Data Systems for Managing and Debugging Machine Learning Workflows

DISS. ETH NO. 28449

DATA SYSTEMS FOR MANAGING AND DEBUGGING MACHINE LEARNING WORKFLOWS

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

BOJAN KARLAŠ
Master of Science in Computer Science, EPFL

born on 23.01.1990.

accepted on the recommendation of

Prof. Dr. Ce Zhang (ETH Zurich), examiner
Prof. Dr. Gustavo Alonso (ETH Zurich), co-examiner
Dr. Matteo Interlandi (Microsoft), co-examiner
Prof. Dr. Sebastian Schelter (University of Amsterdam), co-examiner

2022

ABSTRACT

As machine learning continues becoming more ubiquitous in various areas of our lives, it will become impossible to imagine software development projects that do not involve some learned component. Consequently, we have an ever increasing number of people developing ML applications, which drives the need for better development tools and processes. Unfortunately, even though there has been tremendous effort spent in building various systems for machine learning, the development experience is still far from what regular software engineers enjoy. This is mainly because the current ML tooling is very much focused on solving specific problems and cover only a part of the development workflow. Furthermore, end-to-end integration of these various tools is still quite limited. This very often leaves the developers stuck without guidance as they try to make their way through a labyrinth of possible choices that could be made at each step.

This thesis aims to tackle the usability problem of modern machine learning systems. This involves taking a broader view which goes beyond the model training part of the ML workflow and developing a system for managing this workflow. This broader workflow includes the data preparation process which comes before model training, as well model management which comes after. We seek to identify various pitfalls and pain points that developers encounter in these ML workflows. We then zoom into one particular kind of a usability pain point – labor-efficient data debugging. We pinpoint two categories of data errors (missing data and wrong data), and develop two methods for guiding the attention of the developer by helping them choose the instances of data errors that are the most important. We then empirically evaluate those methods in realistic data repair scenarios and demonstrate that they indeed improve the efficiency of the data debugging process, which in turn translates to greater usability. We finish up with some insights which could be applied to design more usable machine learning systems in the future.

ZUSAMMENFASSUNG

Da maschinelles Lernen (ML) in verschiedenen Bereichen unseres Lebens immer allgegenwärtiger wird, sind Softwareentwicklungsprojekte ohne erlernte Komponenten undenkbar. Folglich haben wir eine ständig wachsende Zahl von Menschen, die ML-Anwendungen entwickeln, was den Bedarf an besseren Entwicklungswerkzeugen und -prozessen antreibt. Obwohl enorme Anstrengungen unternommen wurden, um verschiedene Systeme für maschinelles Lernen zu entwickeln, ist der Entwicklungsprozess leider immer noch weit von dem entfernt, was Softwareentwickler normalerweise erfahren. Dies liegt hauptsächlich daran, dass die aktuellen ML-Tools sehr stark auf die Lösung spezifischer Probleme ausgerichtet sind und nur einen Teil des Entwicklungsworkflows abdecken. Darüber hinaus ist die End-to-End-Integration dieser verschiedenen Tools noch begrenzt. Dies lässt die Entwickler sehr oft ohne Anleitung stecken, wenn sie versuchen, sich ihren Weg durch ein Labyrinth möglicher Entscheidungen zu bahnen, die bei jedem Schritt getroffen werden können.

Diese Arbeit zielt darauf ab, das Usability-Problem moderner maschineller Lernsysteme anzugehen. Dies beinhaltet eine breitere Sichtweise, die über den Modellschulungsteil des ML-Workflows hinausgeht, und die Entwicklung eines Systems zur Verwaltung dieses Workflows. Dieser umfassendere Arbeitsablauf umfasst den Datenvorbereitungsprozess, der vor dem Modelltraining stattfindet, sowie das Modellmanagement, das danach folgt. Wir versuchen, verschiedene Sackgassen zu identifizieren, auf die Entwickler in diesen ML-Workflows stoßen. Dann befassen wir uns einem bestimmten Usability-Problem – dem arbeitseffizienten Debuggen von Daten. Wir lokalisieren zwei Kategorien von Datenfehlern (fehlende Daten und falsche Daten) und entwickeln zwei Methoden, um den Entwickler dabei zu unterstützen, die wichtigsten Fälle von Datenfehlern zu erkennen. Anschließend evaluieren wir diese Methoden empirisch in realistischen Datenreparaturszenarien und zeigen, dass sie tatsächlich die Effizienz des Daten-Debugging-Prozesses verbessern, was wiederum zu einer größeren Benutzerfreundlichkeit führt. Wir schließen mit einigen Erkenntnissen ab, die angewendet werden könnten, um in Zukunft besser nutzbare Systeme für maschinelles Lernen zu entwerfen.

Мами, тати и брату.

ACKNOWLEDGEMENTS

I owe my endless gratitude to my advisor Ce – a true mentor who made an immeasurable impact on my academic development. The honest and unfiltered feedback he has been generously offering me was indispensable in shaping both my research skills and my character, which enabled me to do the work that I could one day be proud of. I am deeply grateful for his infinite patience and countless hours spent sharing his knowledge and giving me much-needed guidance. The trust we were fortunate to develop over the years was foundational in making all that time truly meaningful and I would gladly do it all over again. Ce is a genuinely passionate researcher who is not afraid to dream big and I will continue to be inspired by him throughout my life.

The time of my Ph.D. studies would not be nearly as enjoyable if it wasn't for my dear research group – the DS3Lab. It was remarkable to be a part of a group all the way from its early days with so many different characters and grow together with so many of them. I want to give special thanks to the first DS3-wave with whom I spent the most time – Merve for her kindness and for always being there to offer a supportive hand, Johannes and David for all our exciting off-road adventures, Luka for always trying to put a smile on my face, Cedric and Nora for their inspiring discipline and the unspoken understanding we shared, Susie and Zhikuan for the quirky yet sincere conversations we had, Leo for all the weirdly fun moments, Maurice and Thomas for all the light-hearted laughs, Shaoduo and Jiawei for their gentle humility and coolness, and of course, all of the people who joined the group more recently but are nevertheless an integral part of our academic family. I will always remember the fun moments we shared, both in Zurich, but also in various other places around the world!

During my Ph.D. studies, I was fortunate to have many collaborators to work with and learn from. Among them are some to whom I would like to give special thanks – Wentao for his support through the entirety of my Ph.D. studies and his generous help in honing my writing skills, Matteo for patiently mentoring me during my Microsoft internship and offering many insights on databases and ML systems,

Xu and Peng for the work we did on CPClean which involved solving many tricky problems that their devotion turned into a breeze, Sebastian for the thoughtful guidance and systems know-how that he offered during our work on the DataScope project. I was also fortunate to interact with many professors at ETH. I would like to give special thanks to some of them I had closer contact with – Ankit, Gustavo, Markus, Ana and Mothy, who have shown me nothing but kindness as well as some truly insightful conversations. Extra special thanks go to Matteo, Sebastian, and Gustavo for being part of my defense committee, for the feedback they offered, and for the discussions we had. Finally, even though he was one of my mentors during my Master's at EPFL, I cannot avoid mentioning Bernard. He not only offered me generous support throughout the time I spent in his lab, but he is also the principal source for most of the algorithms knowledge I acquired which was instrumental in solving many of the problems presented in this thesis.

When mentioning the things that influenced my Ph.D. experience it is impossible not to mention my engagement with Modulos which offered me a window into another world. Among the wonderful team that was assembled over the years, I would like to give special thanks to Kevin and Dennis who were the first ones I met back at the Physics department during the time of my Master thesis, as well as Anna and Romain, my very supportive managers with whom I had endless technical and non-technical conversations. Also, during my studies, I deeply enjoyed the gratifying experience of mentoring many students who helped me in gaining many different points of view – Mohammed, Livio, Quentin, Tobias, Filip, and Xiaozhong.

