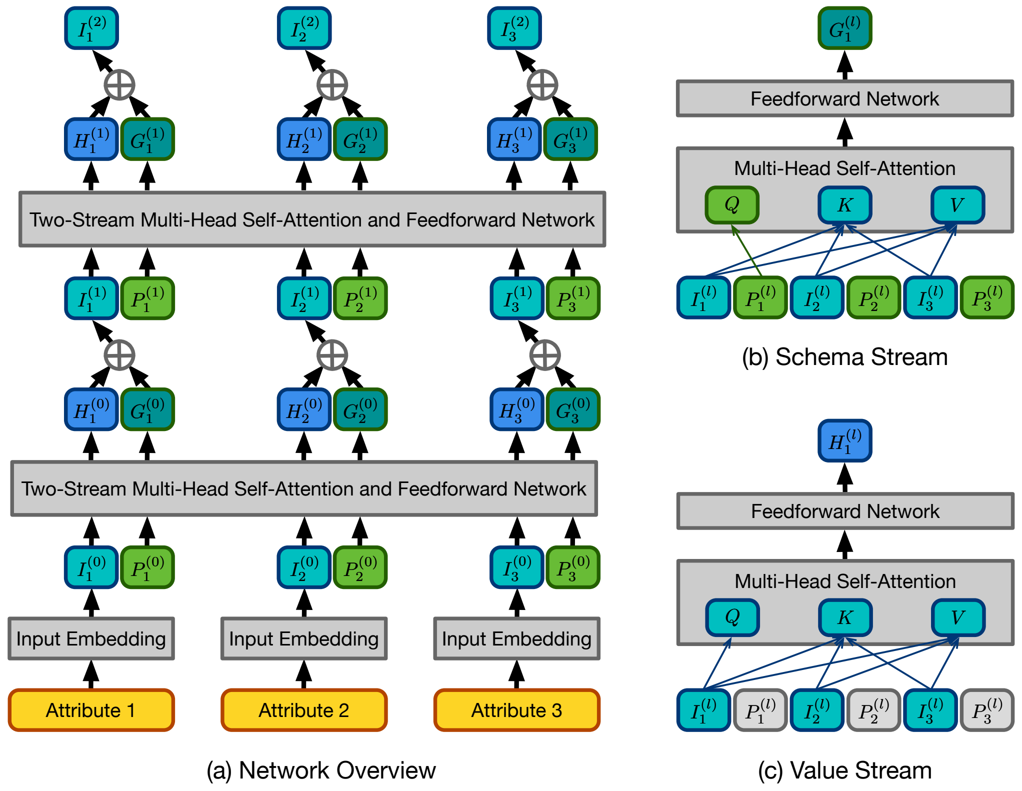


Figure 4.3: (a) Overview of the two-stream multi-head self-attention network. (b) An illustration of the schema stream for the first attribute.(c) An illustration of the value stream for the first attribute.

and use the mean squared error (MSE) loss: $\mathrm{MSE}(x^{(i)}, \hat{x}^{(i)}) = (x^{(i)} - \hat{x}^{(i)})^2$.

For categorical or text-based attributes we use the cross-entropy loss. Consider a tuple x and its attribute i. For its attribute value $x^{(i)}$ let $I_i^0(x^{(i)})$ be the base-embedding before passing through the attention layers of PICKETNET, and $O_i(x_{\text{mask}})$ the contextual encoding of value $x^{(i)}$ after pushing tuple x_{mask} (with attribute i masked) through PICKETNET. Given tuple x, we randomly select a set of other values Z_i from the domain of attribute i. We consider the training loss associated with identifying $x^{(i)}$ as the correct completion value from the set of possible values $\{x^{(i)}\} \cup Z_i$. To compute the training loss we use the cosine similarity between $O_i(x_{\text{mask}})$

and the input encoding $I_i^0(r)$ for each $r \in \{x^{(i)}\} \cup Z_i$, then we apply the softmax function over the similarities and calculate the cross-entropy (CE) loss:

$$\text{CE}(x, Z_i; i, M) = -\log\left(\frac{\exp(\text{sim}(I_i^{(0)}(x^{(i)}), O_i(x_{\text{mask}})))}{\sum_{r \in \{x^{(i)}\} \cup Z_i} \exp(\text{sim}(I_i^{(0)}(r), O_i(x_{\text{mask}})))}\right)$$

where $\text{sim}(a, b)$ is the cosine similarity between a and b.

Loss-based Filtering to Ensure Robust Training The data used to learn PicketNet can be corrupted, in which case self-supervised learning might lead to a biased model due to the presence of noise. To make learning robust to noisy input, we use a loss-based filtering mechanism to detect and ignore corrupted data during training of a PicketNet model. The process we use follows the next steps:

1. Warm-start PicketNet by training over $\mathcal{D}$ for E_1 epochs.

2. Train PicketNet over $\mathcal{D}$ for E_2 epochs and, for each sample in $x \in \mathcal{D}$, record the *epoch-wise average loss* $\text{Loss}_i(x)$ for each attribute i, $i = 1, 2, \ldots, T$.

3. For each sample, aggregate the losses attribute-wise by $\text{Loss}(x) = \sum_{i=1}^{T} \text{Loss}_i(x)/\text{Median}_{\mathcal{D}}(\text{Loss}_i(\cdot))$ where $\text{Median}_{\mathcal{D}}$ computes the median over all points in $\mathcal{D}$.

4. Put a sample into set $\mathcal{D}'$ if its aggregated loss is less than δ_{low} or greater than δ_{high}, where δ_{low} and δ_{high} are pre-specified thresholds; $\mathcal{D}'$ is the set of samples to be removed.

5. Train PicketNet over $\mathcal{C} = \mathcal{D} \setminus \mathcal{D}'$ until convergence.

The thresholds δ_{low} and δ_{high} control the sensitivity of the detection. In practice, we can set δ_{low} and δ_{high} based on a relatively clean validation set. A common strategy is setting their values based on the validation set so

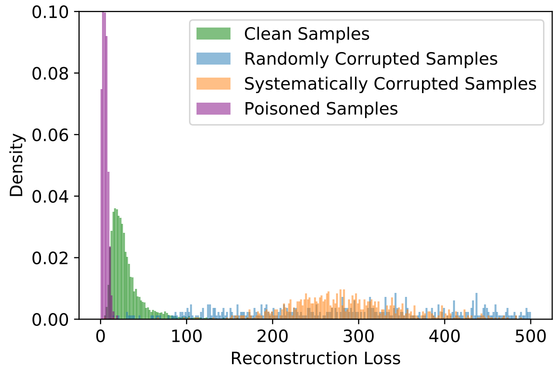


Figure 4.4: Distribution of the reconstruction loss (early in training) for different types of clean and noisy samples.

that the false positive rate (FPR) is under some value (e.g. 5%). When a relatively clean validation set is not available, the thresholds can be set based on the histogram of the reconstruction loss. Filtering out abnormal peaks and low density tails in the histogram is a natural strategy, and we validate the effectiveness of it in Section 4.6.4.

When we do the attribute-wise aggregation, we normalize the loss of each attribute by dividing with the median of it to bring different types of losses to the same scale. The normalized loss characterizes how large the loss is relative to the average level loss in that attribute. We use the median since it is robust against extremely high or low values, while the mean can be significantly shifted by them.

The filtering is two-sided because randomly or systematically corrupted samples and adversarially crafted (poisoned) samples have different behaviors during the early training stage. Outliers with random or systematic noise are internally inconsistent and thus have high reconstruction loss in the early training stage of PicketNet. However, poisoned samples tend to have unusually low reconstruction loss. The reason is that poisoned data tend to be concentrated on a few locations to be effective and appear normal, as is pointed out by Koh et al. (Koh et al., 2018). Such concentration forces deep networks such as PicketNet to fit quickly and therefore the reconstruction loss in the early stage is lower than that of the clean samples. We confirm this hypothesis experimentally. Figure 4.4 shows the distribution of the reconstruction loss for 1) clean, 2) randomly and systematically corrupted, and 3)poisoned samples for a real-world dataset. The noise used in this illustrative example follows the procedure described in Section 4.6.1. The three distributions have notable statistical distances. Hence, we need to remove samples with high loss to capture random or systematic corruptions, and samples with abnormally low loss to defend against poisoning attacks.

4.5 Detecting Data Corruptions

The reconstruction loss of PicketNet is the key to training time and inference time detection of corrupted examples. We now provide more details on these functions of Picket.

Detecting Corrupted Training Data Detection of corrupted training data follows directly from the training procedure of PicketNet described in the previous section (Section 4.4.2). Given an ML pipeline that aims to learn a model $\mathcal{F}$ for a downstream task, we 1) first train a PicketNet model over the data considered for training and 2) only use the data points that are not filtered during the training of PicketNet to train the downstream model $\mathcal{F}$. This approach allows us to apply Picket to any training pipeline regardless of the downstream model. Effectively, the pre-trained PicketNet is used as an encoder capable to detect outlier points. As we show in Section 4.6, our approach is effective across different types of ML models. For adversarially poisoned training data, we find that using Picket as a filter before training, allows us to train downstream ML models that exhibit similar performance to that of models trained on non-corrupted data.

Victim Sample Detection for Inference We now describe how we construct the victim sample detectors to safeguard against corruptions during inference for a trained classifier $\mathcal{F}$ (see Section 4.3). For each class y in the downstream classification task, we build a detector $\mathcal{G}_y$ to identify victim samples, i.e., samples that $\mathcal{F}$ will misclassify due to corruption of the feature values. The detectors are binary classifiers. In our experiments, we use logistic regression models with regularization parameter 1.0 as detectors.

At inference time, the victim sample detectors are deployed along with the downstream model $\mathcal{F}$ and a pre-trained PicketNet model $\mathcal{M}$. Whenever a sample x comes, the downstream model gives the prediction $f(x)$. The corresponding detector $\mathcal{G}_{f(x)}$ takes into account x and the feature-wise reconstruction loss (not aggregated) from $\mathcal{M}$ and decides if x should be marked as suspicious.

We learn the victim-sample detectors by using a data set with artificially corrupted data points. We describe this process below; notice that no human-labeled data is required. We start from the filtered data $\mathcal{C}$ output by Picket during the training phase of the ML pipeline. We first apply the already-trained classifier $\mathcal{F}$ on all points in $\mathcal{C}$ and obtain a subset of points for which $\mathcal{F}$ returns the correct prediction, i.e., $f(x) = y$. We denote this subset $\mathcal{C}_{\text{cor}}$. Moreover, we partition $\mathcal{C}_{\text{cor}}$ into sets $\mathcal{C}^y_{\text{cor}}$, one for each class y of the downstream prediction class. For each partition, we use the points in $\mathcal{C}^y_{\text{cor}}$ to construct artificial victim samples and artificial noisy points for which $\mathcal{F}$ returns the correct prediction despite the injection of noise. We discuss the artificial noise we inject in detail in Section 4.6.1. Let VS^y and NS^y_{cor} be the set of artificial victim samples and the set of noisy but correctly classified sample generated from $\mathcal{C}^y_{\text{cor}}$ respectively. To construct these two data sets we select a random point x^* from $\mathcal{C}^y_{\text{cor}}$ and inject artificial noise to obtain a noisy version x; we then evaluate $f(x)$ and if $f(x) = f(x^*) = y$ we assign the generated point x to NS^y_{cor} otherwise we assign it to VS^y. We iteratively perform the above process for randomly selected points in $\mathcal{C}^y_{\text{cor}}$ until we populate sets VS^y and NS^y_{cor} with enough points such that $|\mathcal{C}^y_{\text{cor}}| = |NS^y_{\text{cor}}| = 0.5 \times |VS^y|$. Given these three sets, we construct a new augmented data set $\mathcal{A}^y = \mathcal{C}^y_{\text{cor}} \cup NS^y_{\text{cor}} \cup VS^y$. We extend the feature vector for each point in $x \in \mathcal{A}^y$ by concatenating it with the reconstruction loss vector obtained after passing each point through the trained PicketNet $\mathcal{M}$. We also assign to it a positive label (indicative that we will obtain a correct prediction) if it originated from $\mathcal{C}^y_{\text{cor}}$ or NS^y_{cor} and a negative label (indicating that we will obtain a wrong prediction) if it originated from VS^y. The output of this procedure is the training data for the victim sample detector $\mathcal{G}_y$. We repeat the above process for each class y.

Ideally, the artificial noise that we inject should have the same distribution as that in the real-world case. However, it is impossible to know the exact noise distribution in advance. A practical solution is injecting mixed-type artificial noise to help the detectors learn an approximate boundary between good and victim samples. As mentioned we discuss the artificial noise we consider in Section 4.6.1. We validate the effectiveness of mixed-type artificial noise in Section 4.6.4.

4.6 Experiments

We evaluate how effective Picket and a diverse array of competing methods are on detecting different types of corruption in ML pipelines during the training and inference phases. We also provide several micro-benchmarks over different design choices in Picket. Finally, we report the runtime and discuss the scalability.

4.6.1 Experimental Setup

Datasets We consider six datasets with different mixtures of numerical, categorical, and text-based attributes. These datasets are obtained from the UCI repository (Dua and Graff, 2017) and the CleanML benchmark (Li et al., 2019). All datasets focus on binary classification tasks. The characteristics of these datasets are summarized in Table 4.1. A detailed description of the datasets is as follows.

- **Wine:** The dataset consists of statistics about different types of wine based on physicochemical tests. The task is to predict if the quality of a type is beyond average or not. The features are purely numerical.
- **Adult:** The dataset contains a set of US Census records of adults. The task is to predict if a person makes over \$50,000 per year. The features are a mixture of categorical and numerical attributes.
- **Marketing:** The dataset comes from a survey on household income consisting of several demographic features. The task is to predict whether the annual gross income of a household is less than \$25,000. The features are purely categorical.
- **Restaurant:** The dataset contains information of restaurants from Yelp. The task is to predict if the price range of a restaurant is "one dollar sign" on Yelp. The features are a mixture of categorical values and textual descriptions.
- **Titanic:** The dataset contains personal and ticket information of passengers. The task is to predict if a passenger survives or not. The features are a mixture of numerical, categorical and textual attributes.
- **HTRU2:** The dataset contains statistics about a set of pulsar candidates collected in a universe survey. The task is to predict if a candidate is a real pulsar or not. The features are purely numeric.

The last dataset, i.e., HTRU2, is purely numerical and we use it in the context of adversarial noise. The datasets above are the ones we use for most of our experiments. In addition, we use Food labeled by (Heidari et al., 2019) for real noise and Alarm (Herskovits, 1992) for the study of scalability. We consider downstream ML pipelines over these datasets that use 80% of each dataset as the training set, and the rest as test data. To reduce the effect of class imbalance, we undersample the unbalanced datasets where over 70% of the samples belong to one class. The numerical attributes are normalized to zero mean and unit variance before noise injection.

Table 4.1: Properties of the datasets in our experiments.

Dataset	Size	Numerical Attributes	Categorical Attributes	Textual Attributes
Wine	4898	11	0	0
Adult	32561	5	9	0
Marketing	8993	0	13	0
Restaurant	12007	0	3	7
Titanic	891	2	5	3
HTRU2	17898	8	0	0

Noise Models In our experiments, we consider different types of noise: 1) random, 2) systematic, 3) adversarial noise, and 4) common errors in real-world datasets.

Random and systematic noise are model agnostic and only take into account the dataset. For random and systematic noise, we corrupt β fraction of the cells in the noisy samples. We now provide a detailed description of the random and systematic noise generation process we consider.

Random Noise: For a categorical or textual attribute, the value of a corrupted cell is flipped to another value in the domain of that attribute. For a numerical attribute, we add Gaussian noise to the value of a corrupted cell, with zero mean and standard deviation of σ_1, where σ_1 is a constant.

Systematic Noise: For categorical and textual data, we randomly generate a predefined function ϕ which maps the value $x^{*(i)}$ of the cell to be corrupted to another value $x^{(i)}$ in the same domain. The mapping function depends on both the original value in that attribute and that in another pre-specified attribute, i.e., $x^{(i)} = \phi(x^{*(i)}, x^{*(j)})$ where $j \neq i$. For a numerical attribute, we add a fixed amount of noise with magnitude σ_2 to the value of a corrupted cell, where σ_2 is a constant.

We consider three settings with varying fractions of corrupted cells in the noisy examples (aw well as the magnitude of error in the case of numerical values) for random and systematic noise, which we refer to as High ($\beta = 0.5$, $\sigma_1 = \sigma_2 = 5$), Medium ($\beta = 0.3$, $\sigma_1 = \sigma_2 = 3$) and Low ($\beta = 0.2$, $\sigma_1 = \sigma_2 = 1$).

For adversarial attacks, we use methods that take into account specific ML models. Specifically, we use data poisoning techniques during training and evasion attack methods at inference. For the part of our evaluation that focuses on training time, we generate poisoned samples using the back-gradient method (Muñoz-González et al., 2017). Since this type of poisoning is specific to different downstream models we consider different dataset-model combinations in our evaluation. For the part of our evaluation that focuses on inference time, we use the projected gradient descent (PGD) attack (Madry et al., 2018a), a popular and effective white-box evasion attack method, to generate adversarial test samples. We use the implementation of PGD attack from the Adversarial Robustness Toolbox (Nicolae et al., 2018). The corruption injected by the PGD attack is bounded by an infinity norm of 0.2. The step size is 0.1 and the number of iterations is 100.

For common errors in real-world datasets, we consider missing values that cannot be detected during pre-processing (e.g. 99999 instead of NaN), multiplicative scaling of attributes (e.g. due to accidental changes of units), and typos in textual or categorical attributes. We synthesize this kind of noise as follows:

1. If the corrupted cell is numerical, with a probability 1/3 it will be 10 times larger, and with the same probability, it will be 10 times smaller. Otherwise, the cell will contain a missing value.

2. If the corrupted cell is categorical or textual, with probability 1/2 one of the characters will be replaced by a random character. Otherwise,

the cell will contain a missing value.

For this kind of noise, we set the fraction of corrupted cells in the noisy samples as $\beta = 0.3$. We also include Food, a dataset that contains real-world errors with manually labeled ground truth (Heidari et al., 2019). It has 3 numerical, 6 categorical and 5 textual attributes. Out of its 3000 samples, 30.3% are corrupted.

As discussed in Section 4.5, we use artificially generated noise to create the training data for learning the victim-sample detectors. We now describe the type of noise we consider. Recall that we consider access to the set of clean sample C and we augment this set with artificially corrupted data. We emphasize that the noise is always different than the noise considered in the training data. Since we assume that the type of noise in the test set is unknown in advance, the artificial noise contains a mixture of different levels of random noise (($\beta = 0.4, \sigma_1 = 4$), ($\beta = 0.25, \sigma_1 = 2$), ($\beta = 0.15, \sigma_1 = 1.5$)). We additionally augment C with samples corrupted by random noise ($\beta = 1, \sigma_1 = 0.25$) and adversarial samples generated by Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2015)(noise bounded by an infinite norm of 0.1) to defend against adversarial noise. This corruption is different from the PGD attack considered during inference to ensure that we evaluate against a different noise distribution during online inference.

Downstream ML Models We consider the following downstream models: 1) A Logistic regression (LR) model with regularization parameter 1.0; 2) A Support Vector Machine (SVM) with a linear kernel and regularization parameter 1.0; 3) A fully-connected neural network (NN) with 2 hidden layers of size 100. We use a small model with 1 hidden layer of size 10 when we perform poisoning attacks due to the runtime complexity of the attack algorithm. The downstream models we choose cover different optimization objectives (logistic/hinge loss and convex/non-convex optimization objectives) and exhibit different robustness. Numerical attributes are encoded as their raw values for downstream models. Categorical and textual attributes are encoded in the same way as in Picket.

Training-Time Baselines We compare against three unsupervised outlier detection methods as follows: 1) Isolation Forest (IF) (Liu et al., 2008), an approach similar to Random Forests but targeting outlier detection, 2) One-Class SVM (OCSVM) (Chen et al., 2001) with a radial basis function kernel, and 3) Robust Variational Autoencoders (RVAE) (Eduardo et al., 2020), a state-of-the-art generative model used for outlier detection on mixed-type tabular data. For IF, we use 100 base estimators in the ensemble. For RVAE, we use the default hyperparameter recommended by Eduardo et al. (Eduardo et al., 2020), which has 972,537 parameters. Note that the capacity of the RVAE model used in our experiments is larger than PicketNet, which has 382,722 parameters.

Inference-Time Baselines We compare against the following baselines: 1) victim-sample detectors, 2) naïve confidence-based methods, and 3) adversarial data detection methods.

Methods based on per-class victim sample detectors follow the same strategy as Picket but use different features. We consider: 1) *Raw Feature (RF)*, the binary classifiers only use the raw features of the data; 2) *RVAE*, the binary classifiers use only the cell-level probability of being outliers provided by RVAE as features; 3) *RVAE+*, the classifiers use a combination of the features from the two methods above.

We also consider the next naïve methods: 1) *Calibrated Confidence Score (CCS)*, which assumes that the predictions of the downstream model have lower confidence for victim samples than clean samples. We calibrate the confidence scores of the downstream models using temperature scaling (Guo et al., 2017). 2) *k-Nearest Neighbors (KNN)*, which assumes that a victim sample has a different prediction from its neighbors. We use different distances for different types of attributes. For numerical attributes, the distance is $d/0.05$ if $d \leq 0.05$, where d is the difference between two normalized values; the distance is 1 if $d > 0.05$. For categorical attributes, we use the Hamming distance and for text attributes the cosine distance. We set k to 10.

We consider two methods of adversarial sample detection: *The Odds are Odd (TOAO)* (Roth et al., 2019), which detects adversarial samples based on the change in the distribution of the prediction logit values after the injection of random noise. It adds Gaussian, Bernoulli, and Uniform noise of different magnitude and takes the majority vote of all noise sources. 2) *Model with Outlier Class (MWOC)* (Grosse et al., 2017), which assumes that the feature distribution of adversarial samples is different from that of benign samples and adds a new outlier class to the downstream model to characterize the distribution of adversarial samples.

For a fair evaluation, we also reveal the augmented version of C used to learn the victim-sample detectors in Picket to competing methods so that they fine-tune their models to noise (RF, RVAE, RVAE+, MWOC, Picket), or use it to find a good threshold (CCS, KNN, TOAO).

Metrics For training-time outlier detection, we report the area under the receiver operating characteristic curve (AUROC). We use AUROC since it is an aggregate measure of performance across all possible threshold settings. We also consider the test accuracy of downstream models. For victim sample detection, we report the F_1 scores of the classification between correctly classified samples and victim samples.

Evaluation Protocol All experiments are repeated five times with different random seeds that control train-test split and noise injection; the mean is reported. We also perform one-sided paired t-tests when we compare the examined methods. A method is considered significantly better than another one if the p value is less than 0.05.

Hyper-parameters of PicketNet PicketNet is not sensitive to hyper-parameters in most cases. The default hyper-parameters we use in the experiments is shown in Table 4.2. For purely numerical datasets, we reduce the dimension of $I_i^{(l)}$ and $P_i^{(l)}$ to 8, and for HTRU2, we reduce the number of self-attention layers to 1. In the other datasets, we always use

Table 4.2: Default hyper-parameters for PICKETNET.

Hyper-Parameter	Value
Number of Self-Attention Layers	6
Number of Attention Heads	2
Dimension of $I_i^{(l)}$ and $P_i^{(l)}$	64
Number of Hidden Layers in Each Feedforward Network	1
Dimension of the Hidden Layers in Feedforward Networks	64
Dropout	0.1
Size of the Negative Sample Set Z_i	4
Warm-up Epochs E_1 for Loss-Based Filtering	50
Loss Recording Epochs E_2	20

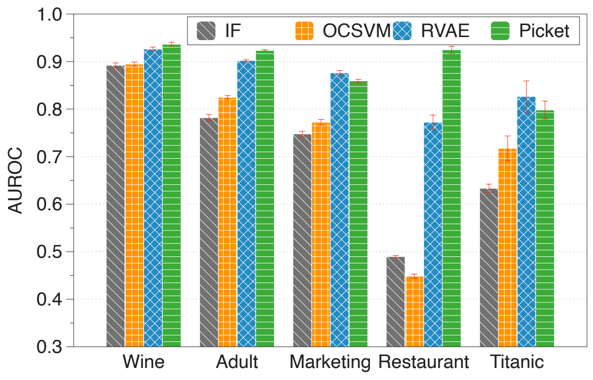


Figure 4.5: AUROC of outlier detection for random noise. The error bars represent the standard errors. Picket is significantly better (with p value less than 0.05) than the others on Wine, Adult, Marketing and Restaurant.

the default hyperparameters. We use the Adam optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$ for training. The learning rate $lr = 0.5d^{-0.5}\min(s^{-0.5}, 300^{-1.5}s)$, where d is the dimension of $I_i^{(l)}$ and $P_i^{(l)}$, s is the index of the training step. lr increases in the first few steps and then decreases. Typically, PICKETNET takes 100 to 500 epochs to converge, depending on the datasets.

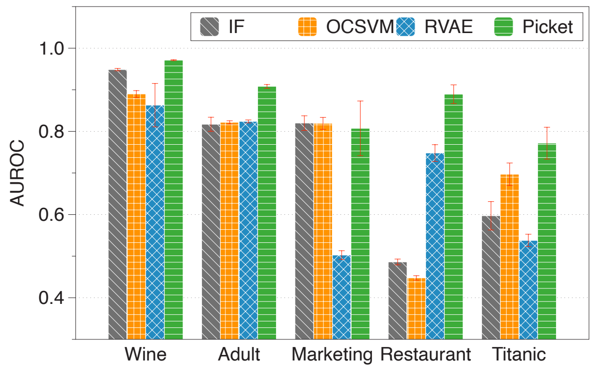


Figure 4.6: AUROC of outlier detection for systematic noise. The error bars represent the standard errors. Picket is significantly better (with p value less than 0.05) than the others on Wine, Adult, Restaurant and Titanic.

4.6.2 Training-Time Evaluation

We evaluate the performance of different methods on detecting erroneous points in the training data. We then evaluate how these methods affect the performance of downstream models.

Detecting Corrupted Training Examples Figure 4.5, 4.6, and 4.7 show the AUROC obtained by the methods for different types of noise, when 20% of the samples are corrupted. The results for random and systematic noise correspond to Medium level noise. Results for Low and High levels are reported in Section 4.7.4. For Figure 4.7, note that the poisoned samples are model-specific and hence we report the dataset model combination on the x-axis. Due to data poisoning being limited to numerical data, we only evaluate on Wine and HTRU. As shown, *Picket is the only approach that consistently achieves an AUROC of close to or more than 0.8 for all datasets and for all noise settings.* Other methods achieve comparable performance in some settings but they are not consistent across diverse settings. IF and OCSVM perform poorly on datasets with textual attributes (Restaurant and Titanic) due to their limited capacity to handle text-based attributes. RVAE works quite well under random noise, but its performance drops a lot

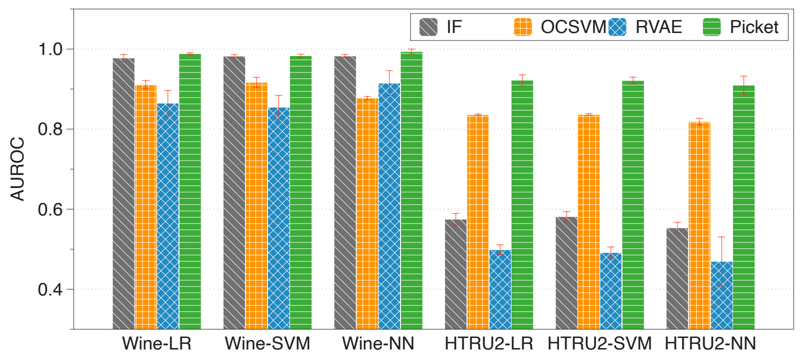


Figure 4.7: AUROC of outlier detection for poisoned samples. The error bars represent the standard errors. Picket is significantly better (with p value less than 0.05) than the others on all the combinations except Wine-SVM.

when it comes to systematic noise, which shows that it is not robust against noise that introduces bias. In the presence of poisoned data, we find that IF performs well on Wine but poorly on HTRU2, but OCSVM shows the opposite. A possible reason is that the two datasets exhibit different types of correlation between attributes, and the two methods are good at capturing only one of them. RVAE shows poor performance for both datasets.

For common errors in the real world, the results are shown in Figure 4.8. We add synthetic errors of this type to Titanic and Restaurant, where 20% are corrupted. We choose these two because they contain textual attributes for typos. We also report the results on Food with real-world noise. We can see that on Restaurant and Titanic, PICKET outperforms the others by more than 6 points. On Food, all the methods perform poorly. This is because the noise level in Food is very low, and therefore hard to detect. In fact, the real noise contained in Food does not have a significant effect on the downstream models (as is shown in Table 4.22).

We also study how the fraction of corrupted samples affects the performance of detection (see Section 4.7.3). PICKET keeps a relatively consistent performance when the fraction of corrupted samples varies.

Effect on Downstream Models We also study the effect of different

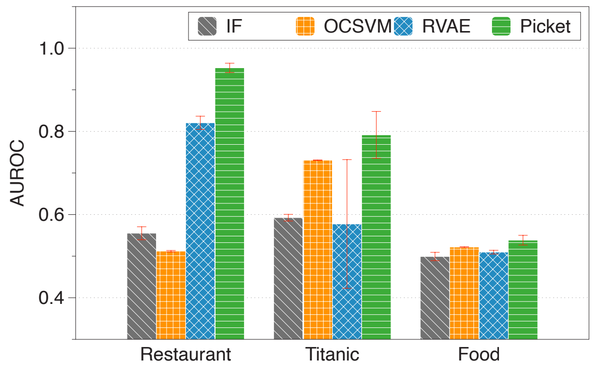


Figure 4.8: AUROC of outlier detection for common errors in the real world. The error bars represent the standard errors. Picket is significantly better (with p value less than 0.05) than the others on all the datasets.

filtering methods on the downstream models. For each method, we filter 20% of the samples with highest outlier scores, and train different downstream models on the resulting training set. For each dataset, the test set is fixed and contains only clean data. As reference points, we also include the test accuracy when 1) the training data is clean without corruption (CL), and 2) the training data is corrupted but no filtering (NF) is performed. Note that in the CL and NF cases, the sample size is different from the rest since there is no filtering in these two. As a side effect of filtering, the decrease in sample size will also affect the performance of the downstream model. We want to include such an effect in our comparison, so we use CL and NF with no sample filtered out as baselines.

First, we consider the case of data poisoning since this type of corruption has the most significant effect on the downstream models. We measure the test accuracy of the downstream models when poisoned data are injected into the training stage. The results are shown in Table 4.3. If we compare CL with NF we see an average drop of six accuracy points if corruptions are ignored and no filtering is applied. We find that all methods reduce the negative impact of the poisoned data and bring up the test accuracy. Nevertheless that *PICKET outperforms all competing baselines and yields*

Table 4.3: Test accuracy of downstream models under adversarial poisoning attacks and different filtering methods. The numbers are made bold when the corresponding method is significantly better (with p value less than 0.05) than all the others.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Wine	LR	0.7261	0.6976	0.7051	**0.7312**	0.7349	0.6745
	SVM	0.7286	0.6933	0.7082	0.7310	0.7386	0.6727
	NN	0.7210	0.6894	0.7035	**0.7320**	0.7365	0.6722
HTRU2	LR	0.8884	0.9015	0.8811	**0.9067**	0.9396	0.8799
	SVM	0.8884	0.8979	0.8887	**0.9232**	0.9424	0.8832
	NN	0.8671	0.8707	0.8643	**0.9000**	0.9280	0.8646

*DM = Downstream Model.

test time accuracy improvements of more than three points in some cases. We see that PICKET is able to recover most of the accuracy loss for all models in the Wine dataset and comes very close to CL for HRTU2. All other methods exhibit smaller accuracy improvements and do not exhibit consistent behavior across datasets.

We also consider the cases of random and systematic noise, as well as common errors in the real world. These types of noise do not directly attack the downstream model. Moreover, most ML models are somewhat robust to these types of noise. As a result, we expect to see a small gap in the test accuracy between CL and NF, and all methods to perform comparably. We report the results in these setups in Section 4.7.5 for completeness.

4.6.3 Inference-Time Evaluation

We evaluate the different methods on victim sample detection under different types of noise. The F_1 scores under random (Medium level), systematic (Medium level), adversarial noise and common errors in the real world are reported in Table 4.4, 4.5, 4.6 and 4.7. Food with real-world noise is not reported since we cannot find enough victim samples from it. We report results for High and Low noise in Section 4.7.6.

From the tables, we can see that PICKET has the best performance in most

Table 4.4: F_1 scores of victim sample detection at inference time under random noise (Medium level). The numbers are made bold when the corresponding method is significantly better (with p value less than 0.05) than all the others.

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	Picket
Wine	LR	0.7690	0.7786	0.8172	0.6667	0.6686	0.6813	0.7150	0.8094
	SVM	0.7812	0.7859	0.8254	0.6667	0.6750	0.6858	0.7622	0.8223
	NN	0.7125	0.7470	**0.7833**	0.5896	0.6669	0.5107	0.6988	0.7631
Adult	LR	0.8352	0.7403	0.8489	0.6692	0.7866	0.2224	0.6725	**0.8602**
	SVM	0.8434	0.7416	0.8553	0.6688	0.8060	0.4696	0.6215	**0.8658**
	NN	0.8131	0.7127	0.8315	0.5117	0.6891	0.3216	0.7132	**0.8411**
Restaurant	LR	0.7726	–[#]	–	0.7403	0.6456	0.6457	0.7459	**0.8266**
	SVM	0.6854	–	–	0.6796	0.6628	0.6596	0.5580	**0.7618**
	NN	0.7605	–	–	0.6994	0.6609	0.6110	0.7025	**0.8203**
Marketing	LR	0.8366	0.6623	0.8403	0.7567	0.7815	0.6666	0.7996	**0.8549**
	SVM	0.8461	0.6689	0.8501	0.7527	0.7886	0.5133	0.8109	**0.8607**
	NN	0.7931	0.6650	0.8029	0.6588	0.7050	0.6648	0.7265	0.8162
Titanic	LR	0.8257	–	–	0.6990	0.6562	0.1409	0.7736	**0.8424**
	SVM	0.8482	–	–	0.6658	0.6436	0.4652	0.7932	0.8528
	NN	0.8393	–	–	0.6631	0.6387	0.2575	0.7566	0.8483

*DM = Downstream Model. [#]RVAE is not applicable to text attributes.

cases. By comparing RF and our method, we show that *the reconstruction loss features provided by PicketNet are good signals to help identify victim samples.* Such signals are better than those provided by RVAE since our method outperforms RVAE+ most of the time. TOAO performs consistently poorly since the assumption it relies on does not hold for the downstream models and datasets we consider. It works for image classification with complex convolutional neural networks under adversarial settings since adding random noise to images could eliminate the effect of adversarial noise. However, for tabular datasets and simple models, especially when the noise is not adversarial, adding random noise does not make a big difference. Another method from the adversarial learning literature (MWOC) works well in some cases even if the noise is not adversarial.

Table 4.5: F_1 scores of victim sample detection at inference time under systematic noise (Medium level). The numbers are made bold when the corresponding method is significantly better (with p value less than 0.05) than all the others.

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	Picket
Wine	LR	0.6883	0.4987	0.6619	0.6667	0.6499	0.3152	**0.7937**	0.7046
	SVM	0.6785	0.5056	0.6630	0.6667	0.6325	0.3399	**0.7957**	0.6973
	NN	0.6760	0.6134	0.5689	0.6865	0.6659	0.3765	**0.7190**	0.6034
Adult	LR	0.8281	0.6960	0.8342	0.6695	0.7488	0.1864	0.7430	**0.8501**
	SVM	0.8414	0.6729	0.8428	0.6694	0.7900	0.3617	0.6646	**0.8643**
	NN	0.8108	0.6534	0.8245	0.5439	0.6808	0.2195	0.7850	**0.8336**
Restaurant	LR	0.7773	–#	–	0.7419	0.6524	0.6496	0.7487	**0.8255**
	SVM	0.7275	–	–	0.7093	0.6475	0.6356	0.6125	**0.7845**
	NN	0.7628	–	–	0.7010	0.6579	0.6051	0.7003	**0.8126**
Marketing	LR	0.8358	0.6504	0.8403	0.7623	0.7770	0.6090	0.8068	**0.8514**
	SVM	0.8501	0.6575	0.8552	0.7716	0.7817	0.6185	0.8208	**0.8638**
	NN	0.8036	0.6355	0.8098	0.6649	0.7074	0.6635	0.7035	0.8118
Titanic	LR	0.8376	–	–	0.7349	0.6493	0.4076	0.7901	0.8438
	SVM	0.8224	–	–	0.6674	0.6387	0.5592	0.7593	0.8412
	NN	0.8112	–	–	0.6660	0.6333	0.3139	0.7462	0.8159

*DM = Downstream Model. #RVAE is not applicable to text attributes.

Table 4.6: F_1 scores of victim sample detection at inference time under adversarial noise. The numbers are made bold when the corresponding method is significantly better (with p value less than 0.05) than all the others.

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	Picket
Wine	LR	0.7899	0.6758	0.7905	0.8233	0.6660	0.5030	0.8287	0.8197
	SVM	0.7951	0.6791	0.8004	0.8119	0.6660	0.5743	0.8324	0.8291
	NN	0.7400	0.6922	0.7347	0.6815	0.6663	0.6620	0.3980	0.7442
HTRU2	LR	0.8727	0.0160	0.8699	0.6667	0.6654	0.5123	0.8389	0.8757
	SVM	0.9409	0.3436	0.9399	0.6667	0.6623	0.6456	0.2211	**0.9438**
	NN	0.9103	0.3007	0.9164	0.7258	0.6656	0.2873	0.7726	0.9201

*DM = Downstream Model.

Table 4.7: F_1 scores of victim sample detection at inference time under common errors in the real world. The numbers are made bold when the corresponding method is significantly better (with p value less than 0.05) than all the others.

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	Picket
Restaurant	LR	0.7335	–$^{\#}$	–	0.7420	0.6527	0.5003	0.7330	0.7445
	SVM	0.6948	–	–	0.7168	0.6415	0.6104	0.6189	0.6928
	NN	0.7716	–	–	0.6818	0.6633	0.5470	0.6762	0.7713
Titanic	LR	0.5633	–	–	0.3350	0.6792	0.5934	0.4740	**0.8905**
	SVM	0.6304	–	–	0.4412	0.6798	0.4374	0.5706	**0.8651**
	NN	0.6100	–	–	0.4140	0.6816	0.6855	0.8093	0.8205

*DM = Downstream Model. #RVAE is not applicable to text attributes.

4.6.4 Micro-Benchmarks

We perform a series of micro-benchmarks to evaluate different design decisions related to Picket.

Effectiveness of the Two-Stream Self-Attention We perform an ablation study to validate the effectiveness of the two-stream self-attention. We evaluate the performance of outlier detection with only one stream and with both. The results are depicted in Figure 4.9. In the case of one stream, we simply let the output of self-attention layer l be either $H_i^{(l)}$ for the value stream, or $G_i^{(l)}$ for the schema stream instead of $H_i^{(l)} + G_i^{(l)}$, where i is the index of the attribute. For fair comparison, we expand the dimension of all the vectors involved in the computation of multi-head self-attention functions and feed-forward networks by a factor of $\sqrt{2}$ in the one-stream cases, so that the network capacity (number of parameters) remains the same after the pruning of one stream. We use three setups: Wine with poisoning attack on NN, Adult with systematic noise (Medium level), and Marketing with random noise (Medium level).

From Figure 4.9, we see that for Adult and Marketing, PicketNet with two streams outperforms both one-stream options. For Wine, the value stream itself works fine, but a combination of the two streams does not impair the performance of the model. Neither of the two one-stream options

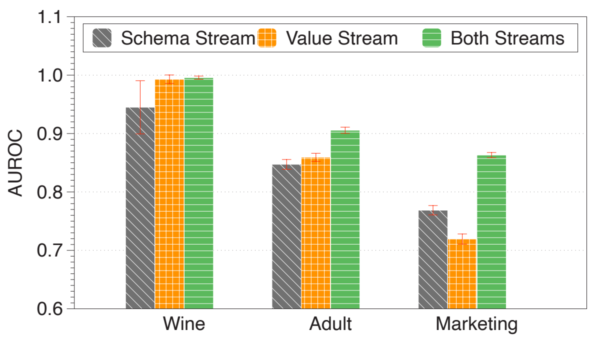


Figure 4.9: Outlier detection under different stream settings. The error bars represent the standard errors.

demonstrates obvious superiority over the other one, since there are cases that the value stream performs better than the schema stream, and cases that the opposite happens.

Effectiveness of Early Filtering We validate the effectiveness of early filtering by comparing the performance of outlier detection at the early stage of PICKETNET's training to that after convergence. The results are shown in Figure 4.10. We use the setup from the previous micro-benchmark. Figure 4.10 shows that filtering at early stages consistently outperforms filtering after convergence. The reason is that in the early stage of training, the model is less likely to overfit to the input, and therefore the reconstruction loss of the outliers differs from that of the clean samples more.

Histogram-Based Threshold Selection We validate the effectiveness of the histogram-based threshold selection strategy mentioned in Section 4.4.2. To better illustrate how it affects the downstream accuracy, we use Wine and HTRU2 with poisoning attacks (20% of the samples are poisoned) where corruption has a significant effect on the downstream models. For each dataset and downstream model combination, we plot the histogram of the Picket reconstruction loss in Figure 4.11, and select the thresholds δ_{low} and δ_{high} accordingly so that the abnormal peaks and low-density tails are filtered out. We report the downstream accuracy after filtering with

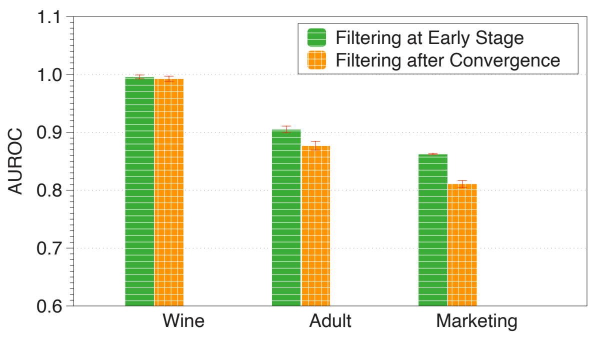
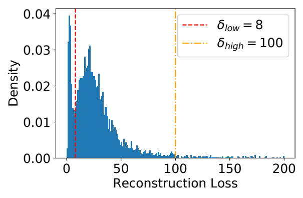
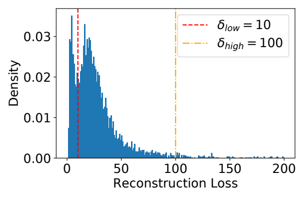


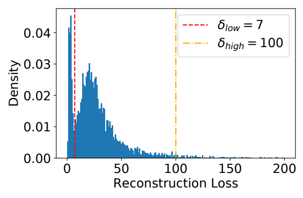
Figure 4.10: Early vs. after-convergence filtering. The error bars represent the standard errors.



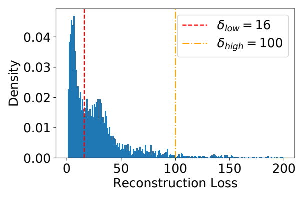
(a) Wine-LR



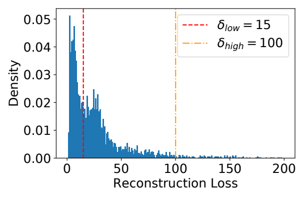
(b) Wine-SVM



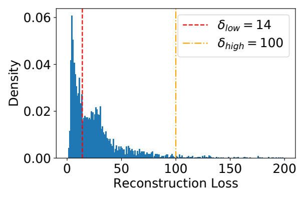
(c) Wine-NN



(d) HTRU2-LR



(e) HTRU2-SVM



(f) HTRU2-NN

Figure 4.11: Histograms of the reconstruction loss under different dataset-model combinations and the thresholds δ_{low}, δ_{high}.

this strategy (Picket-Hist) in Table 4.8. Same as Section 4.6.2, we also report the downstream accuracy under CL and NF as reference points. The results show that Picket-Hist gets very close to CL where the data is clean, and much better than NF where no filtering is applied, which verifies the effectiveness of this threshold-selection strategy.

Effectiveness of Per-Class Victim Sample Detectors We compare the

Table 4.8: Test accuracy of downstream models after filtering based on the histogram of reconstruction loss (Picket-Hist).

Dataset	Downstream Model	Picket-Hist	CL	NF
Wine	LR	0.7500	0.7551	0.6846
	SVM	0.7459	0.7530	0.6836
	NN	0.7133	0.7204	0.6561
HTRU2	LR	0.9344	0.9435	0.8810
	SVM	0.9375	0.9435	0.8856
	NN	0.9207	0.9207	0.8720

performance of our per-class detectors against a unified detector and a score-based detector. The unified detector uses one single logistic regression model over the same features to distinguish between good and victim samples regardless of the downstream predictions. The score-based detector follows the logic of the training time outlier detector, i.e., it aggregates the reconstruction losses attribute-wise, and considers samples with high loss as victims. We perform the comparison on three datasets with all of the three downstream models: Wine with adversarial noise, Adult with systematic noise (Medium level) and Marketing with random noise (Medium level).

The result is shown in Table 4.9. Per-class detectors outperform the other two, which validates the effectiveness of having one detector per class. The unified detector performs poorly because the victim samples in one class differ from those in the other statistically, in which case one class may suffer from corruption in one group of attributes, while the other class may suffer from that in another group of attributes. The score-based detector does not work well since it only has access to the noise level but does not consider the connection between corruptions and the downstream prediction.

Effectiveness of Mixed Artificial Noise We validate the effectiveness of our artificial noise setting (Mixed) by comparing it to the setting where the artificial noise is generated in the same way as the test time noise (Exact). The results are shown in Table 4.10. We use the same datasets and test time noise as the previous micro-benchmark. We find that with mixed artificial noise, the performance of Picket is comparable to the setting where the exact

Table 4.9: A comparison between the per-class detectors, the unified detector, and the score-based detector on inference time victim sample detection.

Dataset	Downstream Model	Per-Class Detectors	Unified Detector	Score-based Detector
Wine	LR	0.8188	0.7023	0.6885
	SVM	0.8287	0.7152	0.7261
	NN	0.7444	0.4027	0.6594
Adult	LR	0.8489	0.6710	0.7197
	SVM	0.8634	0.6983	0.7297
	NN	0.8336	0.6785	0.7225
Marketing	LR	0.8553	0.7740	0.7343
	SVM	0.8618	0.7774	0.7361
	NN	0.8152	0.7370	0.7174

noise distribution is known under random (see Marketing) and systematic noise (see Adult). Under adversarial noise (see Wine), Exact is better than Mixed but the gap is not excessively large.

Table 4.10: F1 scores of Picket on victim sample detection under different artificial noise settings.

Dataset	Downstream Model	Mixed	Exact
Wine	LR	0.8197	0.8646
	SVM	0.8291	0.8812
	NN	0.7442	0.7790
Adult	LR	0.8501	0.8372
	SVM	0.8643	0.8562
	NN	0.8336	0.8157
Marketing	LR	0.8549	0.8544
	SVM	0.8607	0.8592
	NN	0.8162	0.8120

4.6.5 Fairness of Outlier Detection

We compute the equality of opportunity between majority and minority groups to check the fairness of outlier detection. Specifically, the opportunity $\gamma_{\mathtt{G}}$ for each group $\mathtt{G}$ is defined as the fraction of clean examples in that group that are kept after filtering:

$$\gamma_{\mathtt{G}} = N_{\mathtt{G}}^{\text{kept}} / N_{\mathtt{G}}^{\text{clean}}$$

where $N_{\mathtt{G}}^{\text{clean}}$ is the number of clean examples in group $\mathtt{G}$, and $N_{\mathtt{G}}^{\text{kept}}$ is the number of clean examples in $\mathtt{G}$ that are not filtered out. We report the difference of opportunity $\Delta\gamma = \gamma_{\mathtt{G}_m} - \gamma_{\mathtt{G}_M}$, where $\mathtt{G}_M$ is the majority group and $\mathtt{G}_m$ is the minority. $\Delta\gamma$ closer to 0 indicates better fairness.

We choose two demographic datasets, Adult and Marketing, to verify the fairness of the outlier detection methods. For each dataset, we pick one sensitive attribute at a time, and divide its value domain into majority and minority groups as follows:

1. Sort the values by their frequency in descending order.
2. Add values in order to the majority group until it covers more than 80% of the examples.
3. Add the rest of the values into the minority group.

We inject random and systematic noise of medium magnitude to 20% of the examples, filter out 20%, and report the difference of opportunity for each dataset-attribute combination in Table 4.11. We can see that for most of the sensitive attributes, the difference of opportunity is less than 0.05 if the data are filtered by Picket. However, for certain attributes (e.g. Marketing-Marital and Marketing-Language), the difference is quite large, which shows potential risk of unfairness. The other models also show bias towards the majority group for certain attributes. We defer the improvement of fairness as a future direction to explore.

4.6.6 Runtime and Scalability

We report the training time of PicketNet for each dataset in Table 4.13. The device we use is a single NVIDIA Tesla V100-PCIE GPU with 32GB

Table 4.11: Difference of opportunity when 20% of the examples are filtered out.

Noise Type	Dataset-Attribute	IF	OCSVM	RVAE	Picket
Random	Marketing-Marital	-0.0469	-0.0821	**-0.0196**	-0.1400
	Marketing-Age	0.0720	-0.0071	**0.0019**	0.0216
	Marketing-Education	-0.0436	-0.0521	-0.0192	**0.0174**
	Marketing-Live	-0.1131	-0.1242	-0.0488	**-0.0242**
	Marketing-Dual	-0.0226	-0.0737	-0.0357	**-0.0089**
	Marketing-Hometype	-0.0865	-0.1581	-0.0634	**-0.0458**
	Marketing-Ethnic	-0.1762	-0.2258	-0.0760	**-0.0610**
	Marketing-Language	-0.5625	-0.4739	**-0.0753**	-0.3739
	Adult-Workclass	-0.1290	-0.0259	-0.0277	**-0.0042**
	Adult-Marital-status	-0.3111	-0.0676	**-0.0013**	-0.0706
	Adult-Relationship	-0.2545	**0.0027**	0.0081	-0.0188
	Adult-Race	-0.4452	-0.0326	**-0.0259**	-0.0515
Systematic	Marketing-Marital	-0.0541	-0.1178	**-0.0418**	-0.2031
	Marketing-Age	0.0902	**-0.0051**	0.0097	0.0270
	Marketing-Education	-0.0366	-0.0509	-0.0164	**0.0142**
	Marketing-Live	-0.0781	-0.1333	-0.0640	**-0.0090**
	Marketing-Dual	-0.0275	-0.0757	**-0.0224**	-0.0244
	Marketing-Hometype	-0.0995	-0.1892	-0.1182	**-0.0528**
	Marketing-Ethnic	-0.1919	-0.2465	-0.1388	**-0.0777**
	Marketing-Language	-0.5981	-0.5397	**-0.1555**	-0.4791
	Adult-Workclass	-0.1819	-0.0079	-0.0287	**-0.0012**
	Adult-Marital-status	-0.3158	-0.0690	**0.0026**	-0.2519
	Adult-Relationship	-0.2444	-0.0148	**0.0067**	-0.0635
	Adult-Race	-0.4124	-0.0560	**-0.0088**	-0.0719

memory. Note that the current runtime has not been fully optimized.

We also study the attribute-wise scalibilty of PICKETNET using synthetic datasets. The datasets have a different number of attributes ranging from 2 to 20 with a increase step of one, while the other settings are the same (the dimension of $I_i^{(l)}$ and $P_i^{(l)}$ is fixed to 8). We report the training time of 100 epochs in Figure 4.12. The growth of the runtime is roughly quadratic as the number of attributes increases. This is expected since the dependencies between one attribute and all the others yield quadratic complexity. When the number of attributes is excessively large, we can apply simple methods like computing the correlations between attributes to split the attributes into groups, where only the attributes within the same group exhibit correlations. Then, we can apply PICKETNET to learn the structure for each of the groups. We evaluate the effectiveness of this strategy on the Alarm dataset (Herskovits, 1992) which contains 36 attributes and 1000 records. The functional dependencies in Alarm is known. We split the attributes into three groups based on the functional dependencies. Each group contains 12 of them. We run Picket outlier detection on the three groups independently, and then aggregate the reconstruction loss across groups. We inject random and systematic noise of medium magnitude to 20% of the records, and report the AUROC of outlier detection in Table 4.12. The results show that PICKET provides high-quality outlier detection under the aforementioned strategy.

Table 4.12: AUROC scores of outlier detection on the Alarm dataset. The attributes are split into three groups for Picket.

Noise Type	IF	OCSVM	RVAE	PICKET
Random	0.8848	0.8835	0.9357	0.9579
Systematic	0.7410	0.7283	0.7957	0.7967

We report the inference time overhead (runtime of PICKETNET loss computing and victim sample detectors) as long as the runtime of downstream prediction of each dataset in Table 4.14, when the data come in batches of 100. We can see that the overhead of PicketNet loss computing dominates the runtime, but it is still no more than a few seconds. As the downstream model becomes more complex, the relative overhead introduced by PICKET would be reduced.

Table 4.13: Training time of PICKETNET for each dataset.

Dataset	Wine	Adult	Restaurant	Marketing	Titanic	HTRU2
Training Time (sec)	1953	8256	3794	4581	1693	189

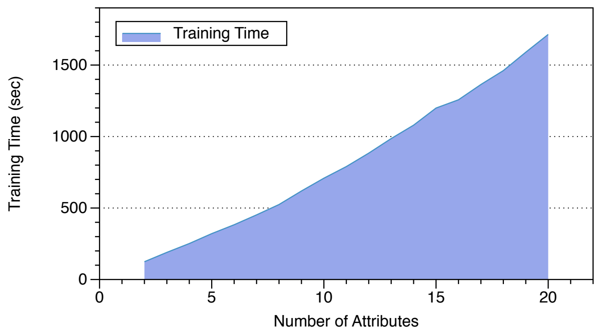


Figure 4.12: Attribute-wise scalibility of PicketNet

Table 4.14: Inference overhead of Picket and runtime of downstream prediction.

Dataset	Wine	Adult	Restaurant	Marketing	Titanic	HTRU2
PicketNet Loss Computing (sec)	0.1512	0.4303	0.3003	0.4048	0.2892	0.0316
Victim Sample Detectors (sec)	0.0006	0.0009	0.0012	0.0009	0.0008	0.0004
Downstream Prediction (LR) (sec)	0.0003	0.0010	0.0016	0.0011	0.0006	0.0003
Downstream Prediction (SVM) (sec)	0.0003	0.0021	0.0012	0.0012	0.0006	0.0003
Downstream Prediction (NN) (sec)	0.0004	0.0021	0.0030	0.0014	0.0005	0.0004

4.7 Additional Experimental Results

4.7.1 Outlier Detection on Synthetic Data

We evaluate the performance of outlier detection on synthetic datasets to understand the effects of several aspects of the data and noise, including the strength of structure, data dimension, noise level, and magnitude of extreme outliers. Here the term structure means dependencies between attributes.

We generate synthetic datasets as follows. Each synthetic data point $x = [x^{(1)}, x^{(2)}, \ldots, x^{(T)}]^T$ is generated by $x = \mathbf{A}z$, where $z \in \mathbb{R}^R$ and $\mathbf{A} \in \mathbb{R}^{T \times r}$. Each entry of z is sampled from the standard Gaussian distribution, and each entry of $\mathbf{A}$ is sampled uniformly from -1 to 1. Unless otherwise specified, we inject random noise with $\beta = 0.2$ and $\sigma_1 = 1$ to 20% of the samples by default.

Effect of Structure By performing outlier detection over synthetic datasets that exhibit different strength of structure, we show that the advantage of Picket over the other outlier detection methods is its ability to capture the structure of the data. We fix $T = 10$ and vary r to change the strength of structure. Smaller r indicates stronger structure and more redundancy across attributes. The results are shown in Figure 4.13. Picket performs better when the structure is strong, while the performance of the other methods is not affected by the strength of structure, which indicates that Picket is able to capture the structure of the data and benefit from it.

Effect of Data Dimension We vary the the data dimension T to study how it affects the performance. The hidden dimension r is set to T so that the attributes are independent. The results are shown in Figure 4.14. The performance of all methods increases as the data dimension gets larger. The reason is that there are more corrupted cells in corrupted samples when the dimension increases, making them easier to be detected. Note that RVAE performs quite well in this setting, which is not surprising since it is built exactly on the assumption that the data come from Gaussian distributions.

Effect of Noise Level We study the effect of noise level, including the fraction of corrupted samples, the fraction of corrupted cells in corrupted samples (β) and the magnitude of the random noise (σ_1). Each time we vary one of the factors and fix the others. The data dimension T is fixed to 10, and r is fixed to 5. As is shown in Figure 4.15, when we vary the fraction of corrupted samples, the performance of all methods keeps stable. Figure 4.16 and 4.17 show that the performance of all methods increases as we increase the fraction of corrupted cells in corrupted samples or the magnitude of the random noise. These results show that the corruption level of the corrupted samples have a more significant effect on the outlier detection performance than the fraction of corrupted samples.

Effect of Extreme Outliers We study how the models behave under extreme outliers with different magnitude. We corrupt 20% of the samples, and among those samples 20% of the cells are multiplied by a scaling factor. We vary the value of the scaling factor and report the detection performance in Figure 4.18. As the scaling factor gets larger, the performance of all methods increases. This is expected since more extreme values deviate more from the normal distribution.

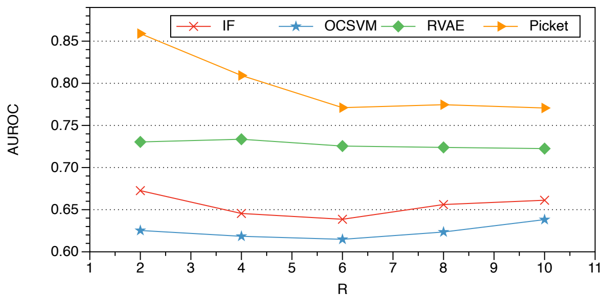


Figure 4.13: Training time outlier detection over synthetic datasets that exhibit different strength of structure.

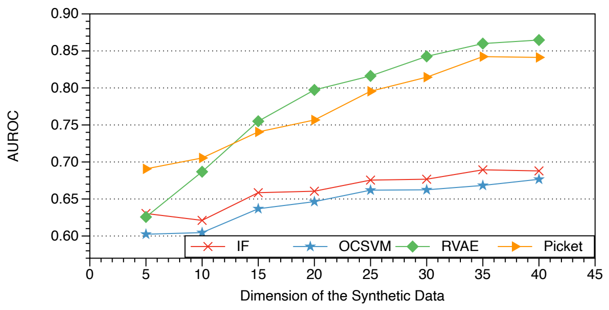


Figure 4.14: Training time outlier detection over synthetic datasets that have different dimensions.

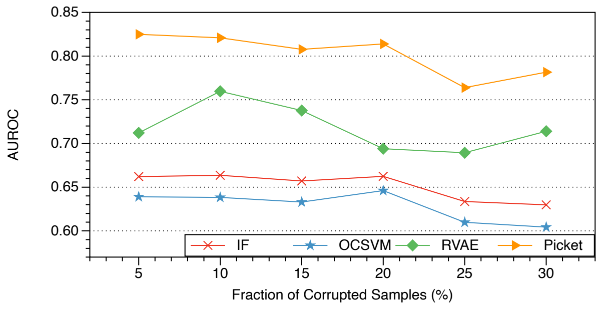


Figure 4.15: Training time outlier detection over synthetic datasets under different fractions of corrupted samples.

4.7.2 Outlier Detection with Cross-Validation

We evaluate the ability to detect outliers for unseen data during training using cross-validation. We use 5 iterations of 2-fold cross-validation with a modified t-test recommended by Dietterich and Thomas (Dietterich, 1998). Specifically, in each iteration, we randomly split the dataset into two folds. Then we use one fold to train the outlier detection models, and the other to validate their performance. The results are reported in Table 4.15. The results shows that Picket achieves the best performance among the examined methods on all dataset-noise combinations for unseen data at training time. In some cases, Picket is significantly better than all competing methods.

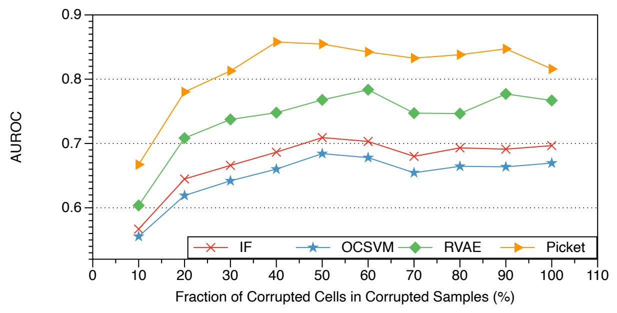


Figure 4.16: Training time outlier detection over synthetic datasets under different fractions of corrupted cells in corrupted samples.

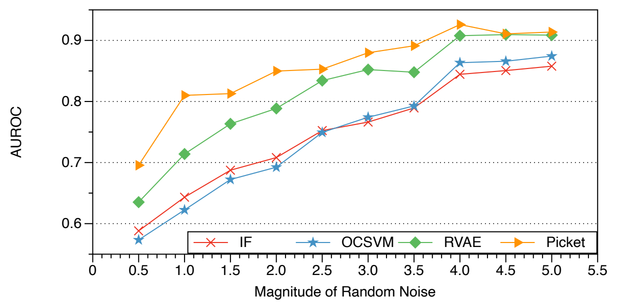


Figure 4.17: Training time outlier detection over synthetic datasets under different noise magnitudes.

4.7.3 Training Time Outlier Detection under Different Fraction of Corrupted Samples

We vary the fraction of corrupted samples, and report the corresponding AUROC of training time outlier detection in Figure 4.19. The datasets we use are Wine with poisoning attack on NN, Adult with systematic noise, and Marketing with random noise. The random and systematic noise is in the Medium level.

From the results we can see that the detection performance could either increase or decrease as the fraction of corrupted samples grows, depending on the type of noise and detection method. One one hand, the outliers are easier to detect when they form a larger cluster; one the other hand,

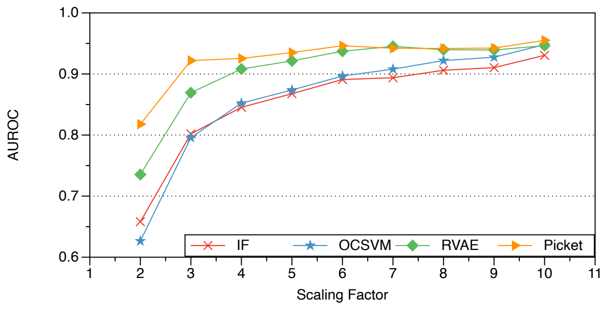


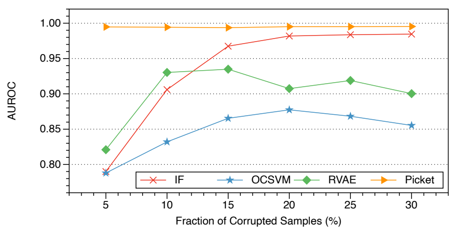
Figure 4.18: Training time outlier detection over synthetic datasets under different levels of extreme values.

Table 4.15: AUROC of outlier detection from cross-validation. The numbers are made bold when the corresponding method is significantly better (p value less than 0.05) than all the others.