Last, but certainly not least, I would like to thank many of my close friends and family who played a huge role in my life and shaped the person I am today. Some of them are new friends I met during my time in Switzerland – Ece, Georgios, Beliz, Milica, Stanko, Vojislav, Lazar, Camilo, Carmelo, and Monica. Some are my dear old friends I know from Serbia – Nemanja, Milica, Đorđe, Dušan, Veljko, Vukašin, Sonja, Nora, Jovana, Anđela, and Miroslav. Finally, my family was an insatiable source of support – especially my aunts Jasmina and Milanka; my uncles Robert and Mirko; my cousins Sara, Jelena, Anja, and Aleksa; my maternal grandparents Marija, Bogdan; my late paternal grandparents Danica and Neđo; and, of course, Gordana, Milanko and Nikola, my mother, father and brother who represent the bedrock of my life. The importance of these people can hardly be put into words but I am sure each one of them is well aware of it. Hvala!

CONTENTS

1

INTRODUCTION

It is no secret that AI applications involving various machine learning (ML) techniques are getting ever-more prevalent. New breakthroughs that are made in the core machine learning field are closely followed by a wave of systems and tools designed to support various aspects of the ML development process. This growing toolbox includes systems for data acquisition, preparation and refinement, as well as for model training, tuning and management. Figure 1.1 gives us a compressed snapshot of this space taken in 2021, which, as we can see is quite rich.

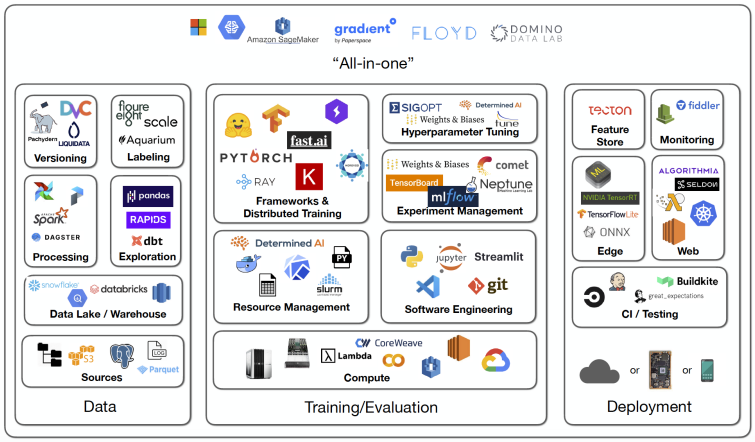

FIGURE 1.1: A snapshot overview of the space of ML tooling taken from [1].

A side effect the growing richness of this space is that the ML development process itself is becoming ever more complex. So much so that we are now training armies of professionals whose main focus is to do this tedious yet important job. The key challenge they routinely face, apart from having to learn how to use many of these tools, has to do with the *overhelming amount of choices* that comes with navigating this space. This severly impacts the usability of this ecosystem as a whole and results in a lot of work hours wasted. There are various questions that

ML developers come across: Which model should we train using our data? If our trained model is not performing well, should we acquire more data or should we repair the data we already have? If we should repair our data, which specific data items should we focus on first? Each of these questions often implies a relatively large number of possible choices. Furthermore, making the right choice is usually highly context-dependant. That is, the right choice always depends on things like the dataset we are using, the goal we are trying to achieve, etc. Therefore, there are essentially no universal answers, which makes the situation additionally challenging.

The goal of this thesis is to find systematic and principled approaches for answering such questions in an effort to improve the usability of ML systems. We start off by trying to identify specific pain points that ML practitioners encounter during the development process. These pain points are essentially places where developers might ask themselves "What should I do now?" They represent opportunities for some hypothetically novel system to offer a signal of guidance. In this thesis we will focus on one particular kind of signal which we refer to as a *measure of importance*.

This forms the basis of ease.ml, our proposed system for managing ML workflows. It is constructed as a set of loosly connected system components, where each one targets some particular pain point by guiding the decision making process of the user. In this thesis we will present the overall system and describe in more detail two specific components used for the problem of data debugging – ease.ml/cpclean and ease.ml/datascope.

Our approach when developing the system components presented in this thesis begins by coming up with some potentially useful measure of importance. We try to formulate these using some theoretically well-founded principles. Since the measures often depend on the current context which must be taken into account, obtaining these measures usually involves some data processing. This can come with significant computational complexity which poses an additional challenge. In our work, we try to explore various tradeoffs and assumptions in order to develop novel algorithms for efficiently computing importance. Finally, we empirically evaluate how effective these importance signals are at performing concrete tasks. We construct experiments based on tasks which are representative of real-world ML development scenarios. Our findings can be summarized in the following statement:

Thesis Statement: *Usability of machine learning systems can be improved by integrating guidance from theoretically well-founded, efficiently computable and practically effective measures of importance.*

In support of this thesis, we make the following technical contributions:

- **(C1) `ease.ml` workflow management system.**
  We present ease.ml, a lifecycle management system for machine learning (ML). Unlike many existing works, which focus on improving individual steps during the machine learning workflow, ease.ml focuses on managing and automating the entire lifecycle itself. We present user scenarios that have motivated the development of ease.ml, the steps of the ease.ml process that covers the machine learning workflow; some design choices we have made; and in the conclusion of this thesis review some lessons we learned, which hopefully can inspire future research.

- **(C2) `ease.ml/cpclean` for optimizing the repair of missing values.**
  We propose certain predictions (CP), as well as its two fundamental queries/primitives (checking and counting), as a tool to study the impact of incomplete data on training ML models. We propose efficient solutions to the two fundamental CP queries for K-nearest-neighbor (KNN) classifiers, despite the hardness of these two queries in general. We propose a novel data cleaning approach, ease.ml/cpclean, which uses *information gain* as a measure of importance which is built on top of the CP primitives. We perform empirical studies which show that ease.ml/cpclean significantly outperforms existing work, particularly on datasets with systematic missing values.

- **(C3) `ease.ml/datascope` for optimizing the repair of wrong values.**
  We present ease.ml/datascope, the first system that efficiently computes Shapley values of training examples over an end-to-end ML pipeline, and illustrate its applications in data debugging for ML training. To this end, we first develop a novel algorithmic framework that computes Shapley value over a specific family of ML pipelines that we call *canonical pipelines* – a positive relational algebra query followed by a K-nearest-neighbor (KNN) classifier. We show that, for many subfamilies of canonical pipelines, computing Shapley value is possible in time that is polynomial in the size of data, contrasting the exponential complexity of computing Shapley value in general. We then put this to practice – given an ML pipeline, we approximate it with a canonical pipeline to use as a proxy. We conduct extensive experiments illustrating different use cases and utilities. Our results show that ease.ml/datascope is up to four orders of magnitude faster over state-of-the-art Monte Carlo-based methods, while being comparably, and often even more, effective in data debugging.

1.1 Methodology

We briefly discuss the general approaches we took in this work and provide some further background on the problems we were trying to tackle. Specifically, we will discuss the main aspects of the modern "data-centric" machine learning workflow and reveal some challenges that come with it. Then we showcase how our notion of importance can be defined by giving examples of specific measures that we use in this thesis. Finally, we discuss how such measures of importance can be integrated into a system that guides decision making of the user.

1.1.1 From Model Centric to Data-Centric ML Workflows

For some time, the view on what the machine learning development workflow should look like was characterized by what is now referred to as a "model-centric" view. Namely, the majority of effort both from the research community and practitioners was spent developing and training better models with more efficient implementations on faster hardware. Among other things, this view makes two key assumptions: (1) that training data is static; and (2) that our job is done once we have a trained model that performs well on our test set.

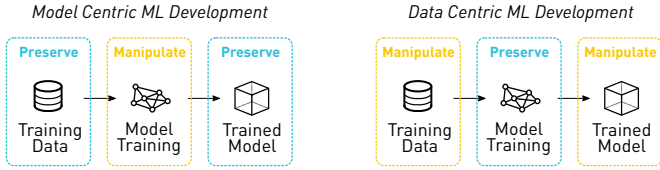

FIGURE 1.2: Based on our view, the shift from model centric to data-centric ML develpoent is about shifting the focus from model development towards data development and data-driven model management.

As signaled by some of the leading figures in the machine learning field [2], we are experiencing a fundamental shift in perspective towards a more "data-centric" view. This shift is depicted in Figure 1.2 where the aspects each view focuses on is marked with a yellow box. This view is more or less based on challenging the above mentioned assumptions, and driven by observations made from real-world

AI development. Even though model training is still very important, following the advent of Auto-ML systems, it requires less and less human effort. We are starting to see much more effort spent on *data preparation* which aims to produce high-quality training data, as well as *model management* which channels trained models towards deployment. Since both of these processes are usually conducted in a never-ending iterative fashion, we refer to them as the *"data cycle"* and the *"model cycle"*. Both of these cycles are centered around data, but they take slightly different paths. We explore this further in Chapter 2.

INTEGRATING EXISTING ML SYSTEMS. Even though the formulation of these data-centric viewpoints is relatively recent, it is based on a lot of pioneering research effort made in the past by the machine learning, systems and data management communities. Many concrete problems have been identified and solutions have been offered. Some of these problems fit in the data cycle and some in the model cycle. Examples of data cycle problems include: repairing data errors with data cleaning, discarding unnecessary training data, explaining model predictions using training data examples, etc. On the model cycle side we have: continuous testing of ML models, overcoming data drifting, etc.

There are many tools and methods out there for solving a lot of specific data-centric problems. However, one significant challenge that remains is the difficulty of navigating this broader space of tools and methods. The main goal when designing ease.ml was to tackle exactly this challenge. The general question was – *How should we organize an ML workflow management system to maximize its usability?*

Our approach was to identify some pain points that lie somewhere in-between existing tools and, at the same time, require the user to make *informed decisions* about what step to perform next. This requirement for informed decision making represents a key obstacle for users and reduces the usability of the entire workflow. The approach we take is to introduce some well-founded, computable and practically useful measures of importance which can offer our users the guidence they need to make better informed decisions.

1.1.2 Guiding Decisions using Measures of Importance

This leads us to the question of which particular pain points can we identify in ML workflows and which measures of importance can be useful for resolving them. In Figure 1.3 we can see a typical workflow which begins with a training dataset that contains data errors and ends with various possible model quality metrics. In this thesis we focus specifically on the problem of *data debugging* in the context of some specific model quality metric. Namely, the question we are trying to answer is:

> *If we would like to improve the quality of our ML model by fixing issues with data quality, which specific parts of data shoud we prioritize?*

The guidance signal that we need in this situation is *data importance*, that is, the importance of individual data examples. Naturally, this importance depends on the context of the ML model training and evaluation loop. This means that we need to identify key elements of this loop and to find ways to include it into our importance computation.

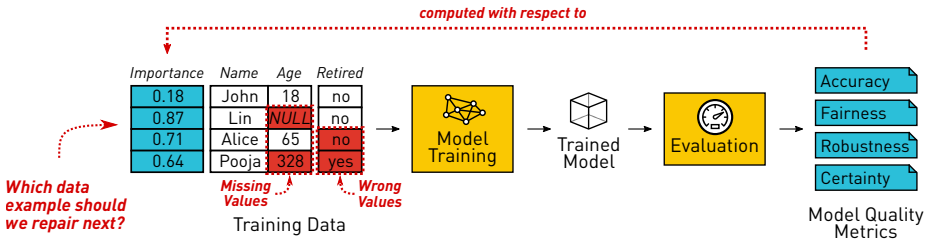

FIGURE 1.3: We distinguish between two types of data errors: missing values and wrong values. They lead us to two different measures of data importance.

When it comes to determining the quantity which would best serve as a principled measure of importance, in our view, the answer depends on the type of *data error* that we are dealing with. We identify two types of data errors:

- **Missing values**, or *incomplete information*, is a type of data error where e.g. a column of a table is not completely filled out for every single data example. This is usually encoded in the form of `NULL` values. Sometimes, a value may not be `NULL` but can still be directly detected by a simple validation function. For example in Figure 1.3 we see a person with age value 328 which

is obviously impossible and can be caught automatically, so we treat it as a missing value as well.

- **Wrong values**, are slightly more pernicious because they do not violate any constraint imposed by the schema and formatting of the dataset. An example that often comes up are wrongly labeled data examples. As a result, they are not easily detectable in an automated way. Often, the way to deal with them is either manually, or using some (semi-)automated data repair method which is usually quite expensive.

Different types of data errors constitute slightly different scenarios and manners of reasoning about what is the source of that data error. Depending on the type of data error, we have identified two types of importance metrics:

- **Information gain.** We take on the view that missing values are a source of *uncertainty*, both in the source training data as well as the predictions of a downstream ML model trained on that data. We can thus say that uncertain data is "confusing" the model, or, in the view of information theory, it is a source of *entropy*. This makes the repair process an act of *reduction of uncertainty*, or, in other words an act of *information gain*. Hence, we use information gain as a measure of importance and in this thesis we demonstrate its effectiveness. We describe this method in detail in Chapter 3.

- **Shapley value.** As mentioned, wrong values are slightly different. We are not able to replace wrong values with any candidate values because we are not able to detect them easily. Therefore, we can only take on the view that wrong values are "hurting" the model by their presence, and the act of repairing them is an act of "removing" the source of "pain". We have found a principled way of expressing this through the Shapley value, a quantity well established in game theory which represents a way of measuring the (positive or negative) contribution of a single player with respect to the group. This method is presented in Chapter 4.

In this thesis, we focus on these two measures of importance, study the computational challenges that they bring along, develop algorithmic frameworks to overcome them and evaluate how good they are at the actual task of optimizing the data repair process.

1.1.3 Integrating Guidance into ML Management Systems

After we develop and empirically evaluate any measure of importance (e.g. information gain, Shapley value, etc), the remaining question is – *How can we enable the user to take advantage of this measure of importance and overcome their particular pain point?*

Each pain point will have a slightly different answer to that question (more details in Chapter 2). As an example, let us examine the specific case of data debugging using data importance. If we look at Figure 1.3 we can notice that the importance values are computed for each tuple in the training dataset. To decide on which data tuples we should repair first, we would simply need to sort the dataset according to these importance values and then apply data repairs either one tuple at a time or in batches.

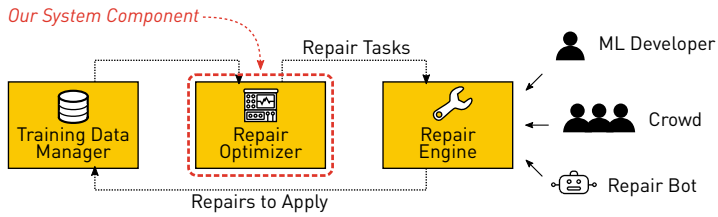

FIGURE 1.4: We introduce a *repair optimizer* system component that analyzes training data and transmits a stream of repair tasks to the *repair interface* which handles the actual repairs. The repairs are either performed by the ML developer directly, crowdsourced or performed automatically by some repair bot.

In Figure 1.4 we can see an example of a processing pipeline for data repairs. How the actual repairs are performed depends on the actual method that the user is taking. We assume they will leverage some of the established methods or tools. The most difficult data repair scenarios usually rely on manual repairing. This is typically performed by one or more human labelers who posseses sufficient domain knowledge about the target dataset. This could be done through some tool with a user interface for repairing data such as Labelbox [3]. We could also imagine some automated labeling engines being provided. They could either be custom-built and domain-specific tools, or some more general tools like HoloClean [4].