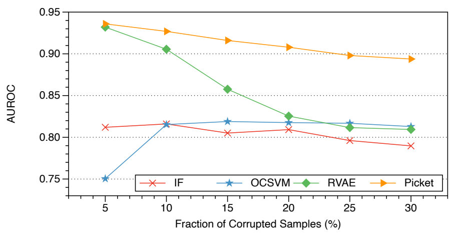
Dataset	Noise Type	IF	OCSVM	RVAE	Picket
Wine	Random-Medium	0.8876	0.8886	0.9170	0.9252
	Systematic-Medium	0.9537	0.8971	0.9123	0.9669
	Poison-LR	0.9756	0.9054	0.9061	0.9781
	Poison-SVM	0.9761	0.9047	0.9025	0.9787
	Poison-NN	0.9877	0.8696	0.9356	0.9921
Adult	Random-Medium	0.7800	0.8260	0.9019	**0.9240**
	Systematic-Medium	0.8048	0.8217	0.8530	**0.9180**
Restaurant	Random-Medium	0.4814	0.4431	0.6985	**0.9281**
	Systematic-Medium	0.4805	0.4449	0.6596	**0.8778**
	Real*	0.5514	0.5116	0.4558	**0.8978**
Marketing	Random-Medium	0.7539	0.7804	0.8688	0.8646
	Systematic-Medium	0.6746	0.6632	0.7787	0.7810
Titanic	Random-Medium	0.6014	0.6933	0.5819	**0.7709**
	Systematic-Medium	0.5811	0.7037	0.5557	0.7691
	Real	0.5851	0.6472	0.5000	0.7314
Food	Real	0.5094	0.5210	0.5180	0.5506

*Real is short for common errors in the real world.

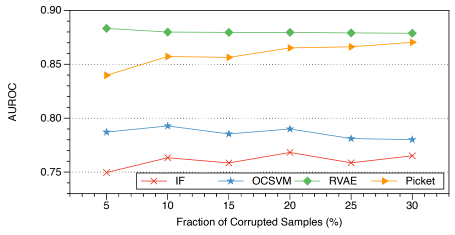
more corrupted samples may mislead the learning of the clean distribution. Nevertheless, Picket keeps a relatively consistent performance with either large or small fraction of corrupted samples, while other methods may have a large gap when the fraction varies.



(a)



(b)



(c)

Figure 4.19: AUROC of outlier detection under different fractions of corrupted samples. (a) Wine under Poisoning Attack. (b) Adult under Systematic Noise (Medium). (c) Marketing under Random Noise (Medium).

4.7.4 Training Time Outlier Detection under Low/High Level Random/Systematic Noise

We depict the AUROC of training time outlier detection under low/high level random/systematic noise in Figure 4.20, 4.21, 4.22, 4.23, when 20% of the samples are corrupted. The observation is quite similar to the case of medium level noise. The performance of Picket is quite good and consistent across different datasets and noise settings. RF and OCSVM perform poorly on the datasets that contain textual attributes. RVAE is competitive in some cases but fails in the others. Note that low level noise is much harder to detect than high level noise. The reason is that samples with high level noise tend to deviate a lot from the clean distribution, while samples with low level noise look quite similar to the clean ones and may not be detectable in some cases. However, low level noise will not affect the downstream model as much as high level noise, unless it is adversarial.

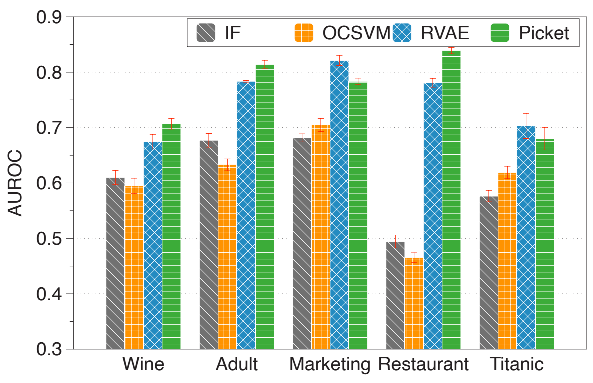


Figure 4.20: AUROC of outlier detection for random noise (Low level). The error bars represent the standard errors.

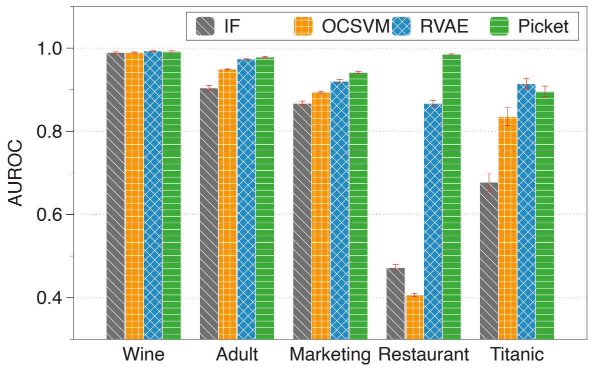


Figure 4.21: AUROC of outlier detection for random noise (High level). The error bars represent the standard errors.

4.7.5 Accuracy of Downstream Models under Random/Systematic Noise

We also study how the accuracy of the downstream models changes when we apply different filtering methods under random and systematic noise. We first focus on random noise. The results are shown in Tables 4.16, 4.17, 4.18. As expected, in the presence of random noise, the performance of the downstream models drops in some cases and remains roughly the same in the other cases if we look at CL and NF. In the cases when the

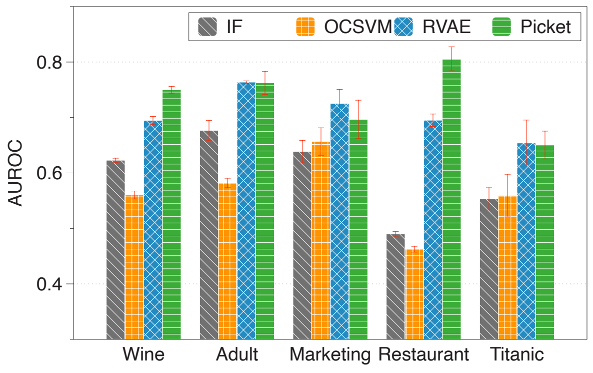


Figure 4.22: AUROC of outlier detection for systematic noise (Low level). The error bars represent the standard errors.

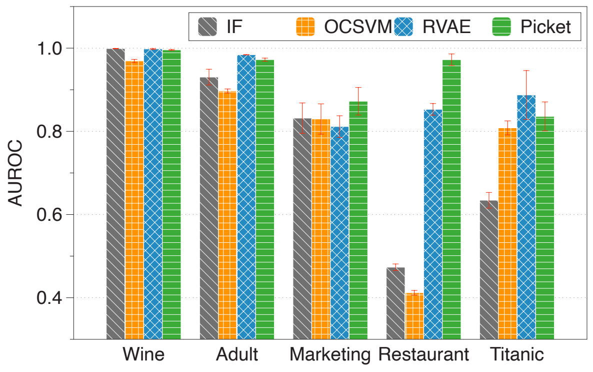


Figure 4.23: AUROC of outlier detection for systematic noise (High level). The error bars represent the standard errors.

downstream accuracy drops, we can see that filtering helps most of the time.

If we compare the performance of Picket and NF in Table 4.17 for Neural Networks, we see that for Adult, Titanic, and Restaurant Picket exhibits slightly worse test accuracy. These results are attributed to the selected thresholds for filtering in Picket (see Section 4.5). In Figure 4.24, we show the test accuracy of the downstream neural network for different levels of the Picket threshold. We can see that for some datasets, random noise serves as regularization and improves the performance of the downstream model. Therefore, we need to tune the threshold to achieve the best performance.

Table 4.16: Test accuracy of downstream models under random noise (Low level) and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Wine	LR	0.7429	**0.7435**	0.7427	0.7429	0.7457	0.7443
	SVM	0.7447	0.7437	**0.7486**	0.7465	0.7465	0.7453
	NN	0.7857	0.7800	0.7849	**0.7941**	0.8051	0.7922
Adult	LR	0.8207	0.8211	0.8127	**0.8233**	0.8240	0.8190
	SVM	0.8181	0.8196	0.8075	**0.8212**	0.8238	0.8187
	NN	**0.7818**	0.7800	0.7803	0.7816	0.7909	0.7836
Restaurant	LR	0.7318	0.7347	**0.7361**	0.7352	0.7375	0.7378
	SVM	0.6922	0.7078	**0.7123**	0.6972	0.7116	0.7060
	NN	0.7128	0.6982	0.7099	**0.7135**	0.7306	0.7182
Marketing	LR	0.7622	0.7661	0.7642	**0.7663**	0.7672	0.7691
	SVM	0.7649	0.7668	0.7655	**0.7678**	0.7681	0.7708
	NN	**0.7362**	0.7282	0.7302	0.7265	0.7261	0.7300
Titanic	LR	0.7810	0.7777	0.7832	**0.7844**	0.7877	0.7821
	SVM	0.7799	0.7866	0.7788	**0.7877**	0.7888	0.7888
	NN	**0.7654**	0.7542	0.7531	**0.7654**	0.7743	0.7709

*DM = Downstream Model.

Table 4.17: Test accuracy of downstream models under random noise (Medium level) and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Wine	LR	**0.7410**	0.7396	**0.7410**	0.7398	0.7457	0.7280
	SVM	0.7441	**0.7457**	0.7443	0.7431	0.7467	0.7259
	NN	0.7743	0.7776	**0.7816**	0.7776	0.7973	0.7761
Adult	LR	0.8140	0.8220	**0.8233**	0.8224	0.8240	0.8111
	SVM	0.8109	0.8200	**0.8219**	0.8207	0.8238	0.8082
	NN	**0.7856**	0.7795	0.7830	0.7850	0.7934	0.7883
Restaurant	LR	0.7342	0.7321	0.7313	**0.7366**	0.7375	0.7349
	SVM	**0.7111**	0.7083	0.6898	0.6858	0.7185	0.6872
	NN	0.7059	0.7064	0.7062	**0.7157**	0.7298	0.7210
Marketing	LR	0.7645	0.7624	0.7642	**0.7656**	0.7672	0.7665
	SVM	0.7654	0.7639	0.7654	**0.7665**	0.7681	0.7669
	NN	0.7267	**0.7360**	0.7301	0.7344	0.7311	0.7310
Titanic	LR	0.7799	0.7821	0.7777	**0.7877**	0.7877	0.7754
	SVM	0.7810	0.7765	0.7788	**0.7933**	0.7888	0.7821
	NN	0.7575	0.7665	0.7408	**0.7765**	0.7944	0.7844

*DM = Downstream Model.

Table 4.18: Test accuracy of downstream models under random noise (High level) and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	Picket	CL	NF
Wine	LR	0.7410	0.7406	0.7398	**0.7418**	0.7457	0.6861
	SVM	0.7441	0.7414	0.7427	**0.7453**	0.7469	0.6806
	NN	0.7865	0.7839	**0.7896**	0.7806	0.7941	0.7780
Adult	LR	0.8047	0.8196	0.8218	**0.8224**	0.8240	0.8002
	SVM	0.8024	0.8196	**0.8207**	0.8205	0.8238	0.7971
	NN	0.7853	0.7763	**0.7867**	0.7861	0.7982	0.7863
Restaurant	LR	**0.7380**	0.7369	0.7335	0.7327	0.7375	0.7416
	SVM	**0.7161**	0.7060	0.7154	0.7126	0.7053	0.6872
	NN	0.7147	0.7172	0.7155	**0.7206**	0.7251	0.7247
Marketing	LR	0.7653	0.7649	0.7641	**0.7668**	0.7672	0.7671
	SVM	0.7660	0.7660	0.7659	**0.7699**	0.7681	0.7686
	NN	0.7255	0.7265	**0.7284**	0.7271	0.7245	0.7295
Titanic	LR	**0.7877**	0.7777	0.7799	0.7799	0.7877	0.7877
	SVM	**0.7922**	0.7810	0.7855	0.7799	0.7888	0.7844
	NN	0.7609	0.7687	0.7709	**0.7765**	0.7866	0.7832

*DM = Downstream Model.

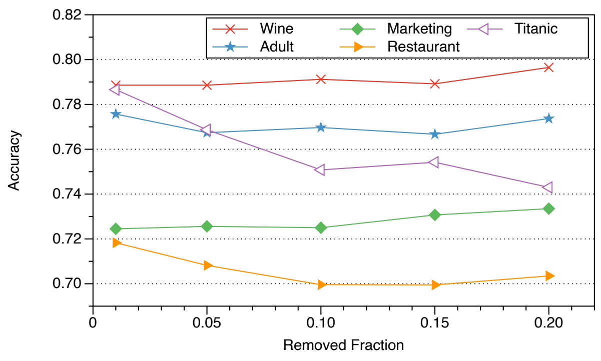


Figure 4.24: Changes in test accuracy of a neural network when filtering different fraction of the points; random noise (Medium level).

We then turn our attention to systematic noise. The results are shown in Table 4.19, 4.20, 4.21. PICKET performs the best in most cases, but still the numbers are quite close. Under common errors in the real world, CL and NF are also quite close, and filtering does not help.

Table 4.19: Test accuracy of downstream models under systematic noise (Low level) and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Wine	LR	0.7418	0.7424	**0.7478**	0.7473	0.7457	0.7408
	SVM	0.7422	0.7453	**0.7498**	0.7492	0.7473	0.7484
	NN	0.7876	0.7890	0.7882	**0.7976**	0.8045	0.7939
Adult	LR	**0.8224**	0.8205	0.8209	0.8189	0.8240	0.8200
	SVM	**0.8203**	0.8196	0.8165	0.8170	0.8238	0.8186
	NN	**0.7816**	0.7746	0.7748	0.7779	0.7955	0.7815
Restaurant	LR	0.7336	0.7339	**0.7359**	0.7336	0.7375	0.7356
	SVM	0.7063	0.6863	0.7035	**0.7082**	0.7108	0.7047
	NN	0.7113	0.7072	0.7079	**0.7160**	0.7301	0.7201
Marketing	LR	0.7639	0.7630	0.7616	**0.7644**	0.7672	0.7668
	SVM	0.7658	0.7634	0.7614	**0.7683**	0.7681	0.7676
	NN	0.7316	0.7305	**0.7329**	0.7312	0.7324	0.7325
Titanic	LR	0.7866	0.7888	0.7799	**0.7989**	0.7877	0.7821
	SVM	0.7899	0.7866	0.7754	**0.8022**	0.7888	0.7911
	NN	0.7575	0.7520	0.7564	**0.7598**	0.7944	0.8011

*DM = Downstream Model.

4.7.6 Test Time Victim Sample Detection under Low/High Level Random/Systematic Noise

In Table 4.23, 4.24, 4.25, 4.26, we show the F1 scores of victim sample detection under low/high level random/systematic noise. The artificial noise setting is the same as described in Section 4.6.3. We can see that PICKET outperforms all the other methods in most cases. MWOC performs quite well for the Wine dataset, but it fails completely under high random noise (the F1 score is 0.33). Similar to the case of medium noise, we observe that the reconstruction loss from PICKETNET provides signals that improve the detection performance (see the comparison between RF and PICKET).

Table 4.20: Test accuracy of downstream models under systematic noise (Medium level) and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Wine	LR	0.7414	0.7388	0.7435	**0.7445**	0.7457	0.7316
	SVM	0.7441	0.7384	0.7459	**0.7463**	0.7461	0.7316
	NN	**0.7959**	0.7933	0.7918	0.7953	0.8000	0.7855
Adult	LR	0.8136	0.8156	**0.8207**	0.8171	0.8240	0.8098
	SVM	0.8103	0.8142	**0.8178**	0.8159	0.8238	0.8080
	NN	0.7822	0.7839	**0.7843**	0.7837	0.7931	0.7869
Restaurant	LR	0.7305	0.7315	0.7351	**0.7383**	0.7375	0.7372
	SVM	0.7070	0.7008	**0.7107**	0.7077	0.7136	0.6964
	NN	0.7198	0.7154	0.7175	**0.7228**	0.7346	0.7215
Marketing	LR	0.7642	0.7640	0.7660	**0.7673**	0.7672	0.7664
	SVM	0.7670	0.7658	0.7655	**0.7686**	0.7681	0.7686
	NN	0.7272	**0.7311**	0.7251	0.7281	0.7277	0.7295
Titanic	LR	**0.7877**	0.7821	0.7799	0.7866	0.7877	0.7877
	SVM	0.7922	0.7777	0.7821	**0.8022**	0.7888	0.7911
	NN	0.7464	0.7508	0.7464	**0.7553**	0.7866	0.7777

*DM = Downstream Model.

Table 4.21: Test accuracy of downstream models under systematic noise (High level) and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Wine	LR	0.7437	0.7359	0.7443	**0.7447**	0.7457	0.7100
	SVM	0.7457	0.7365	**0.7476**	0.7455	0.7467	0.7041
	NN	0.7961	0.7961	0.7990	**0.8008**	0.7992	0.8031
Adult	LR	0.8071	0.8055	**0.8193**	0.8079	0.8240	0.8011
	SVM	0.8039	0.8038	**0.8175**	0.8060	0.8238	0.8002
	NN	**0.7843**	0.7800	0.7834	0.7822	0.7961	0.7885
Restaurant	LR	0.7329	0.7332	0.7346	**0.7371**	0.7375	0.7361
	SVM	0.7155	0.7051	0.7041	**0.7187**	0.6726	0.6925
	NN	0.7100	0.7032	**0.7132**	0.7111	0.7232	0.7124
Marketing	LR	0.7653	**0.7655**	0.7638	0.7636	0.7672	0.7656
	SVM	0.7656	**0.7661**	0.7646	0.7640	0.7681	0.7678
	NN	0.7292	**0.7304**	0.7256	0.7258	0.7303	0.7294
Titanic	LR	0.7777	0.7788	**0.7821**	0.7799	0.7877	0.7877
	SVM	0.7799	**0.7855**	**0.7855**	0.7799	0.7888	0.7866
	NN	0.7553	0.7598	**0.7654**	0.7441	0.7855	0.7832

*DM = Downstream Model.

Table 4.22: Test accuracy of downstream models under common errors in the real world and different filtering methods.

Dataset	DM*	IF	OCSVM	RVAE	PICKET	CL	NF
Restaurant	LR	**0.7388**	0.7351	0.7328	0.7351	0.7404	0.7395
	SVM	0.7028	0.6937	0.6922	**0.7072**	0.6959	0.7112
	NN	0.7187	0.7176	**0.7204**	0.7137	0.7118	0.7215
Titanic	LR	0.7464	0.7497	**0.7732**	0.7475	0.7799	0.7609
	SVM	0.7363	0.7363	**0.7609**	0.7520	0.7542	0.7598
	NN	0.7274	0.7251	0.7285	**0.7318**	0.7095	0.7207
Food	LR	0.6628	0.6960	0.6917	**0.6978**	0.7163	0.6868
	SVM	0.6529	**0.6849**	0.6720	0.6794	0.7095	0.7108
	NN	0.6505	0.6443	0.6431	**0.6560**	0.6609	0.6597

*DM = Downstream Model.

Table 4.23: F_1 scores of victim sample detection at inference time under random noise (Low level).

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	PICKET
Wine	LR	0.7408	0.6910	0.7523	0.6667	0.6626	0.4971	**0.8084**	0.7824
	SVM	0.7440	0.6918	0.7558	0.6667	0.6638	0.6016	**0.8004**	0.7828
	NN	0.6882	0.6318	0.6456	0.6770	0.6656	0.5231	**0.7202**	0.6713
Adult	LR	0.8393	0.6563	0.8486	0.6696	0.7834	0.1968	0.7902	**0.8685**
	SVM	0.8456	0.6743	0.8535	0.6691	0.8131	0.4602	0.7114	**0.8714**
	NN	0.8017	0.5429	0.8052	0.6635	0.6806	0.1900	0.7965	**0.8267**
Restaurant	LR	0.7870	–#	–	0.7586	0.6702	0.6441	0.7649	**0.8328**
	SVM	0.6370	–	–	0.6895	0.6351	0.6634	0.5538	**0.7123**
	NN	0.7609	–	–	0.7066	0.6643	0.6071	0.7075	**0.8119**
Marketing	LR	0.8503	0.6340	0.8565	0.7771	0.7913	0.6630	0.8227	**0.8662**
	SVM	0.8590	0.6324	0.8635	0.7789	0.8034	0.6636	0.7748	**0.8720**
	NN	0.7917	0.6197	0.7986	0.6809	0.7134	0.6665	0.7128	**0.8125**
Titanic	LR	0.8281	–	–	0.7060	0.6487	0.4377	0.7917	**0.8451**
	SVM	0.8547	–	–	0.6750	0.6544	0.6489	0.7738	**0.8731**
	NN	0.8343	–	–	0.6678	0.6432	0.1717	0.7798	**0.8544**

*DM is short for Downstream Model. #RVAE is not applicable to textual attributes.

Table 4.24: F_1 scores of victim sample detection at inference time under random noise (High level).

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	Picket
Wine	LR	0.7525	0.7867	0.7950	0.6657	0.6727	0.5901	0.5860	**0.8059**
	SVM	0.7496	0.7898	0.7984	0.6633	0.6815	0.7256	0.7295	**0.8030**
	NN	0.6805	0.7697	**0.7887**	0.4560	0.6668	0.5752	0.3301	0.7803
Adult	LR	0.7969	0.7725	0.8149	0.6570	0.7593	0.2408	0.5033	**0.8273**
	SVM	0.8035	0.7765	0.8201	0.6580	0.7700	0.4737	0.4909	**0.8312**
	NN	0.7952	0.7781	0.8124	0.3089	0.6988	0.4284	0.4234	**0.8214**
Restaurant	LR	0.7457	–$^{\#}$	–	0.7075	0.6506	0.6504	0.7111	**0.8137**
	SVM	0.6948	–	–	0.6704	0.6553	0.6567	0.5964	**0.7824**
	NN	0.7437	–	–	0.6788	0.6642	0.6119	0.6852	**0.8135**
Marketing	LR	0.8118	0.7044	0.8146	0.7052	0.7566	0.6645	0.7590	**0.8244**
	SVM	0.8111	0.7022	0.8156	0.6994	0.7527	0.6652	0.7486	**0.8247**
	NN	0.7934	0.7068	0.7999	0.6085	0.7042	0.6630	0.7042	**0.8038**
Titanic	LR	0.8134	–	–	0.6437	0.6457	0.4383	0.7153	**0.8227**
	SVM	**0.8113**	–	–	0.6533	0.6354	0.6444	0.6815	0.8105
	NN	0.7993	–	–	0.6516	0.6328	0.2824	0.6505	**0.8058**

*DM is short for Downstream Model. $^{\#}$RVAE is not applicable to textual attributes.

Table 4.25: F_1 scores of victim sample detection at inference time under Systematic noise (Low level).

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	Picket
Wine	LR	0.6826	0.5225	0.6632	0.6667	0.6474	0.4203	**0.8063**	0.7039
	SVM	0.6658	0.5252	0.6566	0.6667	0.6328	0.4835	**0.7933**	0.6915
	NN	0.6741	0.6010	0.5601	0.6856	0.6661	0.4980	**0.6985**	0.6058
Adult	LR	0.8146	0.6291	0.8176	0.6696	0.7463	0.1842	0.7412	**0.8317**
	SVM	0.8360	0.6277	0.8418	0.6694	0.7952	0.3382	0.6374	**0.8589**
	NN	0.8100	0.5607	0.8208	0.6026	0.6763	0.1878	0.7740	**0.8262**
Restaurant	LR	0.7951	–$^{\#}$	–	0.7725	0.6274	0.6460	0.7770	**0.8269**
	SVM	0.7080	–	–	0.6524	0.6585	0.6488	0.5976	**0.7321**
	NN	0.7633	–	–	0.7143	0.6588	0.6080	0.7043	**0.7897**
Marketing	LR	0.8540	0.6090	0.8606	0.7855	0.7923	0.6615	0.8274	**0.8724**
	SVM	0.8597	0.6214	0.8590	0.7939	0.7936	0.6629	0.7828	**0.8676**
	NN	0.7892	0.5557	0.7899	0.6864	0.7142	0.6658	0.6819	**0.7972**
Titanic	LR	0.8064	–	–	0.7235	0.6409	0.3751	0.7684	**0.8300**
	SVM	0.8563	–	–	0.6778	0.6361	0.6498	0.7867	**0.8656**
	NN	0.8314	–	–	0.6679	0.6462	0.1507	0.7667	**0.8434**

*DM is short for Downstream Model. $^{\#}$RVAE is not applicable to textual attributes.

Table 4.26: F_1 scores of victim sample detection at inference time under Systematic noise (High level).

Dataset	DM*	RF	RVAE	RVAE+	CCS	KNN	TOAO	MWOC	PICKET
Wine	LR	0.6866	0.3982	0.6697	0.6667	0.6440	0.3612	**0.7826**	0.6918
	SVM	0.6784	0.4293	0.6712	0.6667	0.6175	0.4102	**0.7688**	0.6878
	NN	0.6701	0.6127	0.5913	0.6876	0.6656	0.5009	**0.7536**	0.5967
Adult	LR	0.8100	0.7619	0.8120	0.6699	0.7234	0.1846	0.7431	**0.8370**
	SVM	0.8156	0.7507	0.8174	0.6694	0.7463	0.3736	0.6833	**0.8313**
	NN	0.8086	0.7341	0.8186	0.4264	0.6883	0.2859	0.7701	**0.8285**
Restaurant	LR	0.7552	–#	–	0.7156	0.6475	0.6525	0.7221	**0.8136**
	SVM	0.7017	–	–	0.6693	0.6626	0.6594	0.5877	**0.7705**
	NN	0.7523	–	–	0.6853	0.6667	0.6123	0.7003	**0.8149**
Marketing	LR	0.8232	0.6981	0.8285	0.7423	0.7620	0.6634	0.7864	**0.8406**
	SVM	0.8361	0.6703	0.8387	0.7138	0.7701	0.6653	0.7433	**0.8483**
	NN	0.7896	0.6960	0.7991	0.6413	0.7066	0.6623	0.7176	**0.8092**
Titanic	LR	0.8255	–	–	0.6843	0.6298	0.4501	0.7830	**0.8270**
	SVM	0.7945	–	–	0.6517	0.6120	0.6686	0.6815	**0.7972**
	NN	0.8240	–	–	0.6665	0.6349	0.2243	0.7519	**0.8347**

*DM is short for Downstream Model. #RVAE is not applicable to textual attributes.

4.8 Robust Mean Estimation under Coordinate-Level Corruption

To understand the relationship between data corruption and learning performance, we study robust mean estimation under fine-grained corruption schemes. First, we introduce two new coordinate-level corruption adversaries (models) and a new measure of distribution shift (d_{ENTRY}) that characterizes the effect of those adversaries on the observed distribution. Second, we present an information theoretic analysis of robust mean estimation and prove information-theoretically optimal bounds for mean estimation over Gaussians $\mathcal{N}(\mu, \mathbf{\Sigma})$ under coordinate-level corruption with respect to Mahalanobis distance. The results hold for replacement-based corruptions as well as missing values.

4.8.1 Background

We review the problem of robust mean estimation and discuss models and measures related to our study.

Robust Mean Estimation Robust mean estimation seeks to recover the mean $\mu \in \mathbb{R}^n$ of a n-dimensional distribution D from a list of i.i.d. samples where an unknown number of arbitrary corruptions has been introduced in the samples. Given access to a collection of N samples $x_1, x_2, \ldots, x_N$ from D on $\mathbb{R}^n$ when a fraction of them have been fully or partially corrupted, robust mean estimation seeks to find a vector $\hat{\mu}$ such that $\|\mu - \hat{\mu}\|$ is as small as possible. We consider two norms to measure the mean estimation error. The first norm is the Euclidean (ℓ_2) distance and the second is the scale-invariant *Mahalanobis distance* defined as $\|\mu - \hat{\mu}\|_{\mathbf{\Sigma}} = |(\mu - \hat{\mu})^T \mathbf{\Sigma}^{-1} (\mu - \hat{\mu})|^{1/2}$, where $\mathbf{\Sigma}$ is the covariance matrix. When the covariance matrix is the identity matrix, the Mahalanobis distance reduces to the Euclidean distance.

Sample-level Corruption A typical model to describe worst-case corruptions is that of a *sample-level adversary*, hereafter denoted $\mathcal{A}_1^\epsilon$. Corruptions introduce a shift of the distribution D, which we can measure using the *total variation distance* (d_{TV}). Total variation distance between two distributions P and Q on $\mathbb{R}^n$ is defined as $d_{\text{TV}}(P, Q) = \sup_{E \subseteq \mathbb{R}^n} |P(E) - Q(E)|$ or equivalently $\frac{1}{2}\|P - Q\|_1$. For two Gaussians $D_1 = \mathcal{N}(\mu_1, \mathbf{\Sigma})$ and $D_2 = \mathcal{N}(\mu_2, \mathbf{\Sigma})$ with $d_{\text{TV}}(D_1, D_2) = \epsilon < 1/2$ it is that $\|\mu_1 - \mu_2\|_{\mathbf{\Sigma}} = \Theta(\epsilon)$, i.e., their total variation distance and the Mahalanobis distance of their means are equivalent up to constants. This result allows tight analyses of Gaussian mean estimation for a bounded fraction of corruptions.

4.8.2 Coordinate-level Corruption Adversaries

We introduce two new adversaries and compare them to the sample-level adversary $\mathcal{A}_1^\epsilon$ from Section 4.8.1:

First, we consider an extension of $\mathcal{A}_1^\epsilon$ to coordinates, and define a *value-*

fraction adversary, denoted by $\mathcal{A}_2^\rho$. Given N samples from distribution D on $\mathbb{R}^n$, adversary $\mathcal{A}_2^\rho$ is allowed to corrupt up to a ρ-fraction of values in each coordinate of the N samples. This adversary can corrupt a total of $\rho \cdot N \cdot n$ values in the N samples; these values can be distributed strategically across samples leading to cases where *most* of the samples are corrupted but still the corruption per coordinate is bounded by ρN.

Second, we define the more powerful *coordinate-fraction adversary* $\mathcal{A}_3^\alpha$ that can corrupt *all samples* in the worst case. $\mathcal{A}_3^\alpha$ is allowed to corrupt up to α-fraction of all values in the N samples, i.e., up to a total of $\alpha \cdot N \cdot n$ values. When $\alpha \geq \frac{1}{n}$, adversary $\mathcal{A}_3^\alpha$ can corrupt *all* N samples.

Adversary Comparison $\mathcal{A}_1^\epsilon$ corresponds to the standard adversary associated with the strong contamination model considered by Diakonikolas and Kane (2019), which either corrupts a sample completely or leaves it intact. Adversaries $\mathcal{A}_2^\rho$ and $\mathcal{A}_3^\alpha$ are more fine-grained since they can corrupt only part of the entries of a sample. As a result, $\mathcal{A}_2^\rho$ and $\mathcal{A}_3^\alpha$ can corrupt more samples than $\mathcal{A}_1^\epsilon$ for similar budget-fractions ϵ, ρ, and α.

We formalize the comparison among $\mathcal{A}_1^\epsilon$, $\mathcal{A}_2^\rho$, and $\mathcal{A}_3^\alpha$ in the next propositions. We seek to understand when an adversary $\mathcal{A}$ can *simulate* another $\mathcal{A}'$, i.e., $\mathcal{A}$ can perform any corruption performed by $\mathcal{A}'$.

Proposition 2. *If α, $\rho \leq \epsilon/n$, then $\mathcal{A}_1^\epsilon$ can simulate $\mathcal{A}_2^\rho$ and $\mathcal{A}_3^\alpha$. If $\alpha \leq \rho/n$, $\mathcal{A}_2^\rho$ can simulate $\mathcal{A}_3^\alpha$.*

Proposition 3. *If α, $\rho \geq \epsilon$, then $\mathcal{A}_2^\rho$ and $\mathcal{A}_3^\alpha$ can simulate $\mathcal{A}_1^\epsilon$. If $\alpha \geq \rho$, $\mathcal{A}_3^\alpha$ can simulate $\mathcal{A}_2^\rho$.*

These propositions show that the two adversary types (sample- and coordinate-level) can simulate each other under different budget conditions, thus, enabling reductions between the two types.