In any of those cases, we can assume the existence of a system component which we will refer to as a *repair engine* that handles *repair tasks* (either as a stream or in batches). If this engine is an interactive manual repair interface, the repair tasks are presented to the users. If it is an automatic tool then it relies on its own algorithms for perfroming the repairs. The output of the engine is sent back to the data management system and applied as a data repair. The main thing to note is that we assume this component to be already available and the only question is how to improve its efficiency by minimizing the amount of work it needs to perform.

To accomplish this, we introduce our *repair optimizer* system component and integrate it into the pipeline for sending the stream of repair tasks to the repair engine. The goal of the system component is to send the tasks in order that will minimize the labeling cost while maximizing model quality. By integrating our importance measure into the process of sending these labeling tasks, we are directly improving the efficiency of the human or automated labeler, and, by extension, leaving a positive impact on the usability of their machine learning workflow.

1.2 Organization of this Thesis

This thesis is organized as follows. We first introduce a broader context of data-centric systems for managing machine learning workflows in Chapter 2. There we describe ease.ml, our proposed machine learning workflow management system. We go over an array of pain points that we were able to identify in our exprience with applied machine learning, along with an array of proposed system components intended to resolve those pain points. We also describe briefly some design decisions which we made when developing the ease.ml system.

We then zoom into the problem of importance based data debugging. As mentioned, in this thesis we explore two types of data debugging problems: missing data repair and wrong data repair. We explore the repairing of missing data in Chapter 3. We first develop a set of abstractions for expressing the notion of uncertainty in ML model predictions. Then we show how this set of abstractions can be applied to the problem of repairing missing data.

Moving on to the second type of data error, wrong values, in Chapter 4 we take a look at efficiently computing the Shapley value, which is the importance measure we use for optimizing wrong data repair. We focus on the additional challenge of

computing Shapley values over ML pipelines. We show how we can leverage the well established theory of data provenance to represent relevant information about datasets as they pass through machine learning pipelines.

We close off with Chapter 5, where we summarize the lessons learned throughout our research exprience and provide a short set of recipes for building future ML workflow management systems.

1.3 Author's Publications

This thesis is mainly based on the following publications:

[1] L. A. Melgar, D. Dao, S. Gan, N. M. Gürel, N. Hollenstein, J. Jiang, B. Karlaš, T. Lemmin, T. Li, Y. Li, S. Rao, J. Rausch, C. Renggli, L. Rimanic, M. Weber, S. Zhang, Z. Zhao, K. Schawinski, W. Wu, and C. Zhang, "Ease.ml: A lifecycle management system for mldev and mlops", in *Conference on Innovative Data Systems Research*, 2021.

[2] B. Karlaš, D. Dao, M. Interlandi, B. Li, S. Schelter, W. Wu, and C. Zhang, "Data debugging with shapley importance over end-to-end machine learning pipelines", *arXiv preprint arXiv:2204.11131*, 2022.

[3] B. Karlaš, P. Li, R. Wu, N. M. Gürel, X. Chu, W. Wu, and C. Zhang, "Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions", *Proceedings of the VLDB Endowment*, 2020.

Throughput my doctoral studies I worked on other research topics that are related but are not directly covered in this thesis. That work has resulted in the following publications:

[1] B. Karlaš, J. Liu, W. Wu, and C. Zhang, "Ease.ml in action: Towards multi-tenant declarative learning services", *Proceedings of the VLDB Endowment (Demo)*, 2018.

[2] C. Yu, B. Karlaš, J. Zhong, C. Zhang, and J. Liu, *Automl from service providers perspective: Multi-device, multi-tenant model selection with gp-ei*, 2019.

[3] C. Renggli, F. A. Hubis, B. Karlaš, K. Schawinski, W. Wu, and C. Zhang, "Ease.ml/ci and ease.ml/meter in action: Towards data management for

statistical generalization", *Proceedings of the VLDB Endowment (Demo)*, 2019.

[4] C. Renggli, B. Karlaš, B. Ding, F. Liu, K. Schawinski, W. Wu, and C. Zhang, "Continuous integration of machine learning models: A rigorous yet practical treatment", in *Proceedings of Machine Learning and Systems*, 2019.

[5] B. Karlaš, M. Interlandi, C. Renggli, W. Wu, C. Zhang, D. M. I. Babu, J. Edwards, C. Lauren, A. Xu, and M. Weimer, "Building continuous integration services for machine learning", in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

[6] M. R. Karimi, N. M. Gürel, B. Karlaš, J. Rausch, C. Zhang, and A. Krause, "Online active model selection for pre-trained classifiers", in *International Conference on Artificial Intelligence and Statistics*, 2021.

[7] C. Renggli, L. Rimanic, N. M. Gürel, B. Karlaš, W. Wu, and C. Zhang, *A data quality-driven view of mlops*, 2021.

[8] S. Schelter, S. Grafberger, S. Guha, O. Sprangers, B. Karlaš, and C. Zhang, "Screening native ml pipelines with arguseyes", in *Conference on Innovative Data Systems Research (Abstract)*, 2022.

2

DATA-CENTRIC SYSTEMS FOR MACHINE LEARNING WORKFLOW MANAGEMENT

We are witnessing a shift in the field of applied machine learning. As many successful systems for automating the process of training ML models managed to gain promenance, the community started to take a closer look at the broader context. Many problems started to get identified. The overall goal remains – enable building better ML applications. However, it became clear that achieving this goal involves more than simply trying to perfect the ML model training process. Instead, we need to focus on the broader ML workflow that starts from raw data sources all the way until models are deployed to production. In this chapter, we take a look at this broader space. We examine some problems and pain points that we have identified during our direct and indirect experience with applied ML. Finally, we present ease.ml, a proposed system that we designed to improve the usability of ML. This system is structured as a framework of loosly connected but composable system components that we designed to tackle the specific usability pitfalls that we identified.

2.1 Background and Related Work

The wide application of machine learning technologies has spawned intensive and extensive research on ML systems. The increasing complexity of ML systems, however, has perplexed many application developers, especially those who are domain experts, but do not have much of a background in statistics and/or computer science. Although many of these users are proficient DBMS users because of its fascinating usability and clean abstraction, they are struggling with modern ML systems. We believe that an emerging challenge of wider applications of ML techniques hinges on the *usability* of ML systems for such non-expert users.

Recently, there has been a flurry of work focused on ML usability. Researchers have developed systems that make it increasingly easy to handle different components of the ML development process. Examples include data acquisition with *weak supervision* (e.g., Snorkel [5], ZeroER [6]), *debugging and validation* (e.g., TFX [7], [8], Rain [9], Krypton [10]), *model management* (e.g., Cerebro [11]) and *deployment* (e.g., ModelDB [12], MLFlow [13]), *knowledge integration* (e.g., DeepDive [14]), *data cleaning* (e.g., HoloClean [4], ActiveClean [15]), and *interaction* (e.g., NorthStar [16]). Thanks to these efforts, today's users are well-equipped with many powerful tools for different stages of ML application development. However, there are still challenges in ML usability. One hypothesis (and observation) we have is that

> *One emerging barrier hindering the usability of ML systems is not the limited availability, capacity, and performance of existing ML tools, but rather the complexity for a non-expert user to navigate a collection of ever-updating, overwhelmingly powerful tools.*

Formulating this hypothesis was not trivial. It is the result of a four-year long effort of collaboration with academic users such as astronomers, biologists, social scientists [17]–[23], as well as with industrial partners. By observing the challenges and struggles that these non-expert users have when using existing ML systems, our view on ML usability has gradually shifted from emphasizing the *speed and efficiency* of ML training and inference, to the *capacity of automation* of ML training, before finally evolving into our current view of the *lifecycle/process management* and the aforementioned hypothesis.