Proposition 2 implies that we can reduce coordinate-level corruption to sample-level corruption by considering $\mathcal{A}_3^\alpha$ as $\mathcal{A}_1^\epsilon$ with $\epsilon = \alpha n$. This reduction guarantees that *any algorithm for mean estimation with guarantees* *for $\mathcal{A}_1^\epsilon$ enjoys the same guarantees for coordinate-level corruption when $\epsilon \geq \alpha \cdot n$.* Similarly, Proposition 3 means that *any lower-bound guarantee on mean estimation for $\mathcal{A}_1^\epsilon$ also holds for $\mathcal{A}_3^\alpha$ when $\epsilon = \alpha$.* However, this characterization is loose as the gap between α and αn is large, raising the question: Are there distributions for which this gap is more tight and are there data properties we can exploit to reduce the dimensional factor of n? Next, we show that structure in data affects the power of coordinate-level corruption and introduces information-theoretically tight bounds for mean estimation under coordinate-level corruption.

4.8.3 Distribution Shift in Coordinate-level Corruption

We propose a new type of distribution shift metric, referred to as d_{ENTRY}, which can capture fine-grained coordinate-level corruption:

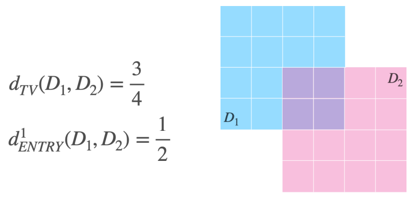
Definition 4.1 (d_{ENTRY}). *Consider the coupling γ of two distributions P, Q, i.e., a joint distribution of P and Q such that the marginal distributions are P, Q. Let the set of all couplings of P, Q be $\Gamma(P, Q)$, and define for $x, y \in \mathbb{R}^n$, $I(x, y) = [\mathbb{1}_{x_1 \neq y_1}, \ldots, \mathbb{1}_{x_n \neq y_n}]^\top$. For D_1, D_2 on $\mathbb{R}^n$,*

$$d^1_{ENTRY}(D_1, D_2) = \inf_{\gamma \in \Gamma(D_1, D_2)} \frac{1}{n} \| \mathop{\mathbb{E}}_{(x,y)\sim\gamma} [I(x, y)] \|_1$$

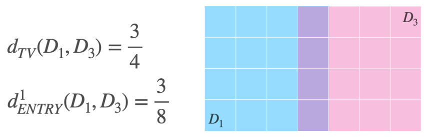
$$d^\infty_{ENTRY}(D_1, D_2) = \inf_{\gamma \in \Gamma(D_1, D_2)} \| \mathop{\mathbb{E}}_{(x,y)\sim\gamma} [I(x, y)] \|_\infty$$

The following theorem shows the relation between d^1_{ENTRY} and $\mathcal{A}_3^\alpha$.

Theorem 4.2. *Let D_1, D_2 be two distributions such that $d^1_{ENTRY}(D_1, D_2) = \alpha'$. $\mathcal{A}_3^\alpha$ corrupts α fraction of N samples from D_1. If $\alpha > 2\alpha'$, $\mathcal{A}_3^\alpha$ has a way to make corruptions so that with probability at least $1 - e^{-\Omega(\alpha^2 N)}$ it is indistinguishable whether the N samples come from D_1 or D_2. If $\alpha < \alpha'/4$,*


$d_{TV}(D_1, D_2) = \frac{3}{4}$
$d^1_{ENTRY}(D_1, D_2) = \frac{1}{2}$
D_1
D_2

(a) Corruption of both coordinates


$d_{TV}(D_1, D_3) = \frac{3}{4}$
$d^1_{ENTRY}(D_1, D_3) = \frac{3}{8}$
D_1
D_3

(b) Corruption of one coordinate

Figure 4.25: Comparing d_{TV} and d_{ENTRY}: D_1 is the original 2D uniform distribution. D_2 is obtained after corruptions from $\mathcal{A}_3^\alpha$ with $\alpha = 1/2$, and D_3 after corruption from $\mathcal{A}_3^\alpha$ with $\alpha = 3/8$.

no matter how $\mathcal{A}_3^\alpha$ makes corruptions, with probability at least $1 - e^{-\Omega(\alpha^2 N)}$, we can tell that the N samples come from D_1.

The relation in the above theorem also holds for d^∞_{ENTRY} and $\mathcal{A}_2^\rho$. The theorem shows that d_{ENTRY} gives a tight asymptotic bound on the power of coordinate-level adversaries.

Intuitively $d^1_{\text{ENTRY}}(D_1, D_2)$ represents how many coordinates need to be corrupted (out of n on average) for D_1 and D_2 to be indistinguishable. Then, given the original distribution D and sufficiently large sample size, $\{D' : d^1_{\text{ENTRY}}(D, D') \leq \alpha\}$ represents the set of distributions that $\mathcal{A}_3^\alpha$ can show us after corruption, and thus d^1_{ENTRY} allows us to capture all possible actions of this adversary. Similarly, d^∞_{ENTRY} captures all possible actions of $\mathcal{A}_2^\rho$. We use d_{ENTRY} when both d^1_{ENTRY} and d^∞_{ENTRY} apply.

We compare d^1_{ENTRY} and d_{TV} in Figure 4.25. We consider a 2D uniform distribution D_1 and two corrupted versions D_2 and D_3. D_2 is obtained after an adversary corrupts both coordinates for samples from the upper-left quadrant of D_1 and one of the coordinates for samples in the lower-left and upper-right quadrant of D_1. D_3 is obtained after an adversary corrupts the horizontal coordinate for samples obtained from the left-most $3/4$ of D_1. In both cases, $d_{\text{TV}}(D_1, D_2) = d_{\text{TV}}(D_1, D_3) = 3/4$ since $3/4$ of the samples from D_1 are corrupted. d_{ENTRY} is different: From Definition 4.1, we have $d^1_{\text{ENTRY}}(D_1, D_2) = 1/2$ and $d^1_{\text{ENTRY}}(D_1, D_3) = 3/8$, thus, we can distinguish

Table 4.27: Our results for robust mean estimation ($||\hat{\mu} - \mu||_{\mathbf{\Sigma}}$) under $\mathcal{A}_1^\epsilon$ (sample-level adversary), $\mathcal{A}_2^\rho$, and $\mathcal{A}_3^\alpha$ (coordinate-level adversaries). The results are for Gaussian distributions.

Structure	$\mathcal{A}_1^\epsilon$	$\mathcal{A}_2^\rho$	$\mathcal{A}_3^\alpha$
No Structure	$\Theta(\epsilon)$	$\Omega(\rho\sqrt{n})$, $O(\rho n)$	$\Theta(\alpha n)$
Linear structure $\boldsymbol{A}$	$\Theta(\epsilon)$	$O(\rho\frac{n}{m_{\boldsymbol{A}}})$	$\Theta(\alpha\frac{n}{m_{\boldsymbol{A}}})$

the two due to the differences in the distances.

4.8.4 Information-theoretic Bounds for Gaussians

We analyze robust mean estimation under coordinate-level corruptions for Gaussian distributions, the de facto choice in the robust mean estimation literature. This choice enables us to draw comparisons to prior mean estimation approaches. Our results are summarized in Table 4.27. We first show an impossibility result for arbitrary Gaussian distributions: in the general case, the information-theoretic analysis based on d_{TV} and sample-level adversaries (Tukey, 1975; Diakonikolas et al., 2019a) is tight even for coordinate-level corruption adversaries. However, we show that this result *does not hold* for distributions that exhibit structure, i.e., redundancy across coordinates. We show that, for structured Gaussian distributions and corruptions that lead to a d_{ENTRY}-bounded distribution shift, one must exploit the structure to achieve information-theoretically optimal error.

Mean Estimation of Arbitrary Gaussians We consider a Gaussian distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$ with full rank covariance matrix $\mathbf{\Sigma}$. We assume that observed samples are corrupted by a coordinate-level adversary. We first present a common upper-bound on the mean estimation error for both $\mathcal{A}_2^\rho$ and $\mathcal{A}_3^\alpha$, and then introduce the corresponding lower-bounds.

We obtain an upper-bound on $\|\hat{\mu}-\mu\|_{\mathbf{\Sigma}}$ by using Proposition 2: A sample-level adversary can simulate a coordinate-level adversary when $\epsilon = \alpha \cdot n$. But, for Adversary $\mathcal{A}_1^{\epsilon}$ the Tukey median achieves optimal error $\|\hat{\mu} - \mu\|_{\mathbf{\Sigma}} = \Theta(\epsilon)$ when $\epsilon < 1/2$. Thus, the Tukey median yields error $O(\alpha n)$ for the coordinate-level adversaries $\mathcal{A}_2^{\rho}$ (when $\rho = \alpha$) and $\mathcal{A}_3^{\alpha}$, when $\alpha \cdot n < 1/2$. Note that the condition $\alpha \cdot n < 1/2$ is necessary for achieving such an upper bound. When $\alpha \cdot n \geq 1/2$, the coordinate-level adversary is able to corrupt more than half of the samples in the worst case, leading to unbounded error, which shows exactly the power of the coordinate-level adversary.

We now focus on lower-bounds for the mean estimation error. We first consider adversary $\mathcal{A}_2^{\rho}$ who can corrupt at most ρ-fraction of each coordinate in the samples. For this setting, the optimal estimation error depends on the *disc* of the covariance matrix $\mathbf{\Sigma}$, where disc is defined as:

Definition 4.3. *(disc) For a positive semi-definite matrix $\boldsymbol{M}$, define $s(\boldsymbol{M})_{ij} = M_{ij}/\sqrt{M_{ii}M_{jj}}$ and disc$(\boldsymbol{M}) = \max_{x \in [-1,1]} \sqrt{x^T s(\boldsymbol{M}) x}$.*

Theorem 4.4. *Let $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ be full rank. Given a set of i.i.d. samples from $\mathcal{N}(\mu, \mathbf{\Sigma})$ where the set is corrupted by $\mathcal{A}_2^{\rho}$, any algorithm for estimating $\hat{\mu}$ must satisfy $\|\mu - \hat{\mu}\|_{\mathbf{\Sigma}} = \Omega(\rho \cdot disc(\mathbf{\Sigma}^{-1}))$.*

From this theorem, we obtain the next corollary for the mean estimation error for $\mathcal{A}_2^{\rho}$:

Corollary 4.5. *Given a set of i.i.d. samples from $\mathcal{N}(\mu, \mathbf{\Sigma})$ where the set is corrupted by $\mathcal{A}_2^{\rho}$, any algorithm that outputs a mean estimator $\hat{\mu}$ must satisfy $\|\mu - \hat{\mu}\|_{\mathbf{\Sigma}} = \Omega(\rho\sqrt{n})$.*

We see that there is a gap between the lower and upper bound on the mean estimation error for $\mathcal{A}_2^{\rho}$. However, we show that such a gap does not hold for $\mathcal{A}_3^{\alpha}$. For $\mathcal{A}_3^{\alpha}$, the lower bound is the same as the upper bound presented above. Specifically, for the coordinate-fraction adversary $\mathcal{A}_3^{\alpha}$, it is impossible to achieve a mean estimation error better than $O(\alpha n)$ in the case of arbitrary Gaussian distributions:

Theorem 4.6. *Let $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ be full rank. Given a set of i.i.d. samples from $\mathcal{N}(\mu, \mathbf{\Sigma})$ where the set is corrupted by $\mathcal{A}_3^\alpha$, any algorithm that outputs a mean estimator $\hat{\mu}$ must satisfy $\|\hat{\mu} - \mu\|_{\mathbf{\Sigma}} = \Omega(\alpha n)$.*

To gain some intuition, consider $\mathcal{A}_3^\alpha$ with $\alpha \geq \frac{1}{n}$. In this case, $\mathcal{A}_3^\alpha$ can concentrate all corruption in the first coordinate of all samples, and hence, we cannot estimate the mean for that coordinate. An immediate result is that for worst-case coordinate-corruptions, i.e., corruptions introduced by $\mathcal{A}_3^\alpha$, over arbitrary Gaussian distributions the mean estimation error is precisely $\|\mu - \hat{\mu}\|_{\mathbf{\Sigma}} = \Theta(\alpha n)$.

Mean Estimation of Structured Gaussians The previous analysis for $\mathcal{A}_3^\alpha$ shows that we cannot improve upon existing algorithms. However, real-world data often exhibit structural relationships between features such that one may be able to infer corrupted values via other visible values (Wu et al., 2020). We show that in the presence of *structure* due to dependencies, one must exploit the structure of the data to achieve information-theoretically optimal error for mean estimation. To show that structure is key, we focus on samples $x_i \in \mathbb{R}^n$ that lie in a low-dimensional subspace such that $x_i = \boldsymbol{A} z_i$, where $\boldsymbol{A} \in \mathbb{R}^{n \times r}$ represents the structure. Such low-rank subspace-structure is natural in many real-world scenarios and we assume linearity for the convenience of analysis. In fact, linear structure can also encode more complex structures (e.g., polynomials) if one considers an augmented set of features. We assume z_i comes from a non-degenerate Gaussian in $\mathbb{R}^r$. We consider a data sample $x = \boldsymbol{A} z$ before corruption and assume that corruption is introduced in x.

In this setting, the coordinate-level adversary has limited effect in mean estimation due to the redundancy that $\boldsymbol{A}$ introduces. We can measure the strength of this redundancy with respect to coordinate-level corruption by considering its row space. The coordinates of $x = \boldsymbol{A} z$, and hence, the corrupted data, exhibit high redundancy when many rows of $\boldsymbol{A}$ span a small subspace. We define a quantity $m_{\boldsymbol{A}}$ to derive information-theoretic bounds for structured Gaussians.

Definition 4.7 ($m_{\boldsymbol{A}}$)**.** *Given matrix $\boldsymbol{A} \in \mathbb{R}^{n \times r}$, $m_{\boldsymbol{A}}$ is the minimum number of rows one needs to remove from $\boldsymbol{A}$ to reduce the dimension of its row space by one.*

When $\boldsymbol{A} = \boldsymbol{I}$ is the identity matrix, it is $m_{\boldsymbol{I}} = 1$ and we can remove any row to reduce its row space; we have low redundancy. But, for $\boldsymbol{A} = [\mathbf{e}_1, \ldots, \mathbf{e}_1]^\top$ where $\mathbf{e}_1$ has 1 in its first coordinate and 0 in the others, $m_{\boldsymbol{A}} = n$ since we need to remove all $\mathbf{e}_1$'s to reduce $\boldsymbol{A}$'s row space. It holds that $1 \leq m_{\boldsymbol{A}} \leq n$.

We next show that the higher the value that $m_{\boldsymbol{A}}$ takes, the weaker a coordinate-level adversary becomes due to the increased redundancy. Intuitively, the coordinate-level adversary has to spend more budget per sample to introduce corruptions that will counteract the redundancy introduced by matrix $\boldsymbol{A}$. Theorem 4.8 shows that $\mathcal{A}_2^\rho, \mathcal{A}_3^\alpha$ cannot alter the original distribution too far in d_{TV}, leading to information-theoretically tight bounds for mean estimation.

Theorem 4.8. *Given two probability distributions D_1, D_2 on $\mathbb{R}^n$ with support in the range of linear transformation $\boldsymbol{A}$,*

$$(m_{\boldsymbol{A}}/n) \cdot d_{TV}(D_1, D_2) \leq d_{ENTRY}(D_1, D_2) \leq d_{TV}(D_1, D_2)$$

Here, D with support in the range of $\boldsymbol{A}$ means a distribution D that is generated such that it lies on the subspace generated by $\boldsymbol{A}$, i.e., there is zero measure outside of this subspace. Since d_{TV} between two Gaussians is asymptotically equivalent to the Mahalanobis distance between them, we get the following corollary using d_{ENTRY}.

Corollary 4.9. *Let $\mathcal{N}(\mu, \boldsymbol{\Sigma})$ be a Gaussian with support in the range of linear transformation $\boldsymbol{A}$. For $\hat{\mu}$ such that $d_{ENTRY}(\mathcal{N}(\mu, \boldsymbol{\Sigma}), \mathcal{N}(\hat{\mu}, \boldsymbol{\Sigma})) \leq \alpha$, $\|\mu - \hat{\mu}\|_{\boldsymbol{\Sigma}} = O(\alpha \frac{n}{m_{\boldsymbol{A}}})$.*

Note that the upper bound above is under the condition that $\alpha < m_{\boldsymbol{A}}/(2n)$ when the corruption is limited to missing entries, and $\alpha < m_{\boldsymbol{A}}/(4n)$ when replacement is allowed. Otherwise, more than half of the samples can be corrupted and unrecoverable (the proof of Theorem 4.10 provides the conditions for recovery, and the break points for mean estimation follow). Corollary 4.9 shows that $\mathcal{A}_2^\rho$ (when $\rho = \alpha$) and $\mathcal{A}_3^\alpha$ can only shift structured distributions by $O(\alpha \frac{n}{m_{\boldsymbol{A}}})$. This result suggests that we can improve upon the previous $O(\alpha n)$ mean estimation guarantees. Furthermore, the following theorem proves that this upper bound is tight under $\mathcal{A}_3^\alpha$.

Theorem 4.10. *Let $\mathcal{N}(\mu, \boldsymbol{\Sigma})$ be a Gaussian with support in the range of linear transformation $\boldsymbol{A}$ and let $\mathcal{A}_3^\alpha$ adversarially corrupt the samples. Any algorithm that outputs a mean estimator $\hat{\mu}$ must satisfy $\|\mu - \hat{\mu}\|_{\boldsymbol{\Sigma}} = \Omega(\alpha \frac{n}{m_{\boldsymbol{A}}})$.*

While our analysis focuses on Gaussian distributions, our analysis framework generalizes to any class of distributions that admits an efficient robust mean estimator under sample-level corruption, e.g. distributions with bounded covariance. This generality stems from our general reduction scheme between coordinate-level and sample-level corruption.

4.8.5 Proofs

Proof of Proposition 2

Proof. Suppose that $\mathcal{A}_3$ affects at least one entry in a subset S of all samples. As at least one coordinate per sample is corrupted, S must be at most an α-fraction of all samples; since $\alpha \leq \epsilon/n$ the sample-level adversary can corrupt the entirety of every sample partially corrupted by the coordinate-level adversary, and thus, it is a stronger adversary given this condition. The proof for $\mathcal{A}_2$ is similar. $\square$

Proof of Proposition 3

Proof. If α, $\rho \geq \epsilon$, similarly to the proof of Proposition 2, $\mathcal{A}_2$ and $\mathcal{A}_3$ can simulate $\mathcal{A}_1$ by placing all its corruptions on the ϵN coordinates corrupted by $\mathcal{A}_1$. If $\alpha \geq \rho$, $\mathcal{A}_3$ can simulate $\mathcal{A}_2$ by corrupting the coordinates corrupted by $\mathcal{A}_2$ since $\mathcal{A}_2$ can never corrupt more than ρ-fraction of coordinates in expectation. On the other hand, if $\alpha \leq \rho/n$, $\mathcal{A}_2$ can corrupt whatever coordinates $\mathcal{A}_3$ decides to corrupt since $\mathcal{A}_3$ cannot corrupt more than αn-fraction of one coordinate. Thus, the three statements hold. □

Proof of Theorem 4.2

Proof. We first show that when $\alpha > 2\alpha'$, $\mathcal{A}_3^\alpha$ has a way to make corruptions so that with probability at least $1 - e^{-\Omega(\alpha^2 N)}$ it is indistinguishable whether the N samples come from D_1 or D_2. From the definition of d^1_{ENTRY}, if we take the coupling γ that achieves the infimum, changing α' fraction of the entries per sample on average will make D_1 indistinguishable from D_2. Therefore, if the adversary corrupts the entries of the N samples according to the coupling γ, by Hoeffding's inequality, the probability that more than $2\alpha'$ fraction of the entries need to be changed to make it impossible to tell whether the samples come from D_1 or D_2 is less than $e^{-\Omega(\alpha^2 N)}$.

Then we show that when $\alpha < \alpha'/4$, no matter how $\mathcal{A}_3^\alpha$ makes corruptions, with probability at least $1 - e^{-\Omega(\alpha^2 N)}$, we can tell that the N samples come from D_1. Since $d^1_{\text{ENTRY}}(D_1, D_2) = \alpha'$, by Monge-Kantorovich duality theorem (see e.g. Theorem 5.10 of this book (Villani, 2009)), there exists a function $u : (\mathbb{R} \cup \{\perp\})^n \to [0, 1]$, where $\perp$ denotes a missing entry, such that $u(x) - u(y) \leq \frac{1}{n}\|x - y\|_0$ and $\mathbb{E}_{D_1}[u(x)] - \mathbb{E}_{D_2}[u(x)] = \alpha'$. This is because the optimal coupling of D_1, D_2 for d^1_{ENTRY} is represents the optimal to the primal Kantorovich problem where $c(x, y) = \frac{1}{n}\|I(x, y)\|_1$, while u represents the optimal to the dual problem. We use u to distinguish whether the corrupted samples come from D_1 or D_2 by checking whether the expectation of u according to the empirical distribution $\hat{D}$ that we observe is closer to the expectation corresponding to D_1 or D_2. By Hoeffding's inequality, the empirical distribution $\hat{D}_1$ of the N samples before corruption satisfies $|\mathbb{E}_{D_1}[u(x)] - \mathbb{E}_{\hat{D}_1}[u(x)]| \le \alpha'/4$ with probability at least $1 - e^{-\Omega(\alpha^2 N)}$. After corruption, we have that $|\mathbb{E}_{\hat{D}}[u(x)] - \mathbb{E}_{\hat{D}_1}[u(x)]| \le \alpha$ by the bound on the number of corrupted entries and the Lipschitz property of u. Thus, with probability at least $1 - e^{-\Omega(\alpha^2 N)}$, $|\mathbb{E}_{D_1}[u(x)] - \mathbb{E}_{\hat{D}}[u(x)]| \le \alpha'/4 + \alpha < \alpha'/2$ while $|\mathbb{E}_{D_2}[u(x)] - \mathbb{E}_{\hat{D}}[u(x)]| > \alpha'/2$, which allows us to distinguish between D_1 and D_2.

In the case of the value-fraction adversary $\mathcal{A}_2^\rho$ that can corrupt ρ-fraction of values in each coordinate, d^∞_{ENTRY} can be bound similarly in $\Theta(\rho)$ by applying Hoeffding's inequality and Kantorovich duality theorem for each coordinate such that $u_i(x) - u_i(y) \le \|x_i - y_i\|_0$ and then comparing the mean for each coordinate. Therefore, for both $\mathcal{A}_2^\rho$ and $\mathcal{A}_3^\alpha$, d_{ENTRY} is a tight characterization of the coordinate-level adversary. $\square$

Proof of Theorem 4.4

Proof. Let $\text{disc}(\mathbf{\Sigma}^{-1}) = \max_{x \in [-1,1]} \sqrt{x^T s(\mathbf{\Sigma}^{-1}) x}$ and let v be the vector with entries $(\mathbf{\Sigma}^{-1}_{ii})^{-1/2}$. To complete the proof, we will show that $d^\infty_{\text{ENTRY}}(N(\mu, \mathbf{\Sigma}), N(\mu + \rho v, \mathbf{\Sigma})) \le \rho$. To do this, we are going to use a hybrid argument showing that by only hiding ρ fraction of the entries in the i-th coordinate, $N(\mu, \mathbf{\Sigma})$ and $N(\mu + \rho \mathbf{e}_i / \mathbf{\Sigma}_{ii}^{-1/2}, \mathbf{\Sigma}))$ become indistinguishable where $\mathbf{e}_i$ is the vector that has 1 in its i^{th} coordinate and 0 in the others. This is because, $d_{\text{TV}}(N(\mu, \mathbf{\Sigma}), N(\mu + \rho \mathbf{e}_i / \mathbf{\Sigma}_{ii}^{-1/2}, \mathbf{\Sigma}))) \le \rho$. By applying this argument sequentially for every coordinate, $N(\mu, \mathbf{\Sigma})$ and $N(\mu + \rho v, \mathbf{\Sigma})$ are indistinguishable under an $\mathcal{A}_2$ adversary. Since the total distance between μ and $\mu + \rho v$ is at least $\rho \cdot \text{disc}(\mathbf{\Sigma}^{-1})$, the theorem follows. $\square$

Proof of Corollary 4.5

Proof. We prove the following lemma that implies Corollary 4.5 when combined with Theorem 4.4.

Lemma 4.11. *For any $n \times n$ PSD matrix $\boldsymbol{M}$, disc($\boldsymbol{M}$) $\in [\sqrt{n}, n]$*

We have that $s(\boldsymbol{M})$ is a PSD matrix with diagonal elements equal to 1. Consider a random x with uniformly random coordinates in $\{-1, 1\}$. Then, $\mathbb{E}[x^T s(\boldsymbol{M})x] = \text{Trace}(s(\boldsymbol{M})) = n$. Thus, $\max_{x\in[-1,1]} \sqrt{x^T s(\boldsymbol{M})x} \geq \sqrt{n}$. This lower bound is tight for $\boldsymbol{M} = \boldsymbol{I}$.

For the upper-bound, we notice that since $s(\boldsymbol{M})$ is PSD, it holds that $|s(\boldsymbol{M})_{ij} + s(\boldsymbol{M})_{ji}| \leq 2$. To see this notice that $x^T s(\boldsymbol{M})x \geq 0$ for both $x = e_i + e_j$ and $x = e_i - e_j$.

Given this, we have that $x^T s(\boldsymbol{M})x \leq \frac{1}{2}\sum_{ij} |s(\boldsymbol{M})_{ij} + s(\boldsymbol{M})_{ji}| \leq n^2$. This gives the required upper-bound. Notice that the upper-bound is tight for the matrix $\boldsymbol{M}$ consisting entirely of 1's. □

Proof of Theorem 4.6

Proof. With a budget of α, $\mathcal{A}_3$ can concentrate its corruption on one particular coordinate, say the first coordinate. If $\alpha n \geq 1$, we will lose all information for the first coordinate, making mean estimation impossible. Since $\alpha < 1/n$, $\mathcal{A}_3$ can corrupt αn-fraction of first coordinates of all samples. Since the marginal distribution with respect to the first dimension is a univariate Gaussian, information-theoretically any mean estimator of the first coordinate must be $\Omega(\alpha n)$-far from the true mean of the first coordinate.

□

Proof of Theorem 4.8

Proof. First, we show the case for d^1_{ENTRY}.

$d^1_{\text{ENTRY}}(D_1, D_2) \leq d_{\text{TV}}(D_1, D_2)$ follows from

$$\begin{aligned} d^1_{\text{ENTRY}}(D_1, D_2) &= \inf_{\gamma \in \Gamma(D_1, D_2)} \frac{\| \mathbb{E}_{(x,y)\sim\gamma} [I(x, y)] \|_1}{n} \\ &= \inf_{\gamma \in \Gamma(D_1, D_2)} \mathop{\mathbb{E}}_{(x,y)\sim\gamma} \left[\frac{||x - y||_0}{n} \right] \\ &\leq \inf_{\gamma \in \Gamma(D_1, D_2)} \Pr_{(x,y)\sim\gamma} [x \neq y] \\ &= d_{\text{TV}}(D_1, D_2) \end{aligned}$$

Then we show that $d^1_{\text{ENTRY}}(D_1, D_2) \geq \frac{m_{\boldsymbol{A}}}{n} d_{\text{TV}}(D_1, D_2)$.

We first show that for any $x \notin \ker(\boldsymbol{A})$, $||\boldsymbol{A}x||_0 \geq m_{\boldsymbol{A}}$. Suppose by way of contradiction that $\Pi_i\, \boldsymbol{A}x$ is nonzero for fewer than $m_{\boldsymbol{A}}$ values of i. Call the rows of $\boldsymbol{A}$ $v_0^T, \ldots, v_{n-1}^T$ and let S be the subspace of $\mathbb{R}^r$ spanned by the v_i's. As $x \notin \ker(\boldsymbol{A})$, $\boldsymbol{A}x$ is nonzero. Hence, $\langle x, v_i \rangle$ is nonzero for some i so $\Pi_S\, x$ is nonzero.

Now, let $\mathcal{B}$ be a basis for S containing $\Pi_S\, x$. Consider the subspace S' of S spanned by $\{v_i \mid \langle x, v_i \rangle = 0\}$. As $\Pi_{S'}\, x = 0$, $\Pi_S\, x$ cannot be an element of S' and so $\mathcal{B}$ is not a basis for S'. Thus, the dimension of S' is less than that of S; as $|\{v_i\}| - |\{v_i \mid \langle x, v_i \rangle = 0\}| < m_{\boldsymbol{A}}$ we have a contradiction of the definition of $m_{\boldsymbol{A}}$. Thus, if $x \neq 0 \in \mathbb{R}^r$, $\Pi_i\, \boldsymbol{A}x$ must be nonzero for at least $m_{\boldsymbol{A}}$ values of i, and hence $||\boldsymbol{A}x||_0 \geq m_{\boldsymbol{A}}$.

Now, suppose that $(x, y) \sim \gamma$ for some $\gamma \in \Gamma(D_1, D_2)$. Then, $x = \boldsymbol{A}x'$ and $y = \boldsymbol{A}y'$ for some $x', y' \in \mathbb{R}^r$. If $x \neq y$, then $\boldsymbol{A}x' \neq \boldsymbol{A}y'$ so $x' - y' \notin \ker(A)$. Thus

$$||\boldsymbol{A}(x' - y')||_0 \geq m_{\boldsymbol{A}}$$

by the above, and so

$$\mathop{\mathbb{E}}_{(x,y)\sim\gamma} [||x - y||_0] \geq m_{\boldsymbol{A}} \Pr_{(x,y)\sim\gamma} [x \neq y]$$

Therefore, we have that

$$\begin{aligned} d^1_{\text{ENTRY}}(D_1, D_2) &= \inf_{\gamma\in\Gamma(D_1,D_2)} \frac{\| \mathbb{E}_{(x,y)\sim\gamma}\left[I(x,y)\right] \|_1}{n} \\ &= \inf_{\gamma\in\Gamma(D_1,D_2)} \mathop{\mathbb{E}}_{(x,y)\sim\gamma}\left[\frac{||x-y||_0}{n}\right] \\ &\geq \inf_{\gamma\in\Gamma(D_1,D_2)} \frac{m_{\boldsymbol{A}}}{n} \Pr_{(x,y)\sim\gamma}\left[x \neq y\right] \\ &= \frac{m_{\boldsymbol{A}}}{n}\, d_{\text{TV}}(D_1, D_2) \end{aligned}$$

In the case of d^∞_{ENTRY}, the left hand side ($d^\infty_{\text{ENTRY}}(D_1, D_2) \geq \frac{m_{\boldsymbol{A}}}{n}\, d_{\text{TV}}(D_1, D_2)$) follows from above by using the fact that $\|x\|_1 \leq n\|x\|_\infty$ for $x \in \mathbb{R}^n$. The right hand side follows from

$$\begin{aligned} d^\infty_{\text{ENTRY}}(D_1, D_2) &= \inf_{\gamma\in\Gamma(D_1,D_2)} \| \mathop{\mathbb{E}}_{(x,y)\sim\gamma}\left[I(x,y)\right] \|_\infty \\ &= \inf_{\gamma\in\Gamma(D_1,D_2)} \max_i \Pr_{(x,y)\sim\gamma}\left[x_i \neq y_i\right] \\ &\leq \inf_{\gamma\in\Gamma(D_1,D_2)} \Pr_{(x,y)\sim\gamma}\left[x \neq y\right] \\ &= d_{\text{TV}}(D_1, D_2) \end{aligned}$$

Therefore, the theorem holds for the d_{ENTRY} metric. □

Proof of Corollary 4.9

Proof. We can obtain the given upper bound relating the distance to d_{TV}. Since $d_{\text{TV}}(\mathcal{N}(0,1), \mathcal{N}(\mu,1)) = erf(\frac{\mu}{2\sqrt{2}})$, for small $\mu > 0$, $erf(\frac{\mu}{2\sqrt{2}}) = \Theta(\mu)$. Then

$$\begin{aligned} d_{\text{TV}}(\mathcal{N}(\mu, \boldsymbol{\Sigma}), \mathcal{N}(\mu', \boldsymbol{\Sigma})) &= d_{\text{TV}}(\mathcal{N}(0, I), \mathcal{N}(\boldsymbol{\Sigma}^{-1/2}(\mu' - \mu), I)) \\ &= d_{\text{TV}}(\mathcal{N}(0,1), \mathcal{N}(\|\boldsymbol{\Sigma}^{-1/2}(\mu' - \mu)\|_2, 1)) \\ &= d_{\text{TV}}(\mathcal{N}(0,1), \mathcal{N}(\|\mu' - \mu\|_{\boldsymbol{\Sigma}}, 1)) = \Theta(\|\mu' - \mu\|_{\boldsymbol{\Sigma}}) \end{aligned}$$

Applying Theorem 4.8, we get that $\|\mu - \mu'\|_{\boldsymbol{\Sigma}} = O(\alpha\frac{n}{m_{\boldsymbol{A}}})$. □

Proof of Theorem 4.10

Proof. We prove the theorem for both missing values and replaced values. In the case of missing values, for the lower bound, $\mathcal{A}_3^\alpha$ may corrupt at most $\frac{\alpha n}{m_{\boldsymbol{A}}}$-fraction of the samples so that the coordinates are non-recoverable and shift part of the original distribution to anywhere along the axes of missing coordinates. Then the proof similarly follows the lower bound proof for estimating the mean of a Gaussian corrupted by $\mathcal{A}_1^\epsilon$. Hence, since we cannot distinguish between two Gaussians that share $1 - \frac{\alpha n}{m_{\boldsymbol{A}}}$ of mass, $\|\hat{\mu} - \mu\|_{\boldsymbol{\Sigma}} = \Omega(\alpha\frac{n}{m_{\boldsymbol{A}}})$.