As a reflection of our hypothesis, we present ease.ml, a research prototype aimed at lifting the burden of managing the entire development lifecycle from *non-expert* ML application developers. Unlike existing AutoML work that aims to provide automated tools for *individual steps* in the development cycle, or other work that aims at making each individual component perform more easily and efficiently, the goal of ease.ml is to provide automated *toolchains* and well-defined *processes* that *stitch together* existing/upcoming tools to improve end-to-end development experience. ease.ml defines an *eight-step, human-in-the-loop process* to provide systematic guidelines for the users during the entire *ML journey*. Each of these steps often requires us to formulate new research problems and non-trivial technical solutions. Putting them together requires us to rethink factors such as the data model and execution model.

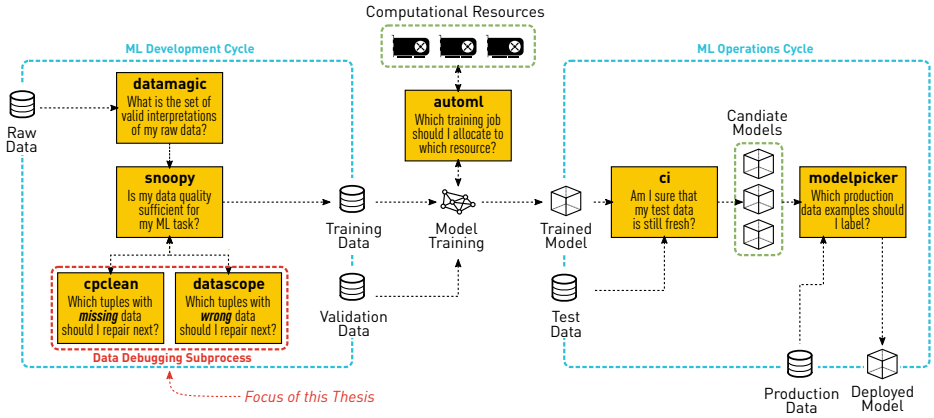

FIGURE 2.1: ease.ml Process

In this chapter, we present the ease.ml process and design decisions that we made in the model for data, execution, and user interaction. The techniques to enable some version of each of the steps depicted in Figure 2.1 have been described in prior research, some of which was done with more direct involvement from the author of this thesis [24]–[30], and some more work conducted by the broader research group that the author is a member of [31]–[37]. The ease.ml system is a first attempt at putting these components together to form an integrated system. As such, it is meant to serve as a proof-of-concept, which we hope to be able to stimulate discussion and inspire future research.

2.2 A Data-Centric ML Workflow

As mentioned, the process of developing ML applications goes beyond designing better models with faster training and inference. In Figure 2.1 we can see this broader context through the lens of the ease.ml process. The goal of the ease.ml process is to provide systematic guidance for non-expert users building ML applications – by following each step in the ease.ml process, users end up with an ML application with certain quality guarantees. The goal of the system is to minimize human effort in a way that is as systematic as possible.

Apart from the model training stage which still sits at the center of the ML workflow, we can see two emerging subprocesses which we refer to as “cycles”

beause of their iterative nature. The two cycles are: (1) the *development cycle*; and (2) the *operations cycle*. The development cycle refers to the process of model development by developing the source data used to train and evaluate the model (i.e. *data development*). The operations cycle involves a data-driven process of *model management* that takes the model from the training subsystem all the way to production. We can notice that both of these processes are inherently driven by data which is why we refer to the overall workflow as "data-centric".

2.3 Pitfalls and Confusions

In this section, we take the perspective of an end user, and walk through a full user experience following an ease.ml process. In ease.ml, we assume the following profile for a non-expert user:

1. The user understands the domain, application, and dataset well. They know how to clean up the data, where to acquire new data, and how to measure the success and quality of an ML model. The user also knows the specific task that they want to achieve by using ML.

2. The user knows how to write a simple Python script to manipulate and transform their data, if necessary.

3. The user knows how to invoke an AutoML system by following the instructions or user manual.

4. The user knows basic concepts of ML (e.g., the meaning of accuracy and a classifier); however, they do not have experience in constructing real-world ML applications beyond the simplified examples which can be found in a textbook.

Each step of ease.ml is designed to avoid some of the pitfalls and confusions that, in our experience, we observed many users struggling with, even when they are equipped with state-of-the-art ML tools. Before we present the concrete ease.ml process, we first describe the key pitfalls and confusions that we hope to address. Note that, this list is *by no means* a complete list of struggles that users have when using existing ML ecosystems and *by no means* does ease.ml provide a full solution to the listed issues.

P1. Unrealistic expectations of quality.

A common problem is that many users often have unrealistic expectations of the quality that ML can provide — it is not uncommon that our users come to us with a very noisy or ill-defined dataset, but still hope for >90% accuracy. Without identifying these problems as early as possible, many users are set on a journey of AutoML and feature engineering that is doomed to fail.

As an analogy to traditional software engineering, users need the functionality of automatic *feasibility study* for ML workloads.

P2. "How many training/testing examples do I need?"

A common question that many users ask during our first meeting is about the amount of data that they need to acquire. Systematic answers to such questions are important to ensure that users are working with representative data samples. Among these, a key step to ensure the quality of the development and deployment process is to *automatically* make sure users have a test set that is large enough to be representative, but small enough to be affordable to acquire. Another question, which most users do not explicitly ask, but is somehow more important, focuses on how to *manage* the statistical power of a test set. Even if given a large enough test set at the beginning, it can quickly lose its statistical power during testing and can lead to overfitting. This can often cause overly optimistic expectations on accuracy and misguide the development process.

As an analogy to software engineering, users need the functionality of *principled test-driven development support* for ML workloads.

P3. "What should I do to further improve accuracy?"

It is rare for an AutoML system to directly output a model with satisfactory quality. Often, it is an iterative process in which users need to continuously improve the data artifact via feature engineering, data acquisition, data labeling, and data cleaning, and to run AutoML systems over different versions of the artifact. One confusion that users often have is *what to do next to further improve a data artifact?* Often, we see users who have access to a collection of powerful tools such as (1) Snorkel for weak label acquisition, (2) HoloClean for automatic data cleaning, (3) Label Box for label acquisition, (4) manual feature engineering, and (5) manual data cleaning, but nonetheless ask: *which tool should I use next?* It is not uncommon for us to

see users spend a huge amount of time on one of these steps, but not end up with better ML accuracy.

As an analogy to software engineering, users need support to avoid *"pre-mature optimizations"* for ML workloads.

P4. "Will this new model that I just found on arXiv today work better?"

ML is an area that is undergoing rapid refreshing — everyday, there seem to be new methods developed for a specific application. One common question that users often have is whether these new techniques, developed by different researchers, can end up helping their ML tasks. However, maintaining an ML artifact with respect to new models and new methods can be expensive (in terms of both computation and manual efforts), and this is an area in which many users are looking for a principled solution.

As an analogy to software engineering, users need the support of *"continuous integration and delivery"* for ML workloads.

P5. "How can we know that we are not applying a model trained on February's data during Christmas?"

We see a subset of users (especially those from industry who are dealing with mission critical applications) who are continuously worried about the mismatch between incoming production data and models. Often, these users have access to a collection of models, each of which is constructed under certain assumptions on the production workload — for example, one may employ twelve models within a one-year time frame, each dealing with a single month's workload. The challenges that these users are facing are (1) how to automatically adapt to the ever-changing production data distribution; and (2) how to pick the best model to use given fresh production data in a labor-efficient manner.

The design of the ease.ml process is inspired by these pitfalls and confusions, which we observed from interacting with our users.

2.4 The ML Development Cycle

At the beginning of the process, the user provides ease.ml with a dataset $\mathcal{D}$. We define the data model precisely in later chapters but, intuitively, a dataset $\mathcal{D}$ consists of: (1) Four sub-datasets with similar schema: a labeled training set and validation set; and an unlabeled pool example and an unlabeled test set; (2) All the uncertainties — for example, missing feature value and all their candidate values, different possible weak supervision labels, and the results of different automatic data cleaning tools — are marked down as meta-data. As we will see, the above will be modeled using a data model that is a probabilistic database, which we will describe later. The goal of development cycle is to make sure that the *user systematically improves the data artifact until the system expects that ML can provide the level of quality that the user desires.*

2.4.1 Automatic Data Injection and Augmentation

- **Pain Point:** What is the set of valid interpretations of my raw data?
- **Input:** Input dataset.
- **Output:** Augmented, machine readable, dataset.
- **Next Step:** Automatic Feasibility Study.
- **System Components:** ease.ml/datamagic [31], [32]

As the first step of the process, the user needs to get their data into the system. Data injection is often a painful process that is frequently overlooked, especially for data stored in a diverse format. As an important preliminary step, ease.ml contains a set of default templates to deal with data injection. One such example is the `Document` template, for which ease.ml translates documents stored in different formats, such as `.PDF`, `.DOCX`, and `.HTML`, into a single, machine-readable data structure. It also contains a set of default data augmentation templates that the systems can automatically apply in order to increase the data volume.

2.4.2 Automatic Feasibility Study

- **Pain Point:** Is my data quality sufficient for my ML task?
- **Input:** (1) Augmented, machine readable, dataset; (2) Expected accuracy.
- **Output:** A {`Yes`, `No`} signal of whether the system believes that the expected accuracy is achievable given the input dataset
- **Next Step:** If `No`, Labor-efficient Data Cleaning and/or Labor-efficient Data Acquisition; If `Yes`, AutoML.
- **System Components:** ease.ml/snoopy [33]

Once the data is machine readable, ease.ml conducts an automatic feasibility study – just like how a real-world ML consultant works, ease.ml automatically looks at the data and provides its *belief* of the best possible accuracy that an ML model can provide. If the answer is "No", ease.ml advises the user to improve their data artifact before firing up an expensive AutoML process.

CHALLENGES. To enable such an *automatic feasibility study* functionality, the system needs to estimate the *irreducible error* of a given task, known as Bayes error, in ML. Providing a practical Bayes error estimator has been an open problem in ML for decades.

2.4.3 Labor-efficient Data Repair

- **Pain Point:** Which data example should I repair?
- **Input:** Augmented, machine readable, dataset.
- **Output:** A prioritized list of dirty data examples to be manually cleaned.
- **Next Step:** Automatic Feasibility Study.
- **System Components:** ease.ml/cpclean [24], ease.ml/datascope

If the answer returned by the automatic feasibility study component is "No," ease.ml advises the user to improve their data artifact before firing up an AutoML

engine. To this end, the user has two options: (1) clean up the data manually or (2) acquire more data and labels. ease.ml treats each of these two options as an "arm" in a multi-armed bandit setting and automatically balances the efforts spent on each of them. When the system asks the user to clean-up the data manually, it provides the user with a prioritized list of dirty examples. This list is selected by estimating the *impact* of cleaning up a certain data example to the downstream ML prediction accuracy. The goal is to minimize the manual cleaning effort to reach high ML accuracy. The system then reruns the *Automatic Feasibility Study* step and gets a new estimate on the irreducible error.

CHALLENGES. To enable such a *labor-efficient data cleaning* functionality, the system needs to analyze the impact of data cleaning over downstream ML tasks. Towards this goal, ActiveClean [15] has done seminal work for a subset of ML models. However, how to support this for more general ML models remains an open question.

2.5 The ML Model Training Cycle

This cycle represents the traditional focus point of the "model-centric" machine learning development approach. In ease.ml, we offer recognize one practical pain point that was unresolved by prior work – allocating jobs to computational resources in a multi-tenant setting.

2.5.1 Multi-Tenant Scheduling of Model Training Jobs

- **Pain Point:** Which training job should I allocate to which resource?
- **Input:** Augmented, machine readable, dataset.
- **Output:** An "endless" *stream* of ML models.
- **Next Step:** Continuous Integration.
- **System Components:** ease.ml/automl [28], [29], [37]

Once the user passes the Pre-ML subprocess, they now have a data artifact that the system *believes* to have the potential to reach their expected accuracy. At this stage, ease.ml fires up the AutoML component and generate ML models. Like other AutoML systems, ease.ml deals with the entire end-to-end ML pipeline, including automatic feature selection and engineering, model selection, hyperparameter tuning, etc. The functionality of this step largely follows existing platforms, but with one specific twist — when there is a new ML model made available to the system, *all* users' applications that could potentially use this model are rerun. As a result, the outcome of the AutoML component of ease.ml is not a single model, but a *stream* of models.

CHALLENGES. To enable such a *continuously updated* AutoML component, the system needs to be efficient in how it prioritizes different applications and different users. When there are multiple applications, each of which can be updated with latest model, the system needs to pick those applications that can benefit the most. This poses a new challenge that we call *multi-tenant AutoML*.

2.6 The ML Opeartions Cycle

This cycle focuses on bridging the gap between the model training process which produces an *endless stream of trained models* and the production environment where models are integrated into broader ML applications. The operations cycle is focused on model management and involves tasks such as reliable model testing and careful selection of the best model that should end up getting deployed.

2.6.1 Continuous Integration

- **Pain Point:** Am I sure that my test data is still fresh?
- **Input:** (1) A stream of ML models output by the AutoML component; (2) user-defined requirements of deployment.
- **Output:** A stream of ML models that satisfy the requirements.
- **Next Step:** Continuous Quality Optimization.
- **System Components:** ease.ml/ci [25]–[27]

The Post-ML subprocess concerns the deployment of models. Given a stream of ML models output by the AutoML component, not all models can be deployed. The first step in the Post-ML subprocess is for ease.ml to test whether each of the models satisfies a user-defined deployment criteria, such as *the new model must be better than the old model by at least 5%* or *the new model cannot change more than 10% of the old model's predictions*. The system then only keeps those models that satisfy the given conditions. In ease.ml, all these criteria can be tested by using a test set. The system lets the user know the number of test examples it needs to achieve a rigorous statistical significance level and when it needs a new test set to avoid overfitting.

CHALLENGES. The challenge is how to re-use a test set efficiently to provide rigorous guarantees, without overfitting. Intuitively, at the moment that ease.ml provides the test result to a user, it "leaks" some information from the test set to the user. To avoid overfitting, the system needs to "measure" the information leakage and then automatically manage the statistical power of the test set. This is a challenging statistics problem known as adaptive analytics.

2.6.2 Continuous Quality Optimization

- **Pain Point:** Which candidate model is best suited for my fresh production data?
- **Input:** (1) A set of ML models that satisfy the user-defined requirements; (2) A set of unlabeled, fresh test examples from the production.
- **Output:** A set of test examples to label such that one can pick the best ML model to use in production.
- **Next Step:** Automatic Quality Debugging and Recommendations.
- **System Components:** ease.ml/modelpicker [30]

In principle, all models passing the previous step can be candidates to be deployed in production. This gives us a candidate set of models. The goal of this step is to pick, from this candidate set, the best model to use for the next period of time (e.g., a day). To achieve this goal, ease.ml asks the user to collect a set of unlabeled test examples from production (e.g., at 8 a.m. everyday). It then tries to pick the candidate model that performs the best on this fresh test set, by acquiring as few labels as possible. This chosen model is then used (e.g., the next day). The user repeats this process, e.g., on a daily basis, such that one often uses the best model for the ever-changing test data distribution. ease.ml also automatically applies existing domain adaptation techniques, which create more models automatically — the system feeds them into the *continuous integration* step to make sure that they satisfy the deployment requirement and automatically add them to the candidate set.