For $\mathcal{A}_3^\alpha$ that replaces values, we prove the following lemma and the theorem follows.

Lemma 4.12. *The adversary corrupts δ coordinates of a sample. Let $\tilde{x}$ be the corrupted sample and $x^* = \boldsymbol{A}z^*$ be the original. We can only information-theoretically recover x^* from $\tilde{x}$ if and only if $\delta < \frac{m_{\boldsymbol{A}}}{2}$. Furthermore, if $\delta < \frac{m_{\boldsymbol{A}}}{2}$ then $\|\tilde{x} - \boldsymbol{A}z^*\|_0 < \delta$ and $\|\tilde{x} - \boldsymbol{A}z'\|_0 \geq m_{\boldsymbol{A}} - \delta$ for any $z' \neq z^*$.*

Assume that if $\delta < \frac{m_{\boldsymbol{A}}}{2}$ then $\|\tilde{x} - \boldsymbol{A}z^*\|_0 < \delta$ and $\|\tilde{x} - \boldsymbol{A}z'\|_0 \geq m_{\boldsymbol{A}} - \delta$ for any $z' \neq z^*$. This implies that we can consider all possible subsets $I \subseteq \mathcal{U}$ where $|I| = n - \frac{m_{\boldsymbol{A}}}{2}$ and solve the linear system of equations of $\tilde{x}_I = \boldsymbol{A}_I z$ and output the solution z, which achieves smallest hamming distance to $\tilde{x}$, as z^*. If $\delta \geq \frac{m_{\boldsymbol{A}}}{2}$, it is information theoretically impossible to recover x^* as $\arg\min_z \|\tilde{x} - \boldsymbol{A}z\|_0$ may not be unique: since the corruptions are adversarial, z^* may not be part of the set of minimizers.

Assume $\delta < \frac{m_{\boldsymbol{A}}}{2}$. Without loss of generality, let $\boldsymbol{A}$ be full rank. If not, the proof follows by replacing r with $\text{rank}(\boldsymbol{A})$ and considering the kernel of $\boldsymbol{A}$. Let A_i denote the i-th row of matrix $\boldsymbol{A}$ and $\mathcal{U} = \{A_i : i \in [n]\}$. Let Δ denote the set of A_i's that correspond to the corrupted coordinates of $\tilde{x}$ so that $|\Delta| = \delta$. Define $\mathcal{S} \supseteq \Delta$ to be the smallest subset of $\mathcal{U}$ such that row space dimension (rank) of $\boldsymbol{A}_{\mathcal{U}\setminus\mathcal{S}}$ is 1 less than that of $\boldsymbol{A}$. By definition of $m_{\boldsymbol{A}}$, $|\mathcal{S}| \geq m_{\boldsymbol{A}}$.

The entries corresponding to rows $\mathcal{U} \setminus \mathcal{S}$ are uncorrupted, so if we solve the linear system $\boldsymbol{A}_{\mathcal{U}\setminus\mathcal{S}} z = \tilde{x}_{\mathcal{U}\setminus\mathcal{S}}$, we will a get a 1-dimensional solution space for z. Thus, any z in this line will give at least $|\mathcal{U} \setminus \mathcal{S}|$ matching coordinates when multipied to $\boldsymbol{A}$ with x^*. Now, we can generate $|\mathcal{S}|$ many solutions, each corresponding to the solution to the linear system $\boldsymbol{A}_{\mathcal{U}\setminus\mathcal{S}\cup\{s\}} z = \tilde{x}_{\mathcal{U}\setminus\mathcal{S}\cup\{s\}}$ for each $s \in \mathcal{S}$.

For s that corresponds to an uncorrupted entry in $\tilde{x}$, the solution to the linear system is the true solution z^* since none of the values in the system was corrupted. That gives us at least $|S| - \delta$ solutions out of $|S|$ solutions to be exactly z^*. Regardless of how the adversary corrupts the δ entries, if $\delta < \frac{m_{\boldsymbol{A}}}{2}$, then the majority solution will always be z^* since $|S| - \delta > \frac{m_{\boldsymbol{A}}}{2} > \delta$. Furthermore, for $z' \neq z^*$, z' can match at most $|\mathcal{U} \setminus \mathcal{S}| + \delta \leq n - m_{\boldsymbol{A}} + \delta$ coordinates of $\tilde{x}$, i.e. $\|\boldsymbol{A}z' - \tilde{x}\|_0 \geq m_{\boldsymbol{A}} - \delta$. However, if $\delta \geq \frac{m_{\boldsymbol{A}}}{2}$, then there is no clear majority so it is impossible to distinguish between the true solution and the other solution. In fact, when δ is strictly greater and corruptions adversarially chosen, $\|\boldsymbol{A}z^* - \tilde{x}\|_0 = \delta$ and there exists some z', $\|\boldsymbol{A}z' - \tilde{x}\|_0 = m_{\boldsymbol{A}} - \delta < \|\boldsymbol{A}z^* - \tilde{x}\|_0$.

□

4.9 Conclusion

We introduced Picket, a first-of-its-kind system that safeguards against data corruptions for machine learning pipelines over tabular data either during training or deployment. To design Picket, we introduced PicketNet, a novel self-supervised deep learning model that corresponds to a Transformer network for tabular data. Picket is designed as a plugin that can increase the robustness of any machine learning pipeline.

In addition, we studied the problem of robust mean estimation under coordinate-level corruption. We proposed d_{ENTRY}, a new measure of distribution shift for coordinate-level corruptions and introduced adversary models that capture more realistic corruptions than prior works. We presented an information-theoretic analysis of robust mean estimation for these adversaries and showed that when the data exhibits redundancy one should first fix corrupted samples before estimation.

5 TASK-SPECIFIC DATA SELECTION FOR FINETUNING MACHINE LEARNING MODELS

5.1 Introduction

Finetuning foundation models (Bommasani et al., 2021) has become a popular paradigm for building task-specific machine learning applications. Foundation models such as BERT (Devlin et al., 2019) and LLaMA (Touvron et al., 2023) are large-scale models pretrained on massive unlabeled data across a wide range of domains. Those models can be specialized to specific downstream tasks through supervised or unsupervised finetuning using task-specific datasets. Supervised finetuning further trains a foundation model on a dataset equipped with labels specific to the target task.

Example 5.1. *A practitioner wants to build a model to extract chemical-protein relations from biomedical paragraphs. The practitioner first collects and annotates a set of 4000 paragraphs. Then instead of training a model from scratch, the practitioner takes a pretrained BERT model and finetunes its weights on the annotated paragraphs in a supervised way.*

Unsupervised finetuning (also called continued pretraining) takes an unlabeled dataset that is relevant to the target task and trains a foundation model on it in an unsupervised way similar to pretraining. The goal of continued pretraining is to tailor the model to a specific domain which is the focus of the target task.

Example 5.2. *Consider a practitioner with the same purpose as in Example 5.1. This time the practitioner has an extra set of 2.7 million biomedical papers crawled from the web but does not have enough budget to label them. To fully utilize the unlabeled data, they continue pretraining the BERT model on the unlabeled texts and then finetunes it on the annotated dataset.*

Continued pretraining and supervised finetuning can lead to significant improvement in downstream tasks, but its effectiveness heavily relies on the right choice of pretraining data. For example, as is shown by Gururangan et al. (2020), continued pretraining of RoBERTa (Liu et al., 2019) on 2.7 million biomedical papers before finetuning improves its performance on chemical-protein relation extraction by over 2 points in F1 score. On the other hand, continued pretraining on the same amount of texts from irrelevant domains leads to a drop of over 2 points. In practice, while web-crawled cross-domain data pools such as Common Crawl[1] and The Pile (Gao et al., 2020)) are easily accessible, the vast majority of the data are irrelevant to the target task. Given the volume of those candidate pools (e.g., Common Crawl contains 250 billion pages), it is impractical to manually select the data that is distributed like the use cases in the target task. Therefore, automated task-specific data selection becomes critical.

In this chapter, we seek a framework to select task-specific data for ML model finetuning. Given a small set of representative examples from a target task, our goal is to select a pretraining set from a massive corpus.

Many works have focused on task-specific data selection but they fall short in several key characteristics. Methods relying on heuristics such as similarity search (Ruder and Plank, 2017b; Gururangan et al., 2020; Aharoni and Goldberg, 2020; Yao et al., 2022; Xia et al., 2024) or binary classification (Aharoni and Goldberg, 2020) do not ensure that the selected data is distributed like the data in the target task. DSIR (Xie et al., 2023) selects text data to match a target distribution in an n-gram feature space, which cannot capture high-level semantics. Furthermore, existing methods are not robust to near-duplicates in the candidate pool. They treat near-duplicates as distinct examples, and hence examples with near-duplicates will be overrepresented in the selected set. A principled and holistic framework for task-specific data selection is missing. Such a framework should satisfy

[1]`https://commoncrawl.org/`

the following requirements.

(*Distribution Alignment*) First, the distribution of the selected data should match the distribution of the target task. Distribution alignment is essential for a model to learn the target distribution and enable sample-efficient finetuning for the target task (Shachaf et al., 2021). The framework should be generic to support distribution alignment in any feature space.

(*Diversity*) Second, the selected data should be diverse so that the model can learn a wide range of domain knowledge rather than overfitting specific examples. In practice, a candidate pool created by web crawling and the integration of multiple sources may contain a large portion of near-duplicates (Fröbe et al., 2021; Lee et al., 2022). For example, the study by Fröbe et al. (2021) on several snapshots of ClueWeb[2] and Common Crawl shows that 14% to 52% of the documents are near-duplicates. Including near-duplicates into the training set unconsciously can compromise diversity and negatively impact model performance (Lee et al., 2022; Hernandez et al., 2022). Therefore, a mechanism to deal with near-duplicates is essential.

(*Scalability*) Finally, the selection algorithm should be efficient, considering the increasing scale of modern data pool. For instance, OpenWeb-Text2 (Gao et al., 2020) contains 69 million documents, and Common Crawl contains 250 billion pages.

To this end, we present a framework to select task-specific data for finetuning that meets the aforementioned requirements. Our framework approaches data selection as an optimization problem that allows a smooth trade-off between distribution alignment and diversity. The optimization problem also admits efficient algorithms to compute the optimal solution. Our technical contributions are as follows.

1. We formulate task-specific data selection as an optimization problem based on optimal transport, assigning probabilities of being selected to the candidate pool. The objective function consists of a distribution

[2]`https://lemurproject.org/`

alignment loss, and a regularization term to encourage diversity among the selected data. The distribution alignment loss quantifies the difference between the distribution assigned to the candidates and the empirical distribution of the target-task representatives. We use the probability transportation cost between the two distributions as the distribution alignment loss, following the formula of optimal transport, which is flexible to be defined in any metric space and can effectively measure the difference between distributions with non-overlapping supports. In addition, optimal transport is closely related to generalization error (Rodríguez Gálvez et al., 2021).

2. We make our framework robust to near-duplicates in the corpus by incorporating kernel density estimation (Parzen, 1962) into the regularization term. By discounting the probability mass assigned to points with near-duplicates, our framework maintains consistent performance when the level of duplication varies.

3. We show the connection between the optimal solution to the optimization problem and nearest neighbor search. This connection allows us to develop efficient algorithms employing approximate nearest-neighbor search techniques (Douze et al., 2024).

We conduct extensive experiments to validate the effectiveness of our framework. We focus on natural language processing tasks where foundation models have shown great advancements. For eight text classification tasks from diverse domains, we continue pretraining transformer-based foundation models on the selected data before finetuning them on the domain-specific tasks. The results show that our method improves the average F1 score by 2.6 points compared to the base model, and outperforms all the competing methods in 11 out of the 16 settings. We also show that when we duplicate 1% of the examples in the corpus for 1000 times, our method maintains consistent performance, while the competing methods suffer a drop of at least 2.5 points in F1 scores on average across datasets. In addition, for task-specific instruction tuning, our framework beats the state-of-the-art method (Xia et al., 2024) by up to 5 points in F1 scores. Our method is efficient, taking 28 hours to preprocess a corpus of 150M examples and less than 1 hour for each task-specific selection. In summary, our framework is effective in improving the performance of task-specific finetuning and is practical in large-scale settings.

5.2 Preliminaries

We use δ_x to denote the Dirac measure centered at x. For a set $\mathcal{A}$ we have

$$\delta_x(\mathcal{A}) = \begin{cases} 1, & \text{if } x \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases}$$

Let $(\mathcal{A}, f)$ be a metric space where $\mathcal{A}$ is a set and $f : \mathcal{A} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ is a distance function. Consider two discrete distributions μ on $\mathcal{U} \subseteq \mathcal{A}$ and ν on $\mathcal{V} \subseteq \mathcal{A}$, where both $\mathcal{U}$ and $\mathcal{V}$ are finite sets. Let u_i be the i^{th} example in $\mathcal{U}$ and $\mu = \sum_{i=1}^{|\mathcal{U}|} \mu_i \delta_{u_i}$ where μ_i is the probability mass assigned to u_i. Similarly, let $\nu = \sum_{j=1}^{|\mathcal{V}|} \nu_j \delta_{v_i}$, where ν_j is the probability mass assigned to v_j, the j^{th} example in $\mathcal{V}$. Let $\boldsymbol{\gamma} \in \mathbb{R}_{\geq 0}^{|\mathcal{U}| \times |\mathcal{V}|}$ be a transport of probability mass between μ and ν, where γ_{ij} is amount of probability mass transported from u_i to v_j. Assume that the cost of transporting one unit of probability mass from u_i to v_j is $f(u_i, v_j)$, the distance between u_i and v_j. The Wasserstein (a.k.a. optimal transport) distance between μ and ν is the minimal cost of transporting all the probability mass from $\mathcal{U}$ to $\mathcal{V}$:

$$\begin{aligned} W(\mu, \nu) = \min_{\gamma \in \mathbb{R}_{\geq 0}^{|\mathcal{U}| \times |\mathcal{V}|}} & \sum_{i=1}^{|\mathcal{U}|} \sum_{j=1}^{|\mathcal{V}|} \gamma_{ij} f(u_i, v_j) \\ \text{subject to} \quad & \sum_{j=1}^{|\mathcal{V}|} \gamma_{ij} = \mu_i, \forall i \in 1, 2, \ldots, |\mathcal{U}| \\ & \sum_{i=1}^{|\mathcal{U}|} \gamma_{ij} = \nu_j, \forall j \in 1, 2, \ldots, |\mathcal{V}| \end{aligned}$$

5.3 Our Framework

We formally state the problem of task-specific data selection and provide an overview of our framework.

Problem Statement We assume access to a set of M representatives $\mathcal{Q} = \{q_i\}_{i=1}^{M}$ from the target task, which we call query examples. Consider a candidate pool $\mathcal{D} = \{x_j\}_{j=1}^{N}$ containing N examples from a wide range of domains. Note that $\mathcal{Q}$ and $\mathcal{D}$ are multisets that may contain duplicates. We aim to select B task-specific examples from the candidate pool guided by the query examples. The selected examples will be used to finetune a model to tailor it to the target task.

Framework Overview Our framework selects task-specific examples by probability sampling following a categorical distribution over the candidate pool. The categorical distribution is determined by an optimization problem. Specifically, our framework takes the candidate pool and the query examples as inputs and outputs a set of task-specific examples. The workflow of our framework is as follows.

- (*Encoding*) We first encode the query examples and the examples in the candidate pool. The distances between the examples will be computed based on the encodings.

- (*Probability Assignment*) We determine the probability mass assigned to each example in the candidate pool by solving an optimization problem. The optimization problem transports probability mass from the query examples to the examples in the candidate pool. The optimization objective is the transportation cost as in the Wasserstein distance, plus a regularizer that encourages the diversity of transportation.
- (*Sampling*) We take a random sample with replacement from the candidate pool following a categorical distribution where the probability is determined by the assignment in the previous step.

5.4 Optimization Problem

In this section, we introduce the proposed optimization problem for probability assignment that admits different instantiations of the regularization term. We show the existence of closed-form solutions for two instantiations. In addition, we propose a regularization term that addresses the problem of near-duplicates in the candidate pool with a closed-form solution.

5.4.1 Optimization Objective

Consider the metric space $(\mathcal{Z}, f)$ where $\mathcal{Z} = \mathcal{Q} \cup \mathcal{D}$ contains all the examples and $f : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ is a distance function. Let $\boldsymbol{d} \in \mathbb{R}_{\geq 0}^{M \times N}$ be the distance matrix, where $d_{ij} = f(q_i, x_j)$ is the distance between the i^{th} query example and the j^{th} example in the candidate pool.

Given $\boldsymbol{d} \in \mathbb{R}_{\geq 0}^{M \times N}$, we consider the following optimization problem, which we refer to as Problem 5.1.

$$\begin{aligned} \min_{\gamma \in \mathbb{R}_{\geq 0}^{M \times N}} & \frac{\alpha}{C} \sum_{i=1}^{M} \sum_{j=1}^{N} \gamma_{ij} d_{ij} + (1-\alpha) G(\boldsymbol{\gamma}) \\ \text{subject to} \quad & \sum_{j=1}^{N} \gamma_{ij} = \frac{1}{M}, \forall i \in [M] \end{aligned} \tag{5.1}$$

where $C > 0$ is a scaling constant, $\alpha \in [0, 1]$ is a hyper-parameter, and G is a function for regularization.

The first term in Problem 5.1 is the cost of probability transport from the empirical distribution of $\mathcal{Q}$ and the categorical distribution we assign to the candidate pool, which characterizes the alignment of the two distributions. The second is a regularization term that encourages the diversity of probability assignment. We will introduce several instantiations of the regularization term later. The hyperparameter α allows smooth trade-off between distribution alignment and diversity. We assign $p_j^* = \sum_{i=1}^{M} \gamma_{ij}^*$ probability to x_j, where $\boldsymbol{\gamma}^*$ is an optimal solution to Problem 5.1.

We propose the following instantiations of the regularizer to quantify the diversity of probability assignment by comparing it to a uniform assignment.

- $G_\infty(\boldsymbol{\gamma}) = M \max_{i \in M, j \in N} |\gamma_{ij} - \frac{1}{MN}|$ captures the largest probability gap between $\boldsymbol{\gamma}$ and the uniform distribution.

- $G_{TV}(\boldsymbol{\gamma}) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} |\gamma_{ij} - \frac{1}{MN}|$ is the total variation distance between $\boldsymbol{\gamma}$ and the uniform distribution.

5.4.2 Closed-Form Solutions

When $G = G_\infty$ or $G = G_{TV}$, Problem 5.1 can be solved by standard linear programming techniques, but they run in $\Omega((MN)^2)$ time, which is prohibitively expensive. Instead, we show the existence of closed-form solutions that can be computed efficiently (see Section 5.5 for the algorithms).

When $G = G_\infty$, we get an optimal solution by transporting the probability of each query example evenly to its K-nearest neighbors in the candidate pool, where K is determined by the condition stated in the following theorem.

Theorem 5.3. *Given $\boldsymbol{d} \in \mathbb{R}_{\geq 0}^{M \times N}$ where $N > 1$, consider Problem 5.1 with $G(\boldsymbol{\gamma}) = G_\infty(\boldsymbol{\gamma}) = M \max_{i \in M, j \in N} |\gamma_{ij} - \frac{1}{MN}|$. For all $i \in [M]$, let $j_1^i, \ldots, j_N^i$ be a reordering of $[N]$ such that $d_{ij_1^i} \leq \cdots \leq d_{ij_N^i}$. Consider $\boldsymbol{\gamma}^* \in \mathbb{R}_{\geq 0}^{M \times N}$*

whose entries are

$$\gamma^*_{ij} = \begin{cases} \frac{1}{KM}, & \textit{if } j \in \{j^i_1, \dots, j^i_K\} \\ 0, & \textit{otherwise} \end{cases}$$

where $K = \max\{k \in [N] | \frac{\alpha}{C} \sum_{i=1}^M \sum_{l=1}^{k-1} (d_{ij^i_k} - d_{ij^i_l}) < (1-\alpha)M\}$. *Assume* $K \leq N/2$*, and then* $\boldsymbol{\gamma}^*$ *is a minimizer of Problem 5.1.* $\boldsymbol{\gamma}^*$ *is the unique minimizer if* $\frac{\alpha}{C} \sum_{i=1}^M \sum_{l=1}^{K} (d_{ij^i_{K+1}} - d_{ij^i_l}) > (1-\alpha)M$ *and* $\nexists i \in [M]$ *such that* $d_{ij^i_{K+1}} = d_{ij^i_K}$.

When $G = G_{TV}$, for each query example, we transport $\frac{1}{MN}$ probability mass to any example in the candidate pool whose distance to the query example is less than $\frac{(1-\alpha)C}{2\alpha}$ plus the distance between the query example and its 1-nearest neighbor. Then we transport all the remaining probability mass to the 1-nearest neighbor of each query example.

Theorem 5.4. *Given* $\boldsymbol{d} \in \mathbb{R}^{M \times N}_{\geq 0}$ *where* $N > 1$*, consider Problem 5.1 with* $G(\boldsymbol{\gamma}) = G_{TV}(\boldsymbol{\gamma}) = \frac{1}{2} \sum_{i=1}^M \sum_{j=1}^N |\gamma_{ij} - \frac{1}{MN}|$. *For all* $i \in [M]$*, let* $j^i_1, \dots, j^i_N$ *be a reordering of* $[N]$ *such that* $d_{ij^i_1} \leq \dots \leq d_{ij^i_N}$. *Consider* $\boldsymbol{\gamma}^* \in \mathbb{R}^{M \times N}_{\geq 0}$ *where* $\forall i \in [M]$

$$\forall k \in \{2, \dots, N\}, \gamma^*_{ij^i_k} = \begin{cases} \frac{1}{MN}, & \textit{if } d_{ij^i_k} - d_{ij^i_1} < \frac{(1-\alpha)C}{\alpha} \\ 0, & \textit{otherwise} \end{cases}$$

and

$$\gamma^*_{ij^i_1} = \frac{1}{M} - \sum_{k=2}^N \gamma^*_{ij^i_k}$$

Then $\boldsymbol{\gamma}^*$ *is a minimizer of Problem 5.1.* $\boldsymbol{\gamma}^*$ *is the unique minimizer if* $\forall i \in [M] \forall k \in [N]$, $d_{ij^i_k} - d_{ij^i_1} \neq \frac{(1-\alpha)C}{\alpha}$ *and* $d_{ij^i_1} \neq d_{ij^i_2}$.

5.4.3 Addressing Near-Duplicates via Kernel Density Estimation

When there exists a large fraction of near-duplicates in the candidate pool, G_∞ and G_{TV} fail to characterize the diversity of probability assignment since they treat near-duplicates as distinct examples. Consequently, when we use G_∞ or G_{TV} as the regularization function, the contents in the near-duplicates will be over-sampled. For example, consider the case where we assign probability mass to the candidate pool following the optimal solution described in theorem 5.3. If 100 of the K^*-nearest neighbors of a query example are duplicates and the others are distinct, the content in the duplicates will receive 100 times as much probability mass as any others.

To address the near-duplicate problem, we propose a regularization function incorporating kernel density estimation (KDE) (Parzen, 1962). KDE is a non-parametric method to estimate the probability density function from finite examples. We use the Epanechnikov kernel such that given the candidate pool $\mathcal{D}$, the density estimate at point x is $\sum_{x' \in \mathcal{D}} \max(1 - \frac{f(x,x')^2}{h^2}, 0)$, where $h > 0$ is the kernel size and f is the distance function. The intuition is that a concentration of points around a position indicates a high probability density at that position and the potential existence of near-duplicates. For example, assume a point x in the candidate pool whose distance to any other point is larger than h, then the density estimate at x would be 1. If we create two duplicates of x and add them to the candidate pool, the density estimate at x will be increased to 3.

Our KDE-based regularization function is

$$G_{\text{KDE}}(\boldsymbol{\gamma}) = M \max_{i \in [M], j \in [N]} \rho_j |\gamma_{ij} - \frac{1/\rho_j}{M \sum_{j' \in [N]} 1/\rho_{j'}}|$$

where $\rho_j = \sum_{x' \in D} (1 - \frac{f(x_j, x')^2}{h^2})$ is the density estimate at x_j. $G_{\text{KDE}}(\boldsymbol{\gamma})$ compares $\boldsymbol{\gamma}$ to the probability assignment that is proportional to the inverse of the density estimate, and penalizes the largest gap weighted by the density estimate. $G_{\text{KDE}}(\boldsymbol{\gamma})$ is a generalization of G_∞ and it degenerates to G_∞ when $\rho_j = 1$ for all $j \in [N]$.

The optimal solution to Problem 5.1 when $G = G_{\text{KDE}}$ assigns probability mass to the nearest neighbors of each query point, weighted by the inverse of the density estimate, as is shown by the following theorem.

Theorem 5.5. *Given $\boldsymbol{d} \in \mathbb{R}_{\geq 0}^{M\times N}$ and $\rho_1, \dots, \rho_N \in \mathbb{R}_{>0}$, consider Problem 5.1 with $G(\boldsymbol{\gamma}) = G_{KDE}(\boldsymbol{\gamma}) = M \max_{i\in[M], j\in[N]} \rho_j |\gamma_{ij} - \frac{1/\rho_j}{M\sum_{j'\in[N]} 1/\rho_{j'}}|$. For all $i \in [M]$, let $j_1^i, \dots, j_N^i$ be a reordering of $[N]$ such that $d_{ij_1^i} \leq \cdots \leq d_{ij_N^i}$. Let $s_k^i = \sum_{l=1}^k 1/\rho_{j_l^i}$, and s be a discrete variable that takes value from $\mathcal{S} = \{s_k^i | i \in [M], k \in [N]\} \cup \{0\}$. Let $c(s) = \sum_{i=1}^M c_i(s)$, where*

$$c_i(s) = \begin{cases} 0, & \text{if } s \leq s_1^i \\ \\ \sum_{l=1}^{k-1} \frac{d_{ij_k^i} - d_{ij_l^i}}{\rho_{j_l^i}}, & \text{if } s_{k-1}^i < s \leq s_k^i (k \geq 2) \end{cases}$$

Let $s^* = \max\{s \in \mathcal{S} | \frac{\alpha}{C} c(s) < (1-\alpha)M\}$, and $K_i = \max\{k \in \{0, \dots, N-1\} | s_k^i \leq s^*\}$. Assume $s^* \leq \frac{1}{2}\sum_{j=1}^N 1/\rho_j$, and then $\boldsymbol{\gamma}^*$ is a minimizer of Problem 5.1 where $\forall i \in [M], k \in [N]$

$$\gamma^*_{ij_k^i} = \begin{cases} 1/(Ms^* \cdot \rho_{j_k^i}), & \text{if } k \leq K_i \\ \frac{1}{M} - \sum_{l=1}^{K_j} 1/(Ms^* \cdot \rho_{j_l^i}), & \text{if } k = K_i + 1 \\ 0, & \text{otherwise} \end{cases}$$

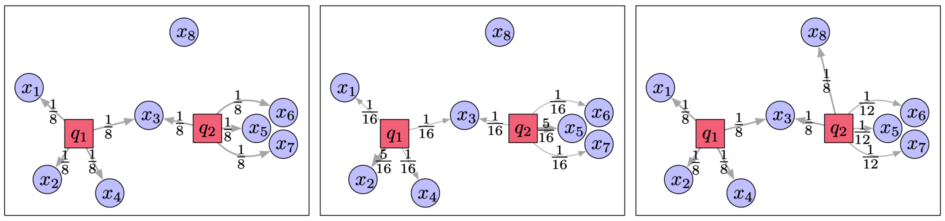
$\boldsymbol{\gamma}^*$ is the unique minimizer if $\nexists s \in \mathcal{S}$ such that $\frac{\alpha}{C} c(s) = (1-\alpha)M$ and $\nexists i \in [M]$ such that $d_{ij_{K_i}^i} = d_{ij_{K_i+1}^i}$ or $d_{ij_{K_i+1}^i} = d_{ij_{K_i+2}^i}$.

Intuitively, we count x_j in the candidate pool as $1/\rho_j$ examples. For each query example, the optimal solution i the theorem above assigns probability mass to the examples in its neighborhood proportional to their adjusted counts. The size of the neighborhood is determined by the limit s^* on the sum of the adjusted counts.

We show an example comparing the three instantiations in Figure 5.1, where we transport probability mass from two query examples to eight examples in the candidate pool following the optimal solutions given by Theorem 5.3, 5.4 and 5.5. In case 5.1a where $G = G_\infty$, assume $K = 4$ in Theorem 5.3 and each query example assigns its probability mass evenly to the 4-nearest neighbors. Note that x_3 receives probability mass from both query examples since it is in the neighborhood of both. In case 5.1b where $G = G_{\text{TV}}$, assume that x_1, x_2, x_3, x_4 receives non-zero mass from q_1, and x_3, x_5, x_6, x_7 receives non-zero mass from q_2. Each query example assigns $\frac{1}{MN} = \frac{1}{16}$ probability mass to its neighbors, and any remaining mass to the 1-nearest neighbor. Therefore, the 1-nearest neighbor receives more mass than the other neighbors. In case 5.1c where $G = G_{\text{KDE}}$, assume $s^* = 4$ in Theorem 5.5. In addition, assume that x_5, x_6, x_7 form a cluster where each of them has a kernel density estimate of $\frac{3}{2}$, and any other example has a kernel density estimate of 1. Similar to case 5.1a, q_1 assigns $\frac{1}{8}$ mass to its 4-nearest neighbors whose density estimate are 1. The neighborhood size of q_2 is based on the adjusted counts of the examples. Since x_5, x_6, x_7 have a density estimate of $\frac{3}{2}$, each of them is discounted as $\frac{2}{3}$ examples. Therefore, for $s^* = 4$, q_2 include x_3, x_5, x_6, x_7, x_8 in its neighborhood so that the sum of the adjusted count is 4. Then the probability mass is assigned to proportional to the adjusted counts. Compared to case 5.1a, the cluster (x_5, x_6, x_7) receives less mass due to their high similarity to each other.

5.5 Probability Assignment Algorithms

We propose efficient algorithms to assign probability mass to the candidate pool according to the optimal solutions to Problem 5.1. For the three instantiations of the regularization term G_∞, G_{TV}, G_{KDE}, the corresponding



(a) G_∞ (b) G_{TV} (c) G_{KDE}

Figure 5.1: Examples of the optimal probability transportation under different regularization terms. Assume two query examples (q_1 and q_2) and eight candidate pool examples ($x_1, \ldots, x_8$) embedded in a 2-dimensional space.

algorithms are KNN-Uniform (Algorithm 8), KNN-TV (Algorithm 9) and KNN-KDE (Algorithm 10). Each algorithm takes the query examples and the candidate pool as inputs and outputs the probability assigned to each example in the candidate pool.

Algorithm 8: KNN-Uniform.

```
1  Input: a set of query examples $\mathcal{Q} = \{q_i\}_{i=1}^M$ from the target domain,
     a corpus $\mathcal{D} = \{x_j\}_{j=1}^N$, number of nearest neighbors to prefetch $L$,
     $\alpha \in [0,1]$, $C > 0$;
2  Output: $p_1, \ldots, p_N$;
3  $\boldsymbol{j}, \boldsymbol{d} \leftarrow \textsc{GetKNN}(\mathcal{Q}, \mathcal{D}, L)$;
4  $K \leftarrow 1$;
5  while $K < L$ and $\frac{\alpha}{C} \sum_{i=1}^M \sum_{k=1}^K [d_{i,K+1} - d_{ik}] < (1-\alpha)M$ do
6      $K \leftarrow K + 1$;
7  foreach $j \in [N]$ do
8      $p_j \leftarrow 0$;
9  foreach $i \in [M]$ do
10     foreach $k \in [K]$ do
11         $p_{j_{ik}} \leftarrow p_{j_{ik}} + \frac{1}{KM}$;
```

All three algorithms prefetch the nearest neighbors of the query examples from the candidate pool as the first step, which can be performed efficiently using approximate nearest neighbor search techniques. Specifically, for each example in $\mathcal{Q}$, $\textsc{GetKNN}(\mathcal{Q}, \mathcal{D}, L)$ returns the indices of the L-nearest neighbors and the corresponding distances, where L is a user-specified parameter. In practice, we set L to be large enough so that the algorithms do not exceed the scope of the L-nearest neighbors. Retrieving the nearest neighbors in a brute-force way requires computing the distance between every query example and all the examples in the candidate pool, which is inefficient when the candidate pool size N is in the order of millions and billions. Therefore, we employ Faiss (Douze et al., 2024), an approximate nearest neighbor search tool for efficiency. We first build a Faiss index for the examples in the candidate pool, and then query the index with each query example to retrieve its nearest neighbors.

Algorithm 9: KNN-TV.

```
1  **Input:** a set of query examples $\mathcal{Q} = \{q_i\}_{i=1}^{M}$ from the target domain,
     a corpus $\mathcal{D} = \{x_j\}_{j=1}^{N}$, number of nearest neighbors to prefetch $L$,
     $\alpha \in [0, 1]$, $C > 0$;
2  **Output:** $p_1, \ldots, p_N$;
3  $\boldsymbol{j}, \boldsymbol{d} \leftarrow \textsc{GetKNN}(\mathcal{Q}, \mathcal{D}, L)$;
4  for $j \in [N]$ do
5      $p_j \leftarrow 0$;
6  for $i \in [M]$ do
7      $p_{j_{i1}} \leftarrow p_{j_{i1}} + \frac{1}{M}$;
8      $k \leftarrow 2$;
9      while $k \leq L$ and $\frac{\alpha}{C}(d_{ik} - d_{i1}) < \frac{1}{2}(1 - \alpha)$ do
10         $p_{j_{ik}} \leftarrow p_{j_{ik}} + \frac{1}{MN}$;
11         $p_{j_{i1}} \leftarrow p_{j_{i1}} - \frac{1}{MN}$;
12         $k \leftarrow k + 1$;
```

Then the algorithms assign probability mass to the nearest neighbors of each example. KNN-Uniform finds the largest K that satisfies the condition in Line 5 and then assigns the probability mass of each query example evenly to its K-nearest neighbors. KNN-TV assigns $\frac{1}{MN}$ unit of probability mass to the nearest neighbors that satisfy the condition in Line 9 and the rest to the 1-nearest neighbor. KNN-KDE assigns probability mass to the nearest neighbors proportional to the inverse of their kernel density estimates (Line 21-24). The sizes of the neighborhoods are determined by Line 21-24, where we increase the limit s on the sum of the inverse of the density estimates over the neighborhood until the condition on Line 21 breaks. We use a priority queue to keep track of the possible values s can take and retrieve the smallest one in each iteration.

Algorithm 10: KNN-KDE.

```
1  Input: a set of query examples $\mathcal{Q} = \{q_i\}_{i=1}^{M}$ from the target domain,
     a corpus $\mathcal{D} = \{x_j\}_{j=1}^{N}$, number of nearest neighbors to prefetch
     $L > 1$, $\alpha \in [0, 1]$, $C > 0$;
2  Output: $p_1, \ldots, p_N$;
3  $\boldsymbol{j}, \boldsymbol{d} \leftarrow \textsc{GetKNN}(\mathcal{Q}, \mathcal{D}, L)$;
4  $\boldsymbol{\rho} \leftarrow \textsc{ComputeKDE}(\boldsymbol{j}, \mathcal{D})$;
5  $\mathcal{H} \leftarrow$ EmptyPriorityQueue();
6  for $i \in [M]$ do
7      $K_i \leftarrow 0$;
8      $c_i \leftarrow 0$;
9      $\mathcal{H}$.push$((1/\rho_{i1}, i))$;
10 while $\mathcal{H}$ is not empty do
11     $s, i \leftarrow \mathcal{H}$.pop();
12     $K_i \leftarrow K_i + 1$;
13     $c_i \leftarrow \sum_{k=1}^{K_i} (d_{i,K_i+1} - d_{ik})/\rho_{ik}$;
14     if $\frac{\alpha}{C} \sum_{i=1}^{M} c_i \geq (1 - \alpha)M$ then
15         $s^* \leftarrow s$;
16         break;
17     if $K_i + 1 < L$ then
18         $\mathcal{H}$.push$((s + 1/\rho_{i,K_i+1}, i))$;
19 for $j \in [N]$ do
20     $p_j \leftarrow 0$;
21 for $i \in [M]$ do
22     for $k \in [K_i]$ do
23         $p_{j_{ik}} \leftarrow p_{j_{ik}} + 1/(Ms^* \cdot \rho_{ik})$;
24     $p_{j_{i,K_i+1}} \leftarrow p_{j_{i,K_i+1}} + \frac{1}{M} - \sum_{k=1}^{K_i} 1/(Ms^* \cdot \rho_{ik})$;
```

Note that KNN-TV assigns almost all the probability mass to the 1-nearest neighbor of each query example when the largest k satisfying the condition in Line 9 is much less than N. Such probability assignment will result in overfitting to the 1-nearest neighbors of the query examples, as we can see in the experiment in Section 5.7.5.3.

In KDE-KNN, we also precompute the kernel density estimate for the L-nearest neighbors of each query example. To estimate the kernel density of each example, we need to compute the distance between it and all the examples in the candidate pool. To reduce the computational cost, we make the following adjustments. First, for each example, we use its I-nearest neighbors among the prefetched examples as the set for KDE. Let $\mathcal{D}'$ be the set containing the L-nearest neighbors of all the query points and $\mathcal{N}_x$ be the I-nearest neighbors of x in $\mathcal{D}'$. We compute the KDE of example x as $\sum_{x' \in \mathcal{N}_x} 1 - \frac{f(x,x')^2}{h^2}$. Second, we build a Faiss index on $\mathcal{D}'$ and query the index for the nearest neighbors and the distances.

Apart from GetKNN and ComputeKDE, KNN-Uniform and KNN-TV run in $O(ML)$ time, and KNN-KDE runs in $O(ML \log M)$ time.

5.6 Discussion

As large language models (LLMs) become increasingly powerful, it raises the question of whether we can leverage models like GPT-4 (OpenAI, 2023) to perform task-specific selection. We envision the following approaches of utilizing LLMs in task-specific data selection and discuss their limitations that require further exploration.

First, we can provide the query examples and the candidates directly to an LLM and ask the models to select a subset of candidates that match the distribution of the query examples with a certain degree of diversity. However, the size of the candidate pool may contain millions or billions of examples, making it impossible to include them in a single input sequence, given the limits on the input length of LLMs (for instance, GPT-4 has a limit of 32,768 tokens). Though we can fit the candidates into the inputs by breaking them into batches, it is not clear how we combine results across batches to form the final training set.

Second, we may ask an LLM to score the candidates based on their similarities to the query examples and select the candidates with the top scores. This selection method is similar to LESS (Xia et al., 2024) which ranks individual examples without considering the alignment of distribution and diversity in the selected data.

Lastly, we may present the query examples to an LLM and instruct the model to generate scripts for selecting task-specific training data from the candidates. However, LLMs are prone to hullicinations (Liu et al., 2024), which may result in code that might be unreliable or even non-executable.

5.7 Experiments

We evaluate instantiations of our framework against a diverse range of competing methods on natural language understanding tasks from various domains, where the selected are used for continued pretraining. We seek to answer the following questions:

1. Whether our framework outperforms the competing methods by incorporating optimal transport for distribution alignment?
2. How robust is our framework to near-duplicates in the candidate pool compared to the competing methods?
3. Is the runtime of our method reasonable for practical applications?

We also provide micro-benchmarks to study the effects of the hyperparameters in our framework. Finally, we show the effectiveness of our framework in improving the performance of instruction tuning for modern large language models in task-specific question answering.

5.7.1 Experimental Setup

Domain-Specific Tasks We consider six downstream datasets across diverse domains. All the datasets focus on classification tasks. The datasets are commonly used in the literature of domain-adaptive continued pretraining (Gururangan et al., 2020; Xie et al., 2023). The properties of the datasets and tasks are provided in Table 5.1. We provide a detailed description of each dataset below.

- **ChemProt** (Kringelum et al., 2016) is from the biomedical field, containing descriptions of interactions between chemicals and proteins. The task is to predict the relations between the chemicals and proteins.
- **RCT** (Dernoncourt and Lee, 2017) is from the biomedical field, containing sentences from abstracts of biomedical papers. The task is to predict the role of a given sentence in the abstract.
- **IMDB** (Maas et al., 2011) contains movie reviews and the task is sentiment classification.
- **Helpfulness** (McAuley et al., 2015) collects reviews from the Amazon web store and the task is sentiment classification.
- **SCIERC** (Luan et al., 2018) contains sentences from scientific abstracts in AI conferences. The task is to predict the relation between the scientific entities mentioned in a sentence.
- **AGNews** (Zhang et al., 2015) is a collection of news articles, and the task is to predict the topic of each article.

Corpus (Candidate Pool) We select data for continued pertaining from a corpus consisting of 150M sequences. The corpus is created as follows. We first take a random 10% of the documents from The Pile (Gao et al., 2020). Then following Xie et al. (2023), we segment the documents into chunks of

Dataset	Train	Validation	Test	Classes
ChemProt	4,169	2,427	3,469	13
RCT	180,040	30,212	30,135	5
IMDB	20,000	5,000	25,000	2
Helpfulness	115,251	5,000	25,000	2
SCIERC	3,219	455	974	7
AGNews	114,947	4,999	7,596	4

Table 5.1: Training, validation, test sizes and the number of classes in the datasets.

128 tokens according to whitespace tokenization and apply a quality filter to discard sequences that are extremely short or contain too many special tokens. We refer the reader to Xie et al. (2023) for the details.

Target-Task Data Accessibility To simulate different levels of access to annotated data in the target task, we consider three settings with varying sizes of annotated data (1K, 3K, and 10K). When the size is set to M and the original training set for the target task is larger than M, we sub-sample it by choosing M examples uniformly at random without replacement. The validation set for the target task is sub-sampled in the same way.

Evaluation Protocol For each task, we provide the training set to the selection methods as the query examples to guide the selection. Then we perform continued pretraining on 1M examples selected by each method from the corpus for one epoch, starting from the pretrained ALBERT (Lan et al., 2019) model. Afterwards, we finetune the model on the task-specific training set and evaluate it on the corresponding test set. To reduce the effect of instability when finetuning on small sets (Mosbach et al., 2021), we finetune five times with different random seeds and choose the model with the highest F1 score on the validation set. Following standard practice (Devlin et al., 2019; Lan et al., 2019), we use masked language modeling as the objective for continued pretraining and add a feedforward layer to the final layer [CLS] token for classification during finetuning. The hyperparameters for

maximum token length	256
batch size	128
optimizer	AdamW
weight decay	0.01
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	1e-6
warmup ratio	0.1
learning rate scheduler	linear
learning rate	5e-4

Table 5.2: Hyperparameters for continued pretraining.

maximum token length	256
batch size	16
epochs	10
patience for early stopping	3 epochs
optimizer	AdamW
weight decay	0.1
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	1e-6
warmup ratio	0.1
learning rate scheduler	linear
learning rate	5e-5

Table 5.3: Hyperparameters for finetuning. We set patience for early stopping to 3 epochs so that finetuning stops when the validation F1 score does not increase for 3 epochs.

pretraining and finetuning are in Table 5.2 and Table 5.3. Following previous works (Gururangan et al., 2020; Yao et al., 2021; Xie et al., 2023), we report micro-F1 scores for ChemProt and RCT, and macro-F1 scores for the others. All the experiments are repeated five times with different random seeds that control data selection, continued pretraining, and finetuning. We take the mean across five runs and also report the standard deviation.

Instantiations of Our Framework We consider two instantiations of our framework, KNN-Uniform and KNN-KDE with $C = 5$ and $\alpha = 0.6$. For KNN-KDE, we set the kernel size h to 0.1. We exclude KNN-TV from our main evaluation but report its performance as a micro-benchmark.

We encode the examples in the corpus and the task-specific representatives into $\mathbb{R}^{512}$ using the Universal Sentence Encoder (Cer et al., 2018) and use l_2 distance as the distance function.

GetKNN is implemented as two-stage retrieval. We first build a coarse Faiss (Douze et al., 2024) index for the corpus $\mathcal{D}$ and use it to retrieve the 2000 nearest neighbors of each query example. The retrieved examples form a new set $\mathcal{D}'$. Then we build a fine-grained index for $\mathcal{D}'$ and use it to retrieve and return the 2000 nearest neighbors of each query example. ComputeKDE in KNN-KDE computes the kernel density of each example in $\mathcal{D}'$ by retrieving its 1000 nearest neighbors using the fine-grained index. The coarse index is `OPQ56_112,IVF65536_HNSW32,PQ7+56`, and the fine-grained index is `IndexFlatL2`. We refer the readers to the Faiss documentation[3] for the details of those indexes.

Baselines We consider the following competing methods. Top-Cosine and Top-Logit are based on embeddings, for which we use the same embeddings as our methods.

- **Uniform** selects examples from the corpus uniformly at random.
- **Top-Cosine** (Aharoni and Goldberg, 2020) takes the element-wise mean of the embedded query examples and ranks the examples in the corpus by the cosine similarity to the mean. Examples with top similarities are retrieved.
- **Top-Logit** (Aharoni and Goldberg, 2020) trains a logistic regression model to decide whether an example belongs to the target domain and retrieves the examples with top logits. We train the model

[3]https://github.com/facebookresearch/faiss

with the query examples and a set of negative examples of the same size. Following Aharoni and Goldberg (2020), we sample the negative examples by ranking the corpus using Top-Cosine and taking a random sample from the bottom two-thirds.

- **DSIR** (Xie et al., 2023) selects examples by importance resampling to match the unigram and bigram distribution of the query examples. The importance weights are decided by two bag-of-ngrams generative models trained on the target-task representatives and 1% of the corpus.
- **IntellSelect** (Moore and Lewis, 2010) ranks the examples by the difference in cross-entropy of two 4-gram language models and retrieves the top examples. The first model is trained on the query examples, and the second is trained on 1% of the corpus.

5.7.2 End-to-End Comparisons

We compare the two instantiations of our framework against the competing methods under various sizes of downstream annotated datasets.

We evaluate the performance of the selection methods provided with different sizes (1K, 3K, and 10K) of annotated training data from the target target. Note that the target-task training sets are used for guiding the selection as well as finetuning the model. The test F1 scores of the downstream classification tasks are reported in Table 5.4, 5.5 and 5.6. As a reference point, we provide the performance of finetuning the model directly without continued pretraining in the first row (Base) in each table.

The results show the effectiveness of our framework in selecting data for task-specific continued pretraining. Notably, KNN-KDE consistently outperforms Base and Uniform, and the gap is significant in most settings. Additionally, KNN-KDE outperforms all the baseline methods in 11 out of the 16 settings. Top-Cosine and Top-Logit perform comparably to our methods (KNN-KDE and KNN-Uniform) when the size of the annotated

Dataset	ChemProt	RCT	IMDB	Helpfulness	SCIERC	AGNews
Base	$71.25_{1.21}$	$79.82_{0.70}$	$88.32_{0.44}$	$63.13_{0.98}$	$66.49_{2.35}$	$87.17_{0.47}$
Uniform	$73.10_{0.98}$	$80.44_{0.56}$	$87.56_{0.57}$	$63.06_{1.70}$	$65.58_{3.93}$	$87.00_{0.34}$
DSIR	$76.55_{0.83}$	$81.91_{0.17}$	$88.18_{0.67}$	$64.50_{0.55}$	$68.83_{0.46}$	$87.50_{0.85}$
IntellSelect	$63.28_{2.28}$	$74.19_{1.00}$	$81.82_{0.87}$	$60.98_{1.01}$	$55.87_{2.34}$	$83.90_{0.59}$
Top-Cosine	$78.07_{0.43}$	$\mathbf{82.22_{0.51}}$	$89.19_{0.35}$	$62.36_{2.01}$	$74.01_{0.78}$	$87.47_{0.62}$
Top-Logit	$78.17_{1.09}$	$81.71_{0.27}$	$88.82_{0.47}$	$\mathbf{65.16_{0.69}}$	$74.02_{1.30}$	$\mathbf{87.53_{0.50}}$
KNN-Uniform	$77.44_{0.81}$	$81.55_{0.31}$	$89.35_{0.25}$	$63.92_{1.81}$	$71.89_{4.57}$	$87.35_{0.21}$
KNN-KDE	$\mathbf{78.25_{1.21}}$	$81.93_{0.27}$	$\mathbf{89.94_{0.17}}$	$63.92_{1.24}$	$\mathbf{74.32_{3.73}}$	$87.37_{0.31}$

Table 5.4: F1 scores of the downstream tasks with 1K annotated data from the target domain. Standard deviations are shown in the subscripts.

target-task dataset is limited to 1K, but fall behind as the size increases. A possible reason is that Top-Cosine and Top-Logit use aggregated statistics of the query examples to retrieve pretraining data, losing more information as the number of query examples grows. In contrast, our methods consider each individual query example for distribution alignment. N-gram-based methods such as DSIR perform less effectively on datasets such as IMDB and SCIERC compared to the embedding-based methods, likely due to their incapability of capturing high-level semantics. IntellSelect shows notably poor performance possibly because the size of the query examples is not sufficient to train the 4-gram language models.

KNN-Uniform shows comparable or only slightly worse performance compared to KNN-KDE since the fraction of near-duplicates in the corpus we use is small. Only 6.8% of the examples have near-duplicates with distances less than 0.1, and among the 6.8% the average number of near-duplicates per example is 11.53. In the next section, we manually inject duplicates into the corpus to test the robustness to near-duplicates.

Dataset	ChemProt	RCT	IMDB	Helpfulness	SCIERC	AGNews
Base	$78.50_{0.40}$	$81.62_{0.15}$	$89.00_{0.19}$	$63.99_{0.65}$	$77.40_{1.61}$	$88.11_{0.21}$
Uniform	$78.88_{0.64}$	$82.32_{0.17}$	$88.71_{0.20}$	$65.19_{0.63}$	$78.55_{1.21}$	$88.49_{0.31}$
DSIR	$82.84_{0.39}$	$82.84_{0.22}$	$89.53_{0.16}$	$65.60_{0.67}$	$78.39_{1.06}$	$89.34_{0.18}$
IntellSelect	$72.26_{0.90}$	$78.50_{0.30}$	$85.79_{0.31}$	$62.88_{0.73}$	$68.31_{2.32}$	$86.10_{0.36}$
Top-Cosine	$\mathbf{83.86_{0.66}}$	$83.02_{0.28}$	$89.99_{0.21}$	$66.30_{0.76}$	$80.05_{1.31}$	$89.03_{0.34}$
Top-Logit	$82.60_{0.31}$	$83.00_{0.17}$	$90.10_{0.22}$	$\mathbf{66.57_{0.47}}$	$80.59_{0.97}$	$89.07_{0.28}$
KNN-Uniform	$82.32_{0.56}$	$\mathbf{83.52_{0.14}}$	$90.54_{0.25}$	$64.81_{0.59}$	$80.18_{1.34}$	$89.32_{0.30}$
KNN-KDE	$83.05_{0.75}$	$83.33_{0.10}$	$\mathbf{90.77_{0.17}}$	$65.53_{0.69}$	$\mathbf{80.75_{0.92}}$	$\mathbf{89.65_{0.12}}$

Table 5.5: F1 scores of the downstream tasks with 3K annotated data from the target domain. Standard deviations are shown in the subscripts.

Dataset	RCT	IMDB	Helpfulness	AGNews
Base	$82.96_{0.21}$	$90.18_{0.10}$	$64.43_{0.59}$	$89.55_{0.40}$
Uniform	$83.61_{0.15}$	$90.39_{0.13}$	$66.40_{0.77}$	$90.18_{0.22}$
DSIR	$84.01_{0.18}$	$90.96_{0.22}$	$67.70_{0.43}$	$90.48_{0.33}$
IntellSelect	$81.06_{0.17}$	$88.07_{0.17}$	$65.80_{0.28}$	$88.58_{0.21}$
Top-Cosine	$84.24_{0.23}$	$91.30_{0.08}$	$67.66_{0.14}$	$90.41_{0.23}$
Top-Logit	$84.11_{0.08}$	$91.35_{0.12}$	$67.86_{0.18}$	$90.45_{0.03}$
KNN-Uniform	$\mathbf{84.36_{0.17}}$	$91.73_{0.23}$	$\mathbf{68.30_{0.51}}$	$90.74_{0.17}$
KNN-KDE	$84.26_{0.23}$	$\mathbf{91.75_{0.19}}$	$68.22_{0.57}$	$\mathbf{90.78_{0.13}}$

Table 5.6: F1 scores of the downstream tasks with 10K annotated data from the target domain. Standard deviations are shown in the subscripts.

5.7.3 Robustness to Near-Duplicates

We evaluate the robustness of the selection methods against near-duplicates in the corpus. We inject duplicates into the corpus and follow the same evaluation protocol described in Section 5.7.1. We set different levels of duplication by varying the fraction of examples chosen for duplication and the duplication factor (number of duplicates per example). The fraction for duplication is set to 0.1% / 1%, and the duplication factor is set to 10 / 100 / 1000. For example, if the fraction for duplication is 0.1% and the duplication factor is 10, we randomly choose 0.1% of the examples from the corpus and duplicate each 10 times.

We use ChemProt (1K), AGNews (3K), and IMDB (10K) to perform the analysis, where the numbers in the parentheses represent the sizes of the annotated data. For a clearer illustration, we omit Uniform and IntellSelect from our analysis in this experiment due to their consistently poor performance in the end-to-end comparison shown by Section 5.7.2.

The results show that KNN-KDE is the only method that is robust to all the duplication settings. We observe that under low duplication levels, specifically when (fraction for duplication, duplication factor) is (0.1%, 10), (0.1%, 100), or (1%, 10), all the methods perform similarly to the case without duplication. Given that the injected duplicates constitute less than 10% of the corpus in those settings, it is not surprising that they do not have much effect on the downstream performance. However, when the duplication factor is increased to 1000 with the fraction for duplication set to 0.1%, the performance of DSIR and Top-Cosine drops, whereas KNN-KDE and KNN-Uniform retain their performance. Moreover, when the duplication factor is increased to 100 and 1000 with the fraction set to 1%, all the methods except KNN-KDE show a notable decline in their performance.

5.7.4 Runtime and Scalability

We report the runtime of the evaluated methods. Each method can be split into a pre-processing stage and a selection stage. In the pre-processing stage, each method makes transformations and collects statistics for the corpus. Top-Cosine and Top-Logit embed the examples in the corpus, and our method further builds indexes for the embeddings. DSIR computes the n-gram distribution based on a sample from the corpus and computes the n-gram representation for every example in the corpus. IntellSelect trains a 4-gram language model on a sample from the corpus. The results from this stage can be reused for different tasks. In the selection stage, each method computes importance scores and selects task-specific examples. The

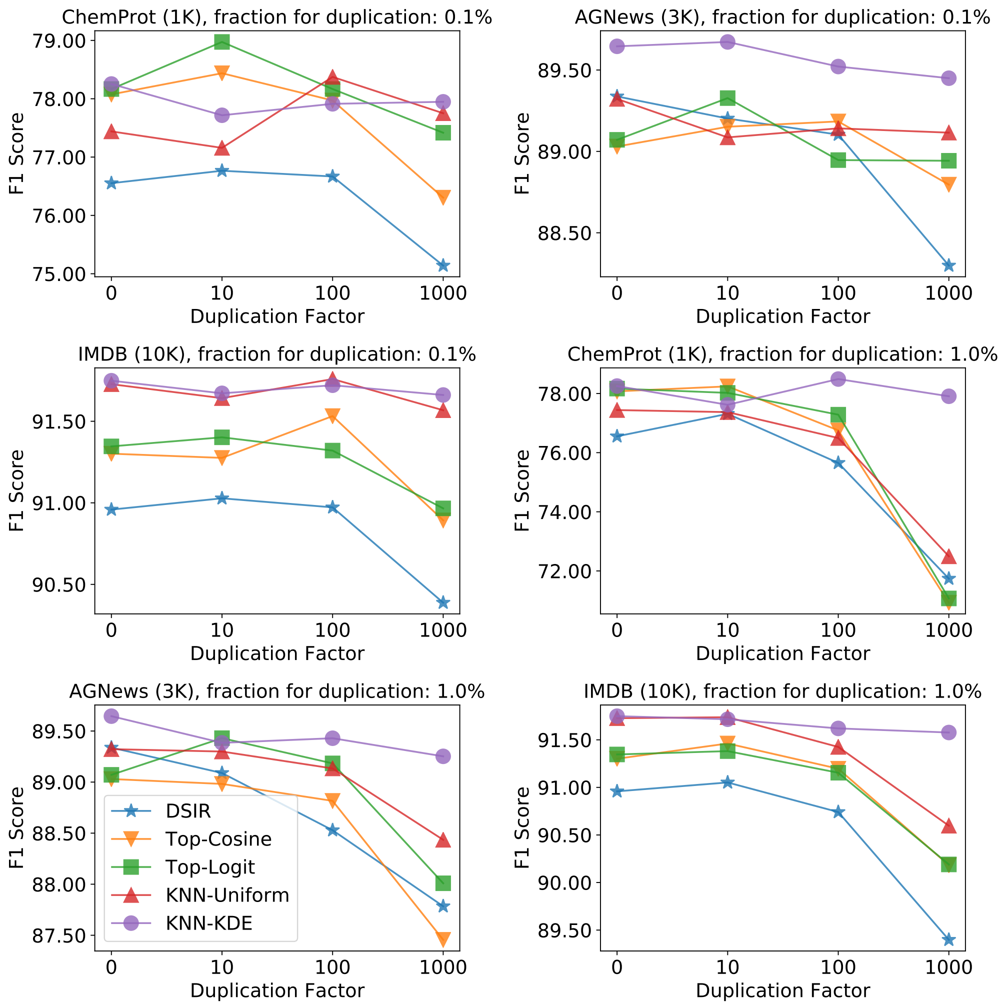


Figure 5.2: F1 scores of the downstream tasks under different duplication settings.

Stage	Pre-Processing	Selection
DSIR	18.78	0.13
IntellSelect	4.23	2.44
Top-Cosine	6.78	0.10
Top-Logit	6.78	0.11
KNN-Uniform	28.38	0.11
KNN-KDE	28.38	0.70

Table 5.7: Runtime (in hours) of the selection methods.

runtime for both stages is shown in Table 5.7. For the selection stage, we use IMDB (10K). We use a machine with an Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz (40 cores) and 250GB RAM. The example embedding is computed using an NVIDIA Tesla V100 GPU with 32GB memory, while the other computations are on the CPU. For all the methods except IntellSelect, the runtime is dominated by the pre-processing stage, while the selection stage can be finished within 1 hour. While our method takes more time in the pre-processing stage compared to Top-Cosine and Top-Logit due to the cost of index building, the cost is one-time and the index can be reused for a variety of tasks that require similarity search. In general, our methods are not excessively more expensive than the others.

5.7.5 Micro-Benchmarks

In this section, we provide micro-benchmarks that study the effects of the hyperparameters in our framework. In addition, we show the performance of KNN-TV, an instantiation of our framework that is not covered in Section 5.7.2. The datasets and sizes used in the micro-benchmarks are ChemProt (1K), AGNews (3K), and IMDB (10K).

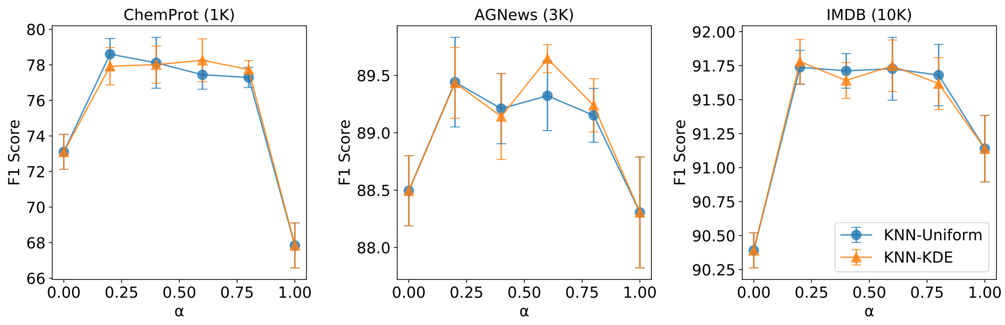


Figure 5.3: Performance of KNN-KDE when α varies. The error bar shows the standard deviation.

5.7.5.1 Tradeoff between Distribution Alignment and Diversity

We study the effects of α, the hyperparameter that controls the tradeoff between distribution alignment and diversity in our framework. We vary the value of α in KNN-Uniform and KNN-KDE and report the F1 scores of the downstream tasks in Figure 5.3. In all three datasets, we observe a notable drop in F1 scores when $\alpha = 0$ or $\alpha = 1$, and consistent performance when the value of α is set to other values. Note that KNN-Uniform or KNN-KDE is equivalent to Uniform when $\alpha = 0$, and transports all the probability mass of each query example to its 1-nearest-neighbor in the corpus when $\alpha = 1$. The former does not consider distribution alignment, while the latter results in overfitting to the 1-nearest-neighbors. For the other values of α, we report the corresponding neighborhood size (the final K in KNN-Uniform and the average of the final K_i in KNN-KDE) in Table 5.10. The consistent performance with $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$ shows that our framework is not sensitive to the choice of α.

5.7.5.2 Effects of Kernel Size in KNN-KDE

We vary the kernel size for the kernel density estimation in KNN-KDE. The performance is shown in Table 5.8. The F1 scores of all three downstream tasks are consistent across different choices of kernel size. The results show

Kernel Size	0.1	0.3	0.5
ChemProt (1K)	$78.25_{1.21}$	$78.48_{0.92}$	$77.75_{0.92}$
AGNews (3K)	$89.65_{0.12}$	$89.33_{0.45}$	$89.30_{0.38}$
IMDB (10K)	$91.75_{0.19}$	$91.59_{0.12}$	$91.80_{0.12}$

Table 5.8: Performance of KNN-KDE when the kernel size varies. F1 scores of the downstream tasks are reported with standard deviations shown in the subscripts.

Dataset	ChemProt (1K)	AGNews (3K)	IMDB (10K)
KNN-TV	$67.85_{0.60}$	$88.47_{0.23}$	$91.19_{0.22}$
KNN-KDE ($\alpha = 1$)	$67.84_{1.26}$	$88.30_{0.48}$	$91.13_{0.24}$
KNN-KDE ($\alpha = 0.6$)	$78.25_{1.20}$	$89.64_{0.12}$	$91.74_{0.19}$

Table 5.9: The performance of KNN-TV compared with KNN-KDE ($\alpha = 1$) and KNN-KDE ($\alpha = 0.6$). F1 scores of the downstream tasks are reported with standard deviations shown in the subscripts.

that the performance of KNN-KDE is not sensitive to the kernel size.