CHALLENGES. The challenge is to minimize the number of labels that we need to acquire to distinguish between a large collection of models. This itself is an interesting online learning problem.

2.7 Design and Implementation

In this section, we highlight a few design choices and implementation details when building ease.ml.

2.7.1 Data Model and Query

DATA MODEL. The core data model of ease.ml is *relational*. Each object is a set X that is an instance of some relation $R := \{c_1 : T_1, c_2 : T_2, ...\}$, where $c_1, c_2, ...$ are column names and $T_1, T_2, ...$ are column types. A type T can be a simple primitive (e.g., string or integer) but can also be more complex, such as a nested relation. Thus, objects are sets of named and typed tuples. For example, we can have a relation $\texttt{Persons} := \{\texttt{name} : \texttt{String}, \texttt{age} : \texttt{Int}, \texttt{houses} : \texttt{House}\}$, where `House` is another relation. We list several other features of our data model in the remainder of this section.

MODELING UNCERTAINTY. We observed that ML datasets often involve uncertainty, for example when performing data augmentation or cleaning. To model this, we adopt probabilistic relational semantics in our data model. Hence, each tuple $t \in X$ is associated with a set of *candidate* tuples $t := \{t_1, t_2, ...\}$, referred to as a *block*. The semantics of each tuple is that it evaluates to a single candidate chosen at random, based on some distribution. For example, we may not be sure about the age of a person, so we can store it as a set of candidate values, each associated with some probability.

COMPUTATIONAL OPERATORS. We use the same relational model to represent our computational operators. A relation can be "invoked" to produce a *query*. For example, $\texttt{Persons}(\texttt{age} = 30)$ would define a query of all people with age 30. Running that query would require a database lookup. Furthermore, we can have a relation associated with an arbitrary function. For example, any of the ease.ml components (e.g., `automl`) is a relational operator. Hence, invoking

$$\texttt{automl}(\texttt{d\_tr} = D_{tr}, \texttt{d\_val} = D_{val}, \texttt{models} = M)$$

defines a query that tries to find a model $m \in M$ which returns the best score when trained on D_{tr} and evaluated on D_{val}.

SUBTYPING. Given that *relations are types*, a query Q over a relation R can be viewed as a subtype $Q \subseteq R$. All types (relations and queries) can thus be stored in a *unified* hierarchical namespace. Executing a query is thus equivalent to computing

an instance of the type Q. This subtyping system is useful to ensure *type safety* of the composition of computational operators.

2.7.2 Workflow Construction and Execution

An ML workflow includes a lot of exploration. Most explored paths lead to dead ends and can be thrown away. Few may turn out to be promising, and we want to store them so that we can revisit them later and reproduce their results. However, it is almost impossible to know the promising paths *beforehand*. An ideal user interface would give us maximum flexibility and minimum overhead in the beginning, while allowing us to store and reuse already explored paths when running the ML workflow.

We design our workflow mechanism with the above goals in mind. We treat the *query* as a central object for performing any form of computation in ease.ml. A query can be iteratively constructed so it implicitly carries information about its entire *lineage*. We provide various facilities for managing queries.

COMPOSITION. Queries are *composable* objects that describe a *result set*. As such, they can be included in other queries to form an augmented workflow graph.

ANALYSIS. We can inspect the structure of the execution graph in order to determine the state of the workflow. We use this analysis to implement the aforementioned `whatnext` command that suggests potential next steps that might be fruitful.

STORAGE. Queries can be stored inside a unified hierarchical namespace just like any other object.

EXECUTION. Finally, the result set of a query can be materialized. This result set can be cached for future reuse. If a query references other queries, we first check if the corresponding results are cached before executing the referenced queries.

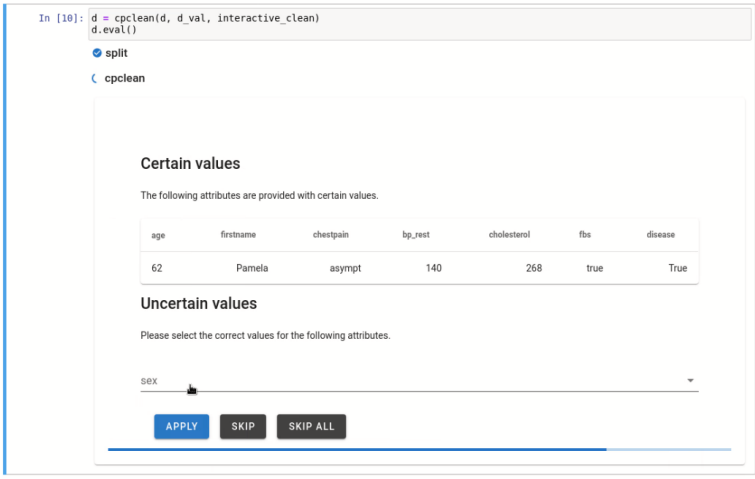

FIGURE 2.2: The interactive labeling interface.

2.7.3 User Interaction

To ensure familiarity for data scientists and interoperability with most ML tools, we choose Python as the main programming language for users to interact with ease.ml. We provide a rich set of functions that can be used for accessing object storage, data inspection, query construction and execution.

For maximum interactivity, we opt for Jupyter notebook as the programming environment, which makes it easy to experiment with different approaches and immediately visualize results.

One step towards this goal involves pluggable UI components that can be an integral part of the user's workflow. For example, a user may want to clean some data using the `cpclean` component provided by ease.ml. This component selects a minimal subset of dirty data points that can be cleaned interactively by the user. In Figure 2.2, we show an example of this interactive cleaning UI that pops below a regular code cell in the Jupyter notebook upon execution of the workflow graph.

RECOMMENDATION ON NEXT EASE.ML STEPS. It can be quite hard to manage the complexity of ML workflows, especially for less experienced data

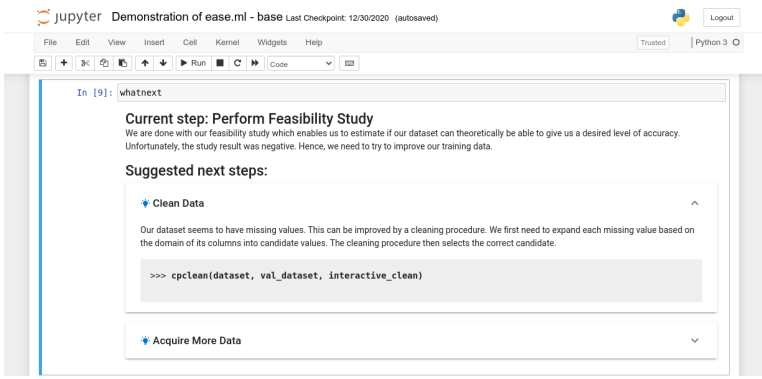

FIGURE 2.3: The interface of the `whatnext` command.

scientists. There are lots of decisions to be made and each sub-optimal decision leads to time being spent on exploring paths that will not be fruitful.

To ease the burden on the user, ease.ml further provides a command called `whatnext`. It analyzes the current state of the ML workflow and offers recommendations about sensible next steps that the user can take. Currently, we implement `whatnext` as a simple rule-based recommendation engine, but it can be augmented in the future. Figure 2.3 presents the interactive UI that pops up when the user invokes the `whatnext` command.

2.8 Conclusion

We have presented ease.ml, a novel platform that aims to provide usable end-to-end management of ML workflows within the overall application development process. We present the steps of the ease.ml process that spans the entire ML development and operations lifecycle, motivated by the pitfalls and confusions we found during interactions with our users. We also briefly describe some key decisions we have made when designing ease.ml.

In the next chapters we will zoom into one specific pitfall – deciding how to perform targeted data repairs to maximize model quality. This will allow us to explore internals of specific components of the ease.ml system.