5.7.5.3 Performance of KNN-TV

We evaluate KNN-TV ($C = 0.25, \alpha = 0.6$) and show the results in Table 5.9. KNN-TV performs similarly to KNN-KDE ($\alpha = 1$), and significantly worse than KNN-KDE ($\alpha = 0.6$). The reason is that KNN-TV assigns almost all the probability mass (more than 99.99%) to the 1-nearest neighbor of each query example and causes overfitting to them, a behavior similar to KNN-KDE ($\alpha = 1$) as expected.

5.7.6 Evaluation on Targeted Instruction Tuning

Targeted instruction tuning is the process of finetuning pretrained generative language models on specific instructions with desired responses, tailoring the model to the target tasks. In this subsection, we select data for targeted

Dataset	ChemProt (1K)	AGNews (3K)	IMDB (10K)
$\alpha = 0.2$	959 / 993	813 / 830	918 / 928
$\alpha = 0.4$	398 / 408	342 / 346	372 / 373
$\alpha = 0.6$	189 / 191	165 / 164	177 / 174
$\alpha = 0.8$	76 / 74	69 / 65	72 / 68

Table 5.10: The neighborhood size of KNN-Uniform / KNN-KDE for different values of α. The numbers before the slashes are for KNN-Uniform and those after are for KNN-KDE.

instruction tuning from a corpus given several examples of the use cases from the target task. We follow the setting from Xia et al. (2024).

Target Tasks. We consider three tasks from standard benchmarks that are widely used by previous works (Xia et al., 2024; Touvron et al., 2023; Brown et al., 2020) for the evaluation of language models. The properties of the datasets and the evaluation metrics are shown in Table 5.11. The number of shots is the number of question-answering examples provided in the prompt when we query the model.

- **TydiQA** (Clark et al., 2020) is a multilingual question-answering dataset containing 11 languages. The task is to find the answer to the given question from a chunk of text.
- **MMLU** (Hendrycks et al., 2021) consists of multiple-choice question across 57 domains. The task is to choose the correct answer from the given options.
- **BBH** (Suzgun et al., 2022) is a collection of questions and answers focused on common sense reasoning. The task is to answer the questions, while the correct answers are not necessarily included in the provided context or questions.

Corpus We use a combination of the following datasets as the corpus for selection: FLAN V2 (Longpre et al., 2023), COT (Wei et al., 2022),

Dataset	# Test Instances	# Query Examples	# Shots	Metric
TydiQA	1,713	9	1	F1 score
MMLU	18,721	285	5	Accuracy
BBH	920	81	3	Accuracy

Table 5.11: Information of the datasets and the corresponding evaluation metrics.

Dolly (Conover et al., 2023), and Open Assistant (Köpf et al., 2023). Those datasets contain human-crafted question-answering pairs and dialogues, with a total of 270K examples.

Model We evaluate the selection methods using two base models: Llama-2-7b (Touvron et al., 2023) and Mistral-7B (Jiang et al., 2023). Both models are generative language models that output texts for input prompts.

Encoding We encode the examples following Xia et al. (2024) using rescaled and randomly projected gradients from a Llama-2-7b model finetuned on a random 5% of the corpus. Specifically, we finetune the base model on the randomly selected dataset for 4 epochs and use the gradients from the checkpoint at the end of each epoch as the example encoding. The dimension of the projected gradient from each epoch is 8,192. We refer the readers to Xia et al. (2024) for more details.

Methods We consider the following methods.

- **Rand** selects a random subset from the corpus.
- **LESS** (Xia et al., 2024) is the state-of-the-art method for targeted instruction tuning. LESS selects training data from the corpus based on their gradient similarity to the query examples. Their formulation is also closely related to the influence function (Koh and Liang, 2017).
- **KNN-Uniform** (denoted as "Ours" in Table 5.12) is an instantiation of our framework. We choose $\alpha = 0.2$ for MMLU and $\alpha = 0.075$ for the other two datasets. The constant C is set to 5. For each

example, we concatenate the gradients from the 4 checkpoints so that each example is encoded as a 32,768-dimensional vector. We do exact nearest neighbor search since the scale of the corpus is relatively small.

Evaluation Protocol We finetune the base model on the selected data for 4 epochs. The dataset size is 0.5% / 1.0% / 5% of the corpus. Since our method is based on probabilistic sampling, we do not select a fixed training set. Instead, in each epoch we sample a new set from the corpus. The hyperparameters for finetuning follows Xia et al. (2024). We repeat each experiment for three runs and report the mean and standard deviation.

Table 5.12: Performance of instruction tuning with dataset selected by our method compared with the baselines. The subscripts represent the standard deviations.

Model		Llama-2-7b			Mistral-7B		
Dataset		TydiQA	MMLU	BBH	TydiQA	MMLU	BBH
Base		14.1	45.7	38.0	20.9	62.4	56.4
Full		51.4	51.5	42.2	55.5	59.7	49.8
Ratio 0.5%	Rand	$41.2_{1.4}$	$45.0_{0.4}$	$37.4_{0.4}$	$54.9_{0.1}$	$59.0_{1.2}$	$51.9_{0.2}$
	LESS	$46.4_{1.2}$	$46.3_{0.6}$	$38.8_{0.6}$	$54.7_{3.0}$	$60.7_{0.3}$	$54.6_{0.8}$
	Ours	$\mathbf{51.6_{0.8}}$	$\mathbf{47.9_{0.2}}$	$\mathbf{39.2_{0.7}}$	$\mathbf{60.7_{2.0}}$	$\mathbf{61.1_{0.3}}$	$\mathbf{58.0_{0.3}}$
Ratio 1.0%	Rand	$43.2_{0.8}$	$46.0_{0.2}$	$37.9_{0.8}$	$54.4_{1.4}$	$59.1_{0.4}$	$53.4_{0.4}$
	LESS	$48.7_{0.6}$	$48.4_{0.1}$	$38.7_{0.6}$	$57.0_{1.0}$	$60.6_{0.1}$	$55.0_{1.2}$
	Ours	$\mathbf{54.0_{0.3}}$	$\mathbf{49.0_{0.1}}$	$\mathbf{40.1_{1.3}}$	$\mathbf{62.9_{0.9}}$	$\mathbf{61.2_{0.4}}$	$\mathbf{57.5_{0.4}}$
Ratio 5.0%	Rand	$45.5_{1.2}$	$46.0_{0.4}$	$39.1_{0.9}$	$52.8_{0.9}$	$59.6_{0.2}$	$53.7_{2.4}$
	LESS	$52.2_{0.8}$	$\mathbf{50.8_{0.2}}$	$40.9_{0.7}$	$59.5_{1.8}$	$\mathbf{61.0_{0.6}}$	$53.8_{0.7}$
	Ours	$\mathbf{52.4_{1.0}}$	$50.5_{0.4}$	$\mathbf{41.0_{0.4}}$	$\mathbf{61.7_{0.3}}$	$59.7_{0.9}$	$\mathbf{56.1_{0.6}}$

Results The results are shown in Table 5.12 where "Base" is the base model without finetuning and "Full" is the model finetuned on the whole corpus. We observe that **our method consistently outperforms the baselines with small selection ratios** (0.5% and 1.0%), where the gap can be quite large (over 2.5 points on TydiQA for both models and on

BBH for Mistral-7B). Our method gets close to or even outperforms Full on TydiQA for both models and on BBH for Mistral-7B with only 1% selection ratio. When the selection ratio increases to 5%, our methods still show comparable results to LESS. We observe a drop in the performance of our method on TydiQA and BBH when the selection ratio increases from 1% to 5%, which may be caused by overfitting. We can early stop the training process to avoid such behavior. We also notice that finetuning Mistral-7B on any selected set does not increase its accuracy on MMLU. The reason could be that the based Mistral-7B model has already been well tuned for multiple-choice questions.

5.8 Proofs

5.8.1 Proof of Theorem 5.4

Proof. Let $\mathcal{L}(\boldsymbol{\gamma}) = \frac{\alpha}{C}\sum_{i=1}^{M}\sum_{j=1}^{N}\gamma_{ij}d_{ij} + (1-\alpha)G_{\text{TV}}(\boldsymbol{\gamma})$ be the optimization objective. We prove the theorem by showing that for any $\boldsymbol{\gamma'} \in \mathbb{R}_{\geq 0}^{M\times N}$ that satisfy the constraint $(\forall i \in [M]\sum_{j=1}^{N}\gamma'_{ij} = \frac{1}{M})$, $\mathcal{L}(\boldsymbol{\gamma'}) \geq \mathcal{L}(\boldsymbol{\gamma}^*)$.

Let $\boldsymbol{\gamma''} \in \mathbb{R}_{\geq 0}^{M\times N}$ be a probability transport such that

$$\forall k \in \{2, \ldots, N\}, \gamma''_{ij^i_k} = \begin{cases} \gamma'_{ij^i_k}, & \text{if } d_{ij^i_k} - d_{ij^i_1} < \frac{(1-\alpha)C}{\alpha} \\ 0, & \text{otherwise} \end{cases}$$

We show that $\mathcal{L}(\boldsymbol{\gamma'}) \geq \mathcal{L}(\boldsymbol{\gamma''})$.

For any $i \in [M]$, let $\hat{k}_i = \max\{k \in [N] | d_{ij^i_k} - d_{ij^i_1} < \frac{(1-\alpha)C}{\alpha}\}$. Then we have

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\gamma}') - \mathcal{L}(\boldsymbol{\gamma}'') \\
=&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{j=1}^{N}(\gamma'_{ij} - \gamma''_{ij})d_{ij} + \frac{1-\alpha}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}(|\gamma'_{ij} - \frac{1}{MN}| - |\gamma''_{ij} - \frac{1}{MN}|) \\
=&\sum_{i=1}^{M}\sum_{j=1}^{N}[\frac{\alpha}{C}d_{ij}(\gamma'_{ij} - \gamma''_{ij}) + \frac{1-\alpha}{2}(|\gamma'_{ij} - \frac{1}{MN}| - |\gamma''_{ij} - \frac{1}{MN}|)] \\
=&\sum_{i=1}^{M}\sum_{k=1}^{N}[\frac{\alpha}{C}d_{ij^i_k}(\gamma'_{ij^i_k} - \gamma''_{ij^i_k}) + \frac{1-\alpha}{2}(|\gamma'_{ij^i_k} - \frac{1}{MN}| - |\gamma''_{ij^i_k} - \frac{1}{MN}|)] \\
=&\sum_{i=1}^{M}[\underbrace{\sum_{k=\hat{k}_i+1}^{N}\frac{\alpha}{C}(d_{ij^i_k} - d_{ij^i_1})\gamma'_{ij^i_k}}_{T_1} + \underbrace{\frac{1-\alpha}{2}\sum_{k=\hat{k}_i+1}^{N}(|\gamma'_{ij^i_k} - \frac{1}{MN}| - \frac{1}{MN})}_{T_2} + \\
&\underbrace{\frac{1-\alpha}{2}(|\gamma'_{ij^i_1} - \frac{1}{MN}| - |\gamma'_{ij^i_1} + \sum_{k=\hat{k}_i+1}^{N}\gamma'_{ij^i_k} - \frac{1}{MN}|)}_{T_3}]
\end{aligned}$$

The last equation is due to the fact that $\gamma''_{ij^i_k} = 0$ for $k > \hat{k}_i$ and $\gamma''_{ij^i_1} = \gamma'_{ij^i_1} + \sum_{k=\hat{k}_i+1}^{N}\gamma'_{ij^i_k}$. Since $d_{ij^i_k} - d_{ij^i_1} \geq \frac{(1-\alpha)C}{\alpha}$ for any $k > \hat{k}_i$, we have $T_1 \geq (1-\alpha)\sum_{k=\hat{k}_i+1}^{N}\gamma'_{ij^i_k}$. By the triangle equality, we have $T_2 \geq \frac{1-\alpha}{2}\sum_{k=\hat{k}_i+1}^{N}(-\gamma'_{ij^i_k})$ and $T_3 \geq \frac{1-\alpha}{2}\sum_{k=\hat{k}_i+1}^{N}(-\gamma'_{ij^i_k})$. Therefore, we have $T_1 + T_2 + T_3 \geq 0$ and consequently $\mathcal{L}(\boldsymbol{\gamma}') \geq \mathcal{L}(\boldsymbol{\gamma}'')$.

Let $\mathcal{K}^i_{\text{high}} = \{2 \leq k \leq \hat{k}_i | \gamma''_{ij^i_k} > \frac{1}{MN}\}$ and $\mathcal{K}^i_{\text{low}} = \{2 \leq k \leq \hat{k}_i | \gamma''_{ij^i_k} < \frac{1}{MN}\}$. Let $\boldsymbol{\gamma}''' \in \mathbb{R}^{M\times N}_{\geq 0}$ be a probability transport such that

$$\forall k \in \{2, \ldots, N\}, \gamma'''_{ij^i_k} = \begin{cases} \gamma^*_{ij^i_k}, & \text{if } k \in \mathcal{K}^i_{\text{high}} \\ \gamma''_{ij^i_k}, & \text{otherwise} \end{cases}$$

Then we show that $\mathcal{L}(\boldsymbol{\gamma}'') \geq \mathcal{L}(\boldsymbol{\gamma}''')$. Since $\gamma'''_{ij^i_k} = \frac{1}{MN}$ for $k \in \mathcal{K}^i_{\text{high}}$ and

$\gamma'''_{ij^i_1} = \gamma''_{ij^i_1} + \sum_{k\in\mathcal{K}^i_{\text{high}}}(\gamma''_{ij^i_k} - \frac{1}{MN})$, we have

$$\begin{aligned}&\mathcal{L}(\boldsymbol{\gamma}'') - \mathcal{L}(\boldsymbol{\gamma}''')\\ =&\sum_{i=1}^{M}[\underbrace{\sum_{k\in\mathcal{K}^i_{\text{high}}}\frac{\alpha}{C}(d_{ij^i_k} - d_{ij^i_1})(\gamma''_{ij^i_k} - \frac{1}{MN})}_{T_4} + \underbrace{\frac{1-\alpha}{2}\sum_{k\in\mathcal{K}^i_{\text{high}}}|\gamma''_{ij^i_k} - \frac{1}{MN}|}_{T_5} +\\ &\underbrace{\frac{1-\alpha}{2}(|\gamma''_{ij^i_1} - \frac{1}{MN}| - |\gamma''_{ij^i_1} + \sum_{k\in\mathcal{K}^i_{\text{high}}}(\gamma''_{ij^i_k} - \frac{1}{MN}) - \frac{1}{MN}|)}_{T_6}]\end{aligned}$$

Again following the triangle inequality, we have $T_6 \geq -\frac{1-\alpha}{2}\sum_{k\in\mathcal{K}^i_{\text{high}}}|\gamma''_{ij^i_k} - \frac{1}{MN}|$, and therefore $T_5 + T_6 \geq 0$. Since we also have $T_4 \geq 0$, it follows that $\mathcal{L}(\boldsymbol{\gamma}'') \geq \mathcal{L}(\boldsymbol{\gamma}''')$.

Finally, we show that $\mathcal{L}(\boldsymbol{\gamma}''') \geq \mathcal{L}(\boldsymbol{\gamma}^*)$. Since $\gamma^*_{ij^i_1} = \gamma'''_{ij^i_1} + \sum_{k\in\mathcal{K}^i_{\text{low}}}(\gamma'''_{ij^i_k} - \frac{1}{MN})$, we have

$$\begin{aligned}&\mathcal{L}(\boldsymbol{\gamma}''') - \mathcal{L}(\boldsymbol{\gamma}^*)\\ =&\sum_{i=1}^{M}[\underbrace{\sum_{k\in\mathcal{K}^i_{\text{low}}}\frac{\alpha}{C}(d_{ij^i_k} - d_{ij^i_1})(\gamma'''_{ij^i_k} - \frac{1}{MN})}_{T_7} + \underbrace{\frac{1-\alpha}{2}\sum_{k\in\mathcal{K}^i_{\text{low}}}|\gamma'''_{ij^i_k} - \frac{1}{MN}|}_{T_8} +\\ &\underbrace{\frac{1-\alpha}{2}(|\gamma'''_{ij^i_1} - \frac{1}{MN}| - |\gamma'''_{ij^i_1} + \sum_{k\in\mathcal{K}^i_{\text{low}}}(\gamma'''_{ij^i_k} - \frac{1}{MN}) - \frac{1}{MN}|)}_{T_9}]\end{aligned}$$

Since $d_{ij^i_k} - d_{ij^i_1} < \frac{(1-\alpha)C}{\alpha}$ for any $k \in \mathcal{K}^i_{\text{low}}$, we have $T_7 \geq (1-\alpha)\sum_{k\in\mathcal{K}^i_{\text{low}}}(\gamma'''_{ij^i_k} - \frac{1}{MN})$. Notice that $\gamma'''_{ij^i_1} \geq \frac{1}{MN}$ since $\gamma'''_{ij^i_1} = \frac{1}{M} - \sum_{k=2}^{N}\gamma'''_{ij^i_k}$ and for $k \in \{2,\ldots,N\}$, $\gamma'''_{ij^i_k} \leq \frac{1}{MN}$. We also have $\gamma'''_{ij^i_1} + \sum_{k\in\mathcal{K}^i_{\text{low}}}(\gamma'''_{ij^i_k} - \frac{1}{MN}) \geq \frac{1}{MN}$ since $\gamma^*_{ij^i_1} \geq \frac{1}{MN}$. Therefore, we have $T_8 + T_9 = (1-\alpha)\sum_{k\in\mathcal{K}^i_{\text{low}}}(\frac{1}{MN} - \gamma'''_{ij^i_k})$ and $T_7 + T_8 + T_9 \geq 0$. The it follows that $\mathcal{L}(\boldsymbol{\gamma}''') \geq \mathcal{L}(\boldsymbol{\gamma}^*)$.

Thus, we have $\mathcal{L}(\boldsymbol{\gamma}') \geq \mathcal{L}(\boldsymbol{\gamma}'') \geq \mathcal{L}(\boldsymbol{\gamma}''') \geq \mathcal{L}(\boldsymbol{\gamma}^*)$.

Next we show that if $\forall i \in [M] \forall k \in [N]$, $d_{ij^i_k} - d_{ij^i_1} \neq \frac{(1-\alpha)C}{\alpha}$ and

$d_{ij_1^i} \neq d_{ij_2^i}$, $\boldsymbol{\gamma}^*$ is the unique solution. We consider two cases. In the first case where $\forall i \in [M] \forall k \in [N]$, $d_{ij_k^i} - d_{ij_1^i} < \frac{(1-\alpha)C}{\alpha}$, we have $\forall i \in [M] \forall j \in [N], \gamma_{ij}^* = \frac{1}{MN}$. For any $\boldsymbol{\gamma'} \neq \boldsymbol{\gamma}^*$, there must exist $i \in [M], k \in \{2, \ldots, N\}$ such that $\gamma'_{ij_k^i} > \frac{1}{MN}$ in which case $T_4 > 0$ or $\gamma'_{ij_k^i} < \frac{1}{MN}$ in which case $T_7 > (1-\alpha) \sum_{k \in \mathcal{K}_{\text{low}}^i} (\gamma'''_{ij_k^i} - \frac{1}{MN})$. In the second case where $\exists i \in [M] k \in [N]$ such that $d_{ij_k^i} - d_{ij_1^i} > \frac{(1-\alpha)C}{\alpha}$, we have $T_1 > (1-\alpha) \sum_{k=\hat{k}_i+1}^{N} \gamma'_{ij_k^i}$ for that i. In both cases, $\mathcal{L}(\boldsymbol{\gamma'}) > \mathcal{L}(\boldsymbol{\gamma}^*)$ and thus $\boldsymbol{\gamma}^*$ is the unique solution.

$\square$

5.8.2 Proof of Theorem 5.3 and Theorem 5.5

We show that Theorem 5.3 states a special case of Theorem 5.5. Then we prove Theorem 5.5 and it follows that Theorem 5.3 holds as well.

We first show the connection between Theorem 5.3 and Theorem 5.5. In Theorem 5.5, when $\rho_j = 1$ for all $j \in [N]$, $s_k^i = k$ and s^* is the same as the K in Theorem 5.3. Then the optimal solution in Theorem 5.5 is also the same as the one in Theorem 5.3 if we substitute s^* by K and the ρ_j's by 1.

Let $\mathcal{L}(\boldsymbol{\gamma}) = \frac{\alpha}{C} \sum_{i=1}^{M} \sum_{j=1}^{N} \gamma_{ij} d_{ij} + (1-\alpha) G_{\text{KDE}}(\boldsymbol{\gamma})$ be the optimization objective. We prove Theorem 5.5 by showing that for any $\boldsymbol{\gamma'} \in \mathbb{R}_{\geq 0}^{M \times N}$ that satisfy the constraint $(\forall i \in [M] \sum_{j=1}^{N} \gamma'_{ij} = \frac{1}{M})$, $\mathcal{L}(\boldsymbol{\gamma'}) \geq \mathcal{L}(\boldsymbol{\gamma}^*)$.

For conciseness, we let $d_{(i,k)} = d_{ij_k^i}$ and $\gamma_{(i,k)} = \gamma_{ij_k^i}$.

We first show that $c(s)$ is a non-decreasing function. Since $c_i(s)$ is a step function and $\sum_{l=1}^{k} \frac{d_{(i,k+1)} - d_{(i,l)}}{\rho_{j_l^i}} - \sum_{l=1}^{k-1} \frac{d_{(i,k)} - d_{(i,l)}}{\rho_{j_l^i}} = \sum_{l=1}^{k} \frac{d_{(i,k+1)} - d_{(i,k)}}{\rho_{j_l^i}} \geq 0$ for any $k \in [N-1]$, $c_i(s)$ is non-decreasing. Therefore, $c_i(s) = \sum_{i=1}^{M} c_i(s)$ is a non-decreasing function.

Let $r' = \max_{i \in [M]} \max_{k \in [K_i]} \rho_{j_k^i} \gamma'_{(i,k)}$. We consider the following two cases.

In the first case when $r' \leq \frac{1}{Ms^*}$, we have

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\gamma}') - \mathcal{L}(\boldsymbol{\gamma}^*) \\
=&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{N} d_{(i,k)}(\gamma'_{(i,k)} - \gamma^*_{(i,k)}) + (1-\alpha)(G_{\text{KDE}}(\boldsymbol{\gamma}') - G_{\text{KDE}}(\boldsymbol{\gamma}^*)) \\
=&\underbrace{\frac{\alpha}{C}\sum_{i=1}^{M}[\sum_{k=1}^{K_i} d_{(i,k)}(\gamma'_{(i,k)} - \gamma^*_{(i,k)}) + d_{(i,K_i+1)}(\gamma'_{(i,K_i+1)} - \gamma^*_{(i,K_i+1)}) + \sum_{k=K_i+2}^{N} d_{(i,k)}\gamma'_{(i,k)}]}_{T_1} \\
&\underbrace{(1-\alpha)(G_{\text{KDE}}(\boldsymbol{\gamma}') - G_{\text{KDE}}(\boldsymbol{\gamma}^*))}_{T_2}
\end{aligned}$$

Since $\forall k \geq K_i{+}2, d_{(i,k)} \geq d_{(i,K_i+1)}$, and $\sum_{k=K_i+1}^{N} \gamma'_{(i,k)} - \gamma^*_{(i,K_i+1)} = -\sum_{k=1}^{K_i}(\gamma'_{(i,k)} - \gamma^*_{(i,k)})$, we have

$$\begin{aligned}
T_1 \geq&\frac{\alpha}{C}\sum_{i=1}^{M}[\sum_{k=1}^{K_i} d_{(i,k)}(\gamma'_{(i,k)} - \gamma^*_{(i,k)}) + d_{(i,K_i+1)}(\sum_{k=K_i+1}^{N}\gamma'_{(i,k)} - \gamma^*_{(i,K_i+1)})] \\
=&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{K_i}\frac{d_{(i,K_i+1)} - d_{(i,k)}}{\rho_{j_k^i}}(\rho_{j_k^i}\gamma^*_{(i,k)} - \rho_{j_k^i}\gamma'_{(i,k)}) \\
\geq&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{K_i}\frac{d_{(i,K_i+1)} - d_{(i,k)}}{\rho_{j_k^i}}(\frac{1}{Ms^*} - r')
\end{aligned}$$

Let $\hat{s} = \min_{i\in[M]} s^i_{K_i+1}$. Then we have $\hat{s} > s^*$ and $\sum_{i=1}^{M}\sum_{k=1}^{K_i}\frac{d_{(i,K_i+1)} - d_{(i,k)}}{\rho_{j_k^i}} = c(\hat{s})$. Since $c(s)$ is non-decreasing, we have $\frac{\alpha}{C}c(\hat{s}) \geq (1-\alpha)M$. Then it follows that $T_1 \geq (1-\alpha)M(\frac{1}{Ms^*} - r')$.

Let $\bar{s} = \sum_{j=1}^{N} 1/\rho_j$. Given the assumption that $s^* \leq \frac{1}{2}\bar{s}$, we have $\frac{1}{Ms^*} \geq 2\frac{1}{M\bar{s}}$. For any $i \in [M]$, for any $k \leq K_i$ we have $\rho_{j_k^i}\gamma^*_{(i,k)} = \frac{1}{Ms^*}$, and for $k = K_i{+}1$ we have $\rho_{j_k^i}\gamma^*_{(i,k)} = \frac{1}{Ms^*}(s^* - s^i_{K_i})\rho_{j_k^i} \leq \frac{1}{Ms^*}(s^i_{K_i+1} - s^i_{K_i})\rho_{j_k^i} = \frac{1}{Ms^*}$.

Therefore, $\max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma^*_{(i,k)} - \frac{1}{M\bar{s}}| = \frac{1}{Ms^*} - \frac{1}{M\bar{s}}$. Then we have

$$\begin{aligned}
T_2 =&(1-\alpha)M(\max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma'_{(i,k)} - \frac{1}{M\bar{s}}| - \max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma^*_{(i,k)} - \frac{1}{M\bar{s}}|)\\
\geq&(1-\alpha)M(\max_{i\in[M]}\max_{k\in[K_i]} |\rho_{j_k^i}\gamma'_{(i,k)} - \frac{1}{M\bar{s}}| - (\frac{1}{Ms^*} - \frac{1}{M\bar{s}}))\\
\geq&(1-\alpha)M(|r' - \frac{1}{M\bar{s}}| - (\frac{1}{Ms^*} - \frac{1}{M\bar{s}}))\\
\geq&(1-\alpha)M(r' - \frac{1}{Ms^*})
\end{aligned}$$

The last inequality follows the triangle inequality. Then it follows that $T_1 + T_2 \geq 0$ and $\mathcal{L}(\boldsymbol{\gamma}') - \mathcal{L}(\boldsymbol{\gamma}^*) \geq 0$.

In the second case when $r' > \frac{1}{Ms^*}$, let $\hat{K}_i = \max\{K \in [N]\cup\{0\} | \sum_{k=1}^{K} r'/\rho_{j_k^i} \leq \frac{1}{M}\}$. When $K > K_i$, $\sum_{k=1}^{K} r'/\rho_{j_k^i} > s^* r' > \frac{1}{M}$. Therefore, $\hat{K}_i \leq K_i$. Consider another probability transport $\boldsymbol{\gamma}'' \in \mathbb{R}_{\geq 0}^{M\times N}$ where

$$\gamma''_{(i,k)} = \begin{cases} r'/\rho_{j_k^i}, & \text{if } k \leq \hat{K}_i \\ \frac{1}{M} - \sum_{k=1}^{\hat{K}_i} r'/\rho_{j_k^i}, & \text{if } k = \hat{K}_i + 1 \\ 0, & \text{otherwise} \end{cases}$$

Note that by the definition of $\hat{K}_i$ we have $\gamma''_{(i,k)}\rho_{j_k^i} < r'$ for $k = \hat{K}_i + 1$.

Then we have

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\gamma}') - \mathcal{L}(\boldsymbol{\gamma}'')\\
=&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{N} d_{(i,k)}(\gamma'_{(i,k)} - \gamma''_{(i,k)}) + (1-\alpha)(G_{\text{KDE}}(\boldsymbol{\gamma}') - G_{\text{KDE}}(\boldsymbol{\gamma}''))\\
=&\underbrace{\frac{\alpha}{C}\sum_{i=1}^{M}[\sum_{k=1}^{\hat{K}_i} d_{(i,k)}(\gamma'_{(i,k)} - \gamma''_{(i,k)}) + d_{(i,\hat{K}_i+1)}(\gamma'_{(i,\hat{K}_i+1)} - \gamma''_{(i,\hat{K}_i+1)}) + \sum_{k=\hat{K}_i+2}^{N} d_{(i,k)}\gamma'_{(i,k)}]}_{T_3}\\
&\underbrace{(1-\alpha)(G_{\text{KDE}}(\boldsymbol{\gamma}') - G_{\text{KDE}}(\boldsymbol{\gamma}''))}_{T_4}
\end{aligned}$$

Since $\forall k \geq \hat{K}_i+2, d_{(i,k)} \geq d_{(i,\hat{K}_i+1)}$, and $\sum_{k=\hat{K}_i+1}^{N} \gamma'_{(i,k)} - \gamma''_{(i,\hat{K}_i+1)} = -\sum_{k=1}^{\hat{K}_i}(\gamma'_{(i,k)} - \gamma''_{(i,k)})$, we have

$$\begin{aligned}T_3 \geq& \frac{\alpha}{C}\sum_{i=1}^{M}[\sum_{k=1}^{\hat{K}_i} d_{(i,k)}(\gamma'_{(i,k)} - \gamma''_{(i,k)}) + d_{(i,\hat{K}_i+1)}(\sum_{k=\hat{K}_i+1}^{N} \gamma'_{(i,k)} - \gamma''_{(i,\hat{K}_i+1)})]\\ =& \frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{\hat{K}_i} \frac{d_{(i,\hat{K}_i+1)} - d_{(i,k)}}{\rho_{j_k^i}}(\rho_{j_k^i}\gamma''_{(i,k)} - \rho_{j_k^i}\gamma'_{(i,k)})\\ \geq& 0\end{aligned}$$

In addition, since $r' > \frac{1}{Ms^*} \geq \frac{2}{M\bar{s}}$ and $\rho_{j_k^i}\gamma''_{(i,k)} \leq r'$ for any $i \in [M]$ and $k \in [N]$, we have $\max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma'_{(i,k)} - \frac{1}{M\bar{s}}| \geq \max_{i\in[M]}\max_{k\in[K_i]} |\rho_{j_k^i}\gamma'_{(i,k)} - \frac{1}{M\bar{s}}| = r' - \frac{1}{M\bar{s}}$ and $\max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma''_{(i,k)} - \frac{1}{M\bar{s}}| \leq r' - \frac{1}{M\bar{s}}$. Therefore,

$$\begin{aligned}T_4 =& (1-\alpha)M(\max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma'_{(i,k)} - \frac{1}{M\bar{s}}| - \max_{i\in[M],k\in[N]} |\rho_{j_k^i}\gamma''_{(i,k)} - \frac{1}{M\bar{s}}|)\\ \geq& 0\end{aligned}$$

Then it follows that $\mathcal{L}(\boldsymbol{\gamma'}) - \mathcal{L}(\boldsymbol{\gamma''}) \geq 0$

Let $\mathcal{S}' = \{s \in \mathcal{S} | \frac{1}{Mr'} < s \leq s^*\}$ and $s^{(1)}, \ldots, s^{(|\mathcal{S}'|)}$ be the elements in $\mathcal{S}'$ in the ascending order. Let $\boldsymbol{\gamma^{(0)}} = \boldsymbol{\gamma''}$, $s^{(0)} = \frac{1}{Mr'}$ and $K_i^{(0)} = \hat{K}_i$. For $t \in [|\mathcal{S}'|]$, let $K_i^{(t)} = \max\{k \in [N] | s_k^i \leq s^{(t)}\}$. we consider the probability transport $\boldsymbol{\gamma^{(t)}} \in \mathbb{R}_{\geq 0}^{M\times N}$ where

$$\gamma_{(i,k)}^{(t)} = \begin{cases} 1/\rho_{j_k^i} \cdot \frac{1}{Ms^{(t)}}, & \text{if } k \leq K_i^{(t)}\\ \frac{1}{M} - \sum_{k=1}^{K_i^{(t)}} 1/\rho_{j_k^i} \cdot \frac{1}{Ms^{(t)}}, & \text{if } k = K_i^{(t)}+1\\ 0, & \text{otherwise}\end{cases}$$

Then we have

$$\begin{aligned}&\mathcal{L}(\boldsymbol{\gamma}^{(t-1)}) - \mathcal{L}(\boldsymbol{\gamma}^{(t)})\\ =&\underbrace{\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{N} d_{(i,k)}(\gamma_{(i,k)}^{(t-1)} - \gamma_{(i,k)}^{(t)})}_{T_5} + \underbrace{(1-\alpha)(G_{\text{KDE}}(\boldsymbol{\gamma}^{(t-1)}) - G_{\text{KDE}}(\boldsymbol{\gamma}^{(t)}))}_{T_6}\end{aligned}$$

By the definition of $K_i^{(t)}$ and $s^{(t)}$, either $K_i^{(t)} = K_i^{(t-1)}$ or $K_i^{(t)} = K_i^{(t-1)} + 1$. For any $i \in [M]$ such that $K_i^{(t)} = K_i^{(t-1)}$, we have $\gamma_{(i,k)}^{(t)} = 0$ for $k > K_i^{(t-1)} + 1$. For any $i \in [M]$ such that $K_i^{(t)} = K_i^{(t-1)} + 1$, we have $s^i_{K_i^{(t)}} = s^{(t)}$, in which case we also have $\gamma_{(i,k)}^{(t)} = 0$ for $k > K_i^{(t-1)} + 1$. Therefore, we have $\gamma^{(t-1)}_{(i,K_i^{(t-1)}+1)} - \gamma^{(t)}_{(i,K_i^{(t-1)}+1)} = -\sum_{k=1}^{K_i^{(t-1)}}(\gamma_{(i,k)}^{(t-1)} - \gamma_{(i,k)}^{(t)})$. Then it follows that

$$\begin{aligned}T_5 =&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{K_i^{(t-1)}}(d_{(i,k)} - d_{(i,K_i^{(t-1)}+1)})(\gamma_{(i,k)}^{(t-1)} - \gamma_{(i,k)}^{(t)})\\ =&\frac{\alpha}{C}\sum_{i=1}^{M}\sum_{k=1}^{K_i^{(t-1)}}(d_{(i,K_i^{(t-1)}+1)} - d_{(i,k)})/\rho_{j_k^i}\cdot\frac{1}{M}\cdot(\frac{1}{s^{(t)}} - \frac{1}{s^{(t-1)}})\end{aligned}$$

Let $\hat{s}^{(t)} = \min_{i\in[M]} s^i_{K_i^{(t-1)}+1}$, and then we have $T_5 = \frac{\alpha}{C}\cdot\frac{1}{M}\cdot(\frac{1}{s^{(t)}} - \frac{1}{s^{(t-1)}})c(\hat{s}^{(t)})$. Since $\hat{s}^{(t)} \le s^*$ and $c(s)$ is non-decreasing, we have $\frac{\alpha}{C}c(\hat{s}^{(t)}) \le \frac{\alpha}{C}c(s^*) < (1-\alpha)M$ and then it follows that $T_5 \ge (1-\alpha)(\frac{1}{s^{(t)}} - \frac{1}{s^{(t-1)}})$.

In addition, since $s^{(t-1)} < s^{(t)} \le s^*$, we have $\rho_{j_k^i}\gamma^{(t-1)} > \rho_{j_k^i}\gamma^{(t)} \ge \frac{1}{Ms^*} \ge \frac{2}{M\bar{s}}$, and further

$$\begin{aligned}T_6 =&(1-\alpha)M(\max_{i\in[M],k\in[N]}|\rho_{j_k^i}\gamma_{(i,k)}^{(t-1)} - \frac{1}{M\bar{s}}| - \max_{i\in[M],k\in[N]}|\rho_{j_k^i}\gamma_{(i,k)}^{(t)} - \frac{1}{M\bar{s}}|)\\ =&(1-\alpha)(\frac{1}{s^{(t-1)}} - \frac{1}{s^{(t)}})\end{aligned}$$

Therefore, we have $\mathcal{L}(\boldsymbol{\gamma}^{(t-1)}) - \mathcal{L}(\boldsymbol{\gamma}^{(t)}) = T_5 + T_6 \ge 0$. Since $\boldsymbol{\gamma'} \ge \boldsymbol{\gamma''} = \boldsymbol{\gamma}^{(1)} \ge \cdots \ge \boldsymbol{\gamma}^{(|\mathcal{S}'|)} = \boldsymbol{\gamma}^*$, we have $\boldsymbol{\gamma'} \ge \boldsymbol{\gamma}^*$.

If $\nexists s \in \mathcal{S}$ such that $\frac{\alpha}{C}c(s) = (1-\alpha)M$ and $\nexists i \in [M]$ such that $d_{(i,K_i)} = d_{(i,K_i+1)}$ or $d_{(i,K_i+1)} = d_{(i,K_i+2)}$, we have $T_1 > (1-\alpha)M(\frac{1}{Ms^*} - r')$ and $T_3 > 0$, and therefore $\boldsymbol{\gamma}' > \boldsymbol{\gamma}^*$, i.e., $\boldsymbol{\gamma}^*$ is the unique solution.

5.9 Conclusion

In this chapter, we proposed a holistic framework for task-specific data selection for ML model finetuning. We formulated target-specific data selection as an optimization problem based on optimal transport, where we transport probability mass from the representative example of the target task to candidate examples in the candidate pool. This formulation ensures the distribution alignment between the selected data and the target use cases. In addition, we added a regularization term to the optimization objective to encourage the diversity of selection. We incorporated kernel density estimation into the regularization term to make the selection robust to near-duplicates in the candidate pool. Experimentally we showed that our method is effective in both domain-specific continued pretraining and targeted instruction tuning.

6 CONCLUSION AND FUTURE DIRECTIONS

We conclude this dissertation by summarizing the main results. In addition, we discuss directions for future work.

6.1 Summary

The goal of this dissertation is to design efficient and effective methods for data validation and task-specific data selection to promote both the performance and robustness of ML applications. Toward this goal, we proposed new algorithms and frameworks that are not only of theoretical interest but also highly practical.

In the first part, we introduced new algorithms for denial constraint verification and discovery that achieve near-linear complexity relative to the dataset size, a theoretical improvement over prior works with quadratic complexity. We made the connection between denial constraint verification and orthogonal range search and utilized range-search data structures to efficiently verify the validity of denial constraints and enumerate the violations. Based on our verification algorithm, we further developed an anytime algorithm for denial constraint discovery, eliminating the reliance on the time-consuming blocking stage employed by previous works. Our experimental evaluation demonstrated that our denial constraint verification algorithm is faster than the current state-of-the-art method by an order of magnitude. In addition, our denial constraint discovery algorithm is able to output valid constraints in the early stage of execution, while the methods in prior works are blocked for an unreasonable amount of time.

In the second part, we proposed a framework to guard against data corruption in ML pipelines at both training and inference time. Our framework filters corrupted data at training time, and flags data whose prediction is misled by corruption at inference time. The key component of our framework is a new deep-learning model that can capture the distribution of mixed-type tabular data. The model is trained in a self-supervised way on potentially corrupted data without the need for manual annotation. We used the reconstruction loss from our model to identify corrupted training data and queries whose prediction will be wrong with a high probability due to corruption. In the experiments, we showed that our framework is effective in guarding against data corruption for both training and inference across different type of corruption, consistently outperforming the baselines.

In the last part, we presented a framework for task-specific data selection. Our framework formulates task-specific data selection as an optimization problem consisting of a probability transportation loss and a regularization term, allowing a smooth tradeoff between distribution alignment and diversity. In addition, we addressed the problem of near-duplicates by incorporating kernel density estimation into the regularization term. We proved that the optimal solutions of our framework can be connected to nearest neighbor search, and developed efficient algorithms to compute the solutions utilizing approximate nearest neighbor search techniques. In the empirical evaluation, we showed that our framework is effective in selecting data for both domain-specific continued pretraining and targeted instruction tuning.

6.2 Future Work

We point out potential directions for future work.

Denial Constraint Discovery and Verification over Dynamic Data The connection between denial constraint verification and orthogonal range search opens up an avenue to explore denial constraint discovery and verification in the setting of dynamic data, where the dataset is updated frequently. Dynamic data impose extra challenges on denial constraint verification and discovery since the validity of the constraints keeps changing as data gets updated. Luckily, there is a rich history of dynamic algorithms for both range trees and k-d trees, allowing the development of elegant algorithms with provable guarantees.

Interpretability of Learning-Based Data Validation Unlike rule-based data validation, learning-based data validation methods often output a single confidence score, which does not provide much information regarding why an example is considered low-quality. Revealing the cause of rejection would help the user understand the flaw in the data and take action. In addition, explainable data validation results are more trustworthy to humans. Although interpretable ML (Rudin et al., 2021) has been studied for years, it is unclear what kind of explanation is the most helpful to the user in the context of data validation.

Model-Specific Data Validation The majority of the data validation methods are model-agnostic, as they do not consider the downstream learning process. However, some works (Bian et al., 2023; Krishnan et al., 2016; Li et al., 2019) have shown that not all errors in the training data will cause degradation in ML performance. Model-specific data validation should identify data errors that a specific model is not robust to, which is critical to save sub-sequential cleaning efforts and ensure the maximum utility of the data. Existing works for model-specific data validation or cleaning (Bian et al., 2023; Krishnan et al., 2016) are limited to simple models such as Naïve Bayes classifiers and their results cannot be generalized to modern deep learning settings. A potential direction would be utilizing influence-based techniques (Ilyas et al., 2022; Koh and Liang, 2017) to approximate the effects of data errors on the learning outcomes.

Quality Standards for Text Data For tabular data, there are well-established formulations of constraints to ensure data quality. However, there is no such standard for the quality of text data, and consequently, researchers and practitioners rely on rules that are defined manually in an ad-hoc manner (Brown et al., 2020; Xie et al., 2023; Chen et al., 2024). As data quality is critical for modern language models, standard quality checks for text data are desired. To build such standards, the relationship between different aspects of text data and the learning performance needs to be carefully examined.

Theoretical Analysis on Task-Specific Selection and Finetuning

This dissertation empirically shows that distribution alignment and diversity are essential for task-specific finetuning, but a theoretical analysis of the relationship between the two factors and learning performance is missing in the current literature. It would be interesting to explore how distribution alignment and diversity contributes to the performance gain from finetuning, and develop theoretical results to guide the tradeoff between the two.

REFERENCES

2023. https://hpi.de/naumann/projects/repeatability/data-profiling/metanome-dc-algorithms.html. Accessed: 09/28/2023.

Abbas, Amro, Kushal Tirumala, Dániel Simig, Surya Ganguli, and Ari S. Morcos. 2023. Semdedup: Data-efficient learning at web-scale through semantic deduplication. 2303.09540.

Abedjan, Ziawasch, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9(12):993–1004.

Aharoni, Roee, and Yoav Goldberg. 2020. Unsupervised domain clusters in pretrained language models. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, ed. Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, 7747–7763. Online: Association for Computational Linguistics.

Arora, Sanjeev, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations.*

Bentley, Jon Louis, and Jerome H Friedman. 1979. Data structures for range searching. *ACM Computing Surveys (CSUR)* 11(4):397–409.

Bentley, Jon Louis, and James B Saxe. 1980. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms* 1(4): 301–358.

Bian, Song, Xiating Ouyang, Zhiwei Fan, and Paraschos Koutris. 2023. Certifiable robustness for naive bayes classifiers. 2303.04811.

Biggio, Battista, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. In *Proceedings of the 29th international coference on international conference on machine learning*, 1467–1474. ICML'12, Madison, WI, USA: Omnipress.

Bird, Steven, Ewan Klein, and Edward Loper. 2009. *Natural language processing with python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Bleifuß, Tobias, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proc. VLDB Endow.* 11(3):311–323.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5:135–146.

Bommasani, Rishi, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258.*

Breck, Eric, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data validation for machine learning. In *MLSys-19.*

Breck, Eric, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. 2019. Data validation for machine learning. In *Proceedings of sysml.*

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020.

Language models are few-shot learners. In *Advances in neural information processing systems*, ed. H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, vol. 33, 1877–1901. Curran Associates, Inc.

Calì, Andrea, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2002. Data integration under integrity constraints. In *Advanced information systems engineering: 14th international conference, caise 2002 toronto, canada, may 27–31, 2002 proceedings 14*, 262–279. Springer.

Carlini, Nicholas, and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, 39–57. IEEE.

Cer, Daniel, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder. `1803.11175`.

Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. `2107.03374`.

Chen, Ruibo, Yihan Wu, Lichang Chen, Guodong Liu, Qi He, Tianyi Xiong, Chenxi Liu, Junfeng Guo, and Heng Huang. 2024. Your vision-language model itself is a strong filter: Towards high-quality instruction tuning with data selection. `2402.12501`.

Chen, Yunqiang, Xiang Sean Zhou, and T.S. Huang. 2001. One-class svm for learning in image retrieval. In *Proceedings 2001 international conference on image processing (cat. no.01ch37205)*, vol. 1, 34–37 vol.1.

Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sashank Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2024. Palm: scaling language modeling with pathways. *J. Mach. Learn. Res.* 24(1).

Christen, Peter. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering* 24(9):1537–1555.

Chu, Xu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proc. VLDB Endow.* 6(13):1498–1509.

Clark, Jonathan H., Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020. TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics* 8:454–470.

Conover, Mike, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm.

Covington, Paul, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th acm conference on recommender systems*, 191–198. RecSys '16, New York, NY, USA: Association for Computing Machinery.

Dernoncourt, Franck, and Ji Young Lee. 2017. Pubmed 200k rct: a dataset for sequential sentence classification in medical abstracts. In *International joint conference on natural language processing*.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

———. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics.

Diakonikolas, Ilias, Gautam Kamath, Daniel Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. 2019a. Robust estimators in high-dimensions without the computational intractability. *SIAM Journal on Computing* 48(2): 742–864.

Diakonikolas, Ilias, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. 2019b. Sever: A robust meta-algorithm for stochastic optimization. In *Proceedings of the 36th international conference on machine learning*, ed. Kamalika Chaudhuri and Ruslan Salakhutdinov, vol. 97 of *Proceedings of Machine Learning Research*, 1596–1606. PMLR.

Diakonikolas, Ilias, Gautam Kamath, Daniel M Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. 2017. Being robust (in high dimensions) can be practical. In *Proceedings of the 34th international conference on machine learning-volume 70*, 999–1008. JMLR. org.

Diakonikolas, Ilias, and Daniel M. Kane. 2019. Recent advances in algorithmic high-dimensional robust statistics. `1911.05911`.

Dietterich, Thomas G. 1998. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation* 10(7): 1895–1923.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International conference on learning representations*.

Douze, Matthijs, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. `2401.08281`.

Du, Nan, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui

Wu, Zhifeng Chen, and Claire Cui. 2022. Glam: Efficient scaling of language models with mixture-of-experts. `2112.06905`.

Dua, Dheeru, and Casey Graff. 2017. UCI machine learning repository.

Eduardo, Simao, Alfredo Nazabal, Christopher K. I. Williams, and Charles Sutton. 2020. Robust variational autoencoders for outlier detection and repair of mixed-type data. In *Proceedings of the twenty third international conference on artificial intelligence and statistics*, ed. Silvia Chiappa and Roberto Calandra, vol. 108 of *Proceedings of Machine Learning Research*, 4056–4066. PMLR.

Elazar, Yanai, Akshita Bhagia, Ian Helgi Magnusson, Abhilasha Ravichander, Dustin Schwenk, Alane Suhr, Evan Pete Walsh, Dirk Groeneveld, Luca Soldaini, Sameer Singh, Hannaneh Hajishirzi, Noah A. Smith, and Jesse Dodge. 2024. What's in my big data? In *The twelfth international conference on learning representations*.

Engstrom, Logan, Axel Feldmann, and Aleksander Madry. 2024. Dsdm: Model-aware dataset selection with datamodels. `2401.12926`.

Fan, Wenfei, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel discrepancy detection and incremental detection. *Proceedings of the VLDB Endowment* 14(8):1351–1364.

Fariha, Anna, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. 2021. Conformance constraint discovery: Measuring trust in data-driven systems. In *SIGMOD '21: International conference on management of data, virtual event, china, june 20-25, 2021*, 499–512. ACM.

Feng, Yukun, Patrick Xia, Benjamin Van Durme, and João Sedoc. 2022. Automatic document selection for efficient encoder pretraining. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, ed. Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, 9522–9530. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics.

Fröbe, Maik, Janek Bevendorff, Lukas Gienapp, Michael Völske, Benno Stein, Martin Potthast, and Matthias Hagen. 2021. Copycat: Near-duplicates within and between the clueweb and the common crawl. In *Proceedings of the 44th international acm sigir conference on research and development in information retrieval*, 2398–2404. SIGIR '21, New York, NY, USA: Association for Computing Machinery.

Gálvez, Borja Rodríguez, German Bassi, Ragnar Thobaben, and Mikael Skoglund. 2021. Tighter expected generalization error bounds via wasserstein distance. In *Advances in neural information processing systems*, ed. A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan.

Gao, Leo, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Gao, Yue, Ilia Shumailov, and Kassem Fawaz. 2022a. Rethinking image-scaling attacks: The interplay between vulnerabilities in machine learning systems. In *Proceedings of the 39th international conference on machine learning*, vol. 162 of *Proceedings of Machine Learning Research*, 7102–7121. Baltimore, MI: PMLR.

Gao, Yue, Ilia Shumailov, Kassem Fawaz, and Nicolas Papernot. 2022b. On the limitations of stochastic pre-processing defenses. In *Proceedings of the 36th conference on neural information processing systems*.

Ge, Chang, Shubhankar Mohapatra, Xi He, and Ihab F. Ilyas. 2021. Kamino: Constraint-aware differentially private data synthesis. *Proc. VLDB Endow.* 14(10):1886–1899.

Geerts, Floris, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with llunatic. *The VLDB Journal* 29:867–892.

Giannakopoulou, Stella, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning denial constraint violations through relaxation. In *Proceedings of the 2020 acm sigmod international conference on management of data*, 805–815.

Gionis, Aristides, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, vol. 99, 518–529.

Goodfellow, Ian J, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *CoRR* abs/1412.6572.

Grosse, Kathrin, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*.

Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th international conference on machine learning-volume 70*, 1321–1330. JMLR. org.

Gururangan, Suchin, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, ed. Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, 8342–8360. Online: Association for Computational Linguistics.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 ieee conference on computer vision and pattern recognition (cvpr)*, 770–778.

Heidari, Alireza, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 international conference on management of data*, 829–846.

Hendrycks, Dan, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International conference on learning representations*.

Hernandez, Danny, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, Scott Johnston, Ben Mann, Chris Olah, Catherine Olsson, Dario Amodei, Nicholas Joseph, Jared Kaplan, and Sam McCandlish. 2022. Scaling laws and interpretability of learning from repeated data. `2205.10487`.

Herskovits, Edward. 1992. Computer-based probabilistic-network construction. Ph.D. thesis, Stanford, CA, USA. UMI Order No. GAX92-05646.

Hu, Shengyuan, Tao Yu, Chuan Guo, Wei-Lun Chao, and Kilian Q Weinberger. 2019. A new defense against adversarial images: Turning a weakness into a strength. In *Advances in neural information processing systems*, 1633–1644.

Huhtala, Yka, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal* 42(2):100–111.

Ibaraki, Toshihide, Alexander Kogan, and Kazuhisa Makino. 1999. Functional dependencies in horn theories. *Artificial Intelligence* 108(1-2):1–30.

Ilyas, Andrew, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. 2022. Datamodels: Understanding predictions with data and data with predictions. In *Proceedings of the 39th international* *conference on machine learning*, ed. Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, vol. 162 of *Proceedings of Machine Learning Research*, 9525–9587. PMLR.

Jiang, Albert Q., Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. `2310.06825`.

Jumper, John, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andrew J Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596(7873):583–589.

Karlaš, Bojan, 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. *Proc. VLDB Endow.* 14(3):255–267.

Kingma, Diederik P., and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings*, ed. Yoshua Bengio and Yann LeCun.

Koh, Pang Wei, and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th international conference on machine learning - volume 70*, 1885–1894. ICML'17, JMLR.org.

Koh, Pang Wei, Jacob Steinhardt, and Percy Liang. 2018. Stronger data poisoning attacks break data sanitization defenses. *Machine Learning* 111: 1 – 47.

Kossmann, Jan, Thorsten Papenbrock, and Felix Naumann. 2022. Data dependencies for query optimization: a survey. *The VLDB Journal* 31(1): 1–22.

Kringelum, Jens, Sonny Kim Kjaerulff, Søren Brunak, Ole Lund, Tudor I Oprea, and Olivier Taboureau. 2016. Chemprot-3.0: a global chemical biology diseases mapping. *Database* 2016:bav123.

Krishnan, Sanjay, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. Activeclean: interactive data cleaning for statistical modeling. *Proc. VLDB Endow.* 9(12):948–959.

Köpf, Andreas, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. 2023. Openassistant conversations – democratizing large language model alignment. `2304.07327`.

Lan, Zhenzhong, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR* abs/1909.11942. `1909.11942`.

Laurençon, Hugo, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Villanova del Moral, Teven Le Scao, Leandro Von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, Jörg Frohberg, Mario Šaško, Quentin Lhoest, Angelina McMillan-Major, Gérard Dupont, Stella Biderman, Anna Rogers, Loubna Ben allal, Francesco De Toni, Giada Pistilli, Olivier Nguyen, Somaieh Nikpoor, Maraim Masoud, Pierre Colombo, Javier de la Rosa, Paulo Villegas, Tristan Thrush, Shayne Longpre, Sebastian Nagel, Leon Weber, Manuel Romero Muñoz, Jian Zhu, Daniel Van Strien, Zaid Alyafeai, Khalid Almubarak, Vu Minh Chien, Itziar Gonzalez-Dios, Aitor Soroa, Kyle Lo, Manan Dey, Pedro Ortiz Suarez, Aaron Gokaslan, Shamik Bose, David Ifeoluwa Adelani, Long Phan, Hieu Tran, Ian Yu, Suhas Pai, Jenny Chim, Violette Lepercq, Suzana Ilic, Margaret Mitchell, Sasha Luccioni, and Yacine Jernite. 2022. The bigscience ROOTS corpus: A 1.6TB composite multilingual dataset. In *Thirty-sixth conference on neural information processing systems datasets and benchmarks track*.

Lee, Katherine, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: Long papers)*, 8424–8445. Dublin, Ireland: Association for Computational Linguistics.

Li, Peng, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. Cleanml: A benchmark for joint data cleaning and machine learning [experiments and analysis]. *arXiv preprint arXiv:1904.09483*.

Li, Yudong, Shigeng Zhang, Weiping Wang, and Hong Song. 2023. Backdoor attacks to deep learning models and countermeasures: A survey. *IEEE Open Journal of the Computer Society* 4:134–146.

Liu, Fang, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, and Li Zhang. 2024. Exploring and evaluating hallucinations in llm-powered code generation. `2404.00971`.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*, 413–422.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR* abs/1907.11692. 1907.11692.

Longpre, Shayne, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Luan, Yi, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. 2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, ed. Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, 3219–3232. Brussels, Belgium: Association for Computational Linguistics.

Maas, Andrew L., Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, ed. Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, 142–150. Portland, Oregon, USA: Association for Computational Linguistics.

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018a. Towards deep learning models resistant to adversarial attacks. In *International conference on learning representations*.

———. 2018b. Towards deep learning models resistant to adversarial attacks. In *International conference on learning representations*.

Mahdavi, Mohammad, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 international conference on management of data*, 865–882. SIGMOD '19, New York, NY, USA: Association for Computing Machinery.

McAuley, Julian J., Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. *CoRR* abs/1506.04757. `1506.04757`.

Moore, Robert C., and William Lewis. 2010. Intelligent selection of language model training data. In *Proceedings of the acl 2010 conference short papers*, 220–224. ACLShort '10, USA: Association for Computational Linguistics.

Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 2574–2582.

Mosbach, Marius, Maksym Andriushchenko, and Dietrich Klakow. 2021. On the stability of fine-tuning {bert}: Misconceptions, explanations, and strong baselines. In *International conference on learning representations.*

Muñoz-González, Luis, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th acm workshop on artificial intelligence and security*, 27–38.

Murphy, Kevin P. 2012. *Machine learning: a probabilistic perspective.* MIT press.

Nicolae, Maria-Irina, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. 2018. Adversarial robustness toolbox v1.2.0. *CoRR* 1807.01069.

OpenAI. 2023. Gpt-4 technical report. `2303.08774`.

Overmars, Mark H. 1983. *The design of dynamic data structures*, vol. 156. Springer Science & Business Media.

Pang, Tianyu, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. 2020. Rethinking softmax cross-entropy loss for adversarial robustness. In *International conference on learning representations*.

Pang, Tianyu, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. 2019. Improving adversarial robustness via promoting ensemble diversity. *arXiv preprint arXiv:1901.08846*.

Parzen, Emanuel. 1962. On estimation of a probability density function and mode. *The annals of mathematical statistics* 33(3):1065–1076.

Pena, Eduardo H. M., Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of approximate (and exact) denial constraints. *Proc. VLDB Endow.* 13(3):266–278.

Pena, Eduardo HM, Eduardo C de Almeida, and Felix Naumann. 2021. Fast detection of denial constraint violations. *Proceedings of the VLDB Endowment* 15(4):859–871.

Pena, Eduardo HM, Edson R Lucas Filho, Eduardo C de Almeida, and Felix Naumann. 2020. Efficient detection of data dependency violations. In *Proceedings of the 29th acm international conference on information & knowledge management*, 1235–1244.

Pena, Eduardo HM, Fabio Porto, and Felix Naumann. 2022. Fast algorithms for denial constraint discovery. *Proceedings of the VLDB Endowment* 16(4): 684–696.

Polyzotis, Neoklis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. 2019. Data validation for machine learning. *Proceedings of machine learning and systems* 1:334–347.

Rae, Jack W., Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2022. Scaling language models: Methods, analysis & insights from training gopher. `2112.11446`.

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21(1).

Rekatsinas, Theodoros, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB*

Endow. 10(11):1190–1201.

Rodríguez Gálvez, Borja, Germán Bassi, Ragnar Thobaben, and Mikael Skoglund. 2021. Tighter expected generalization error bounds via wasserstein distance. *Advances in Neural Information Processing Systems* 34: 19109–19121.

Roth, Kevin, Yannic Kilcher, and Thomas Hofmann. 2019. The odds are odd: A statistical test for detecting adversarial examples. In *Proceedings of the 36th international conference on machine learning, ICML 2019, 9-15 june 2019, long beach, california, USA*, ed. Kamalika Chaudhuri and Ruslan Salakhutdinov, vol. 97 of *Proceedings of Machine Learning Research*, 5498–5507. PMLR.

Ruder, Sebastian, and Barbara Plank. 2017a. Learning to select data for transfer learning with Bayesian optimization. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, ed. Martha Palmer, Rebecca Hwa, and Sebastian Riedel, 372–382. Copenhagen, Denmark: Association for Computational Linguistics.

———. 2017b. Learning to select data for transfer learning with Bayesian optimization. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, 372–382. Copenhagen, Denmark: Association for Computational Linguistics.

Rudin, Cynthia, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. 2021. Interpretable machine learning: Fundamental principles and 10 grand challenges. `2103.11251`.

Schelter, Sebastian, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *Proc. VLDB Endow.* 11(12):1781–1794.

Shachaf, Gal, Alon Brutzkus, and Amir Globerson. 2021. A theoretical analysis of fine-tuning with linear teachers. In *Advances in neural information processing systems*, ed. M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, vol. 34, 15382–15394. Curran Associates, Inc.

Simonyan, Karen, and Andrew Zisserman. 2014. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, 568–576.

Steinhardt, Jacob, Pang Wei Koh, and Percy Liang. 2017a. Certified defenses for data poisoning attacks. In *Proceedings of the 31st international conference on neural information processing systems*, 3520–3532. NIPS'17, Red Hook, NY, USA: Curran Associates Inc.

Steinhardt, Jacob, Pang Wei W Koh, and Percy S Liang. 2017b. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*, 3517–3529.

Su, Weijie, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2020. Vl-bert: Pre-training of generic visual-linguistic representations. In *International conference on learning representations*.

Suzgun, Mirac, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Tirumala, Kushal, Daniel Simig, Armen Aghajanyan, and Ari S. Morcos. 2023. D4: Improving llm pretraining via document de-duplication and diversification. `2308.12284`.

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric

Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. `2302.13971`.

Tukey, John W. 1975. Mathematics and the picturing of data. In *Proceedings of the international congress of mathematicians, vancouver, 1975*, vol. 2, 523–531.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, ed. I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, vol. 30. Curran Associates, Inc.

Villani, Cédric. 2009. *Optimal transport: old and new*, vol. 338. Springer.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems*, ed. Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho.

Wenzek, Guillaume, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the twelfth language resources and evaluation conference*, ed. Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, 4003–4012. Marseille, France: European Language Resources Association.

Wu, Richard, Aoqian Zhang, Ihab F Ilyas, and Theodoros Rekatsinas. 2020. Attention-based learning for missing data imputation in holoclean. *Proceedings of Machine Learning and Systems* 307–325.

Xia, Mengzhou, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. Less: Selecting influential data for instruction tuning.

Xiao, Chang, Peilin Zhong, and Changxi Zheng. 2019. Resisting adversarial attacks by k-winners-take-all. *arXiv preprint arXiv:1905.10510*.

Xiao, Renjie, Zijing Tan, Haojin Wang, and Shuai Ma. 2022. Fast approximate denial constraint discovery. *Proc. VLDB Endow.* 16(2):269–281.

Xie, Sang Michael, Shibani Santurkar, Tengyu Ma, and Percy Liang. 2023. Data selection for language models via importance resampling. In *Thirty-seventh conference on neural information processing systems*.

Xue, Zhenxia, Youlin Shang, and Aifen Feng. 2010. Semi-supervised outlier detection based on fuzzy rough c-means clustering. *Mathematics and Computers in simulation* 80(9):1911–1921.

Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, 5754–5764.

Yao, Xingcheng, Yanan Zheng, Xiaocong Yang, and Zhilin Yang. 2021. NLP from scratch without large-scale pretraining: A simple and efficient framework. *CoRR* abs/2111.04130. `2111.04130`.

———. 2022. NLP from scratch without large-scale pretraining: A simple and efficient framework. In *Proceedings of the 39th international conference on machine learning*, ed. Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, vol. 162 of *Proceedings of Machine Learning Research*, 25438–25451. PMLR.

Zha, Daochen, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, and Xia Hu. 2023. Data-centric ai: Perspectives and challenges. In *Sdm*.

Zhang, Xiang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *CoRR* abs/1509.01626. `1509.01626`.

Zhang, Yuhao, Aws Albarghouthi, and Loris D'Antoni. 2020a. Robustness to programmable string transformations via augmented abstract training. `2002.09579`.

Zhang, Yuhao, Aws Albarghouthi, and Loris D'Antoni. 2021. Certified robustness to programmable transformations in LSTMs. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, ed. Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, 1068–1083. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Zhang, Yunjia, Zhihan Guo, and Theodoros Rekatsinas. 2020b. A statistical perspective on discovering functional dependencies in noisy data. In *Proceedings of the 2020 acm sigmod international conference on management of data*, 861–876. SIGMOD '20, New York, NY, USA: Association for Computing Machinery.

Zhou, Guorui, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining*, 1059–1068. KDD '18, New York, NY, USA: Association for Computing Machinery.

Zilberstein, Shlomo. 1996. Using anytime algorithms in intelligent systems. *AI magazine* 17(3):73–73.