3

REPAIR OF MISSING DATA GUIDED BY INFORMATION GAIN

In previous chapters we have seen an overview of the broader space of ML workflow management systems as well as a presentation of our own vision of what that system could look like. In the remaining chapters we will look closer at one specific problem that arises very frequently in practice – data debugging. This is a process of discovering and repairing data errors in order to improve the quality of data, and in turn, improve model quality. In particular, we are interested in labor-efficient data debugging, which aims to maximize our model quality while minimizing our repair efforts. We will examine two methods which are geared for two types of data errors: missing data and wrong data. In this chapter we focus on the first type of data error – missing data. We will describe the internals of ease.ml/cpclean, a system component designed for optimizing this type of data debugging task. At its core is an algorithm which we call **CPClean**, which drives the actual optimization. In Figure 3.1 we can see the place of this component in the overall ease.ml workflow.

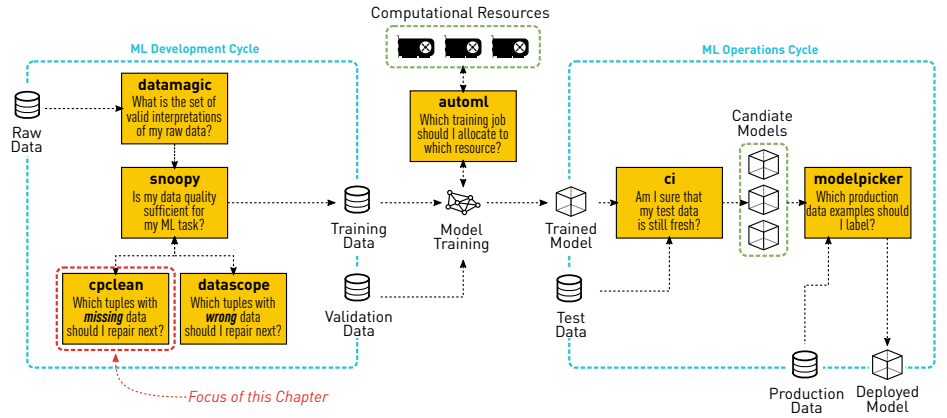

FIGURE 3.1: In this chapter we focus on repairing tuples with *missing data*.

We start off with missing data, or, in other words, *incomplete data*. In this chapter we review the theory that is going to be at the foundation of **CPClean**, our repair method for missing data. The theoretical contribution relies on a fundamental question – *How should we reason about model predictions over missing data?*

In this chapter, we will seek inspiration from established database theory and examine how databases represent and perform queries over incomplete data. In databases we use the NULL value to indicate that a cell in a table has a missing value. The meaning of NULL in database theory is that the true value is unknown, and the field can potentially assume any *candidate value* taken from some domain. When we consider all possible ways replace NULL values in our dataset, we will see that this induces an exponential number of possible datasets, which we refer to as the set of *possible worlds*. We will then try to apply the same semantics to the problem of learning over incomplete data. Then, we will define two fundamental queries for reasoning about predictions of a model trained on incomplete data and present PTIME algorithms for answering those queries.

Finally, we will show how these two queries can be applied to the problem of labor efficient repair of missing values. We will show the connection between data repair and the reduction of uncertainty. Then, we will express this reduction of uncertainty using a well known quantity taken from information theory – *information gain*. This will be the measure of importance which we will use as a "guiding signal" for making decisions about which data examples should be prioritized. Finally, we will perform empirical evaluations of this guiding signal, and show its effectiveness at minimizing our cleaning effort.

3.1 Background and Related Work

Building high-quality Machine learning (ML) applications often hinges on the availability of high-quality data. However, due to noisy inputs from manual data curation or inevitable errors from automatic data collection/generation programs, in reality, data is unfortunately seldom clean. Inconsistency and incompleteness are ubiquitous in real-world datasets, and therefore can have an impact on ML applications trained on top of them. In this chapter, we focus on the question: *Can we reason about the impact of data incompleteness on the quality of ML models trained over it?*

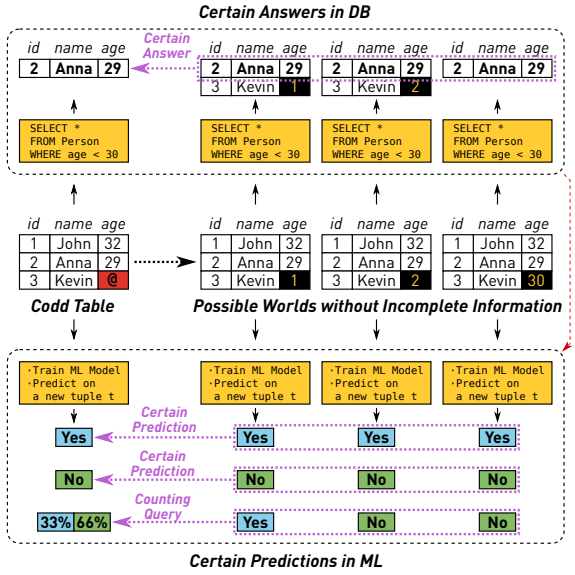

FIGURE 3.2: An illustration of the relationship between the semantics of *certain answers* (above) and *certain predictions* (below). Both examples are constructed on top of table with a single missing value (the age of Kevin) being expanded into a set of *possible worlds*.

Figure 3.2 illustrates one dataset with incomplete information. In this example, we have the incomplete dataset D with one missing cell (we will focus on cases in which there are *many* cells with incomplete information) — the age of Kevin is not known and therefore is set as `NULL` (@). Given an ML training algorithm $\mathcal{A}$, we can train an ML model over D, $\mathcal{A}_D$, and given a clean test example t, we can get the prediction of this ML model $\mathcal{A}_D(t)$. The focus of this chapter is to understand *how much impact the incomplete information (@) has on the prediction* $\mathcal{A}_D(t)$. This question is not only of theoretical interest but can also have interesting practical implications — for example, if we know that, for a large enough number of samples of t, the incomplete information (@) does not have an impact on $\mathcal{A}_D(t)$ at all, spending the effort of cleaning or acquiring this specific piece of missing information will not change the quality of downstream ML models.

LEARNING OVER INCOMPLETE INFORMATION The traditional database view provides us a powerful tool to reason about the impact of data incompleteness on downstream operations. In this chapter, we take a natural step and extend this to

machine learning (ML) — given a Codd table T with n variables defined over some domain $\mathcal{D}_v$, the set of its $|\mathcal{D}_v|^n$ many possible worlds $rep(T)$, and an ML classifier $\mathcal{A}$, one could train one ML model $\mathcal{A}_I$ for each possible world $I \in rep(T)$. Given a test example t, we say that t can be *certainly predicted (CP'ed)* if $\forall I \in rep(T)$, $\mathcal{A}_I(t)$ always yields the same class label, as illustrated in Figure 3.2. This notion of certain prediction (CP) offers a canonical view of the impact from training classifiers on top of incomplete data. Specificlly, we consider the following two CP queries:

(Q1) Checking Query — Given a test data example, determine whether it can be CP'ed or not;

(Q2) Counting Query — Given a test data example that *cannot* be CP'ed, for each possible prediction, compute the number of classifiers that *support* this prediction.

When no assumptions are made about the classifier, Q1 and Q2 are, not surprisingly, hard. In this chapter, we focus on (1) developing efficient solutions to both Q1 and Q2 for a specific family of classifiers, while (2) in the meantime, trying to understand the empirical implication and application of CP to the emerging research topic of *data cleaning for machine learning*.

EFFICIENT ALGORITHMS FOR NEAREST NEIGHBOR CLASSIFIERS. We first study efficient algorithms to answer both CP queries for the well known K-nearest neighbor (KNN) classifier. Surprisingly, we show that, *both CP queries can be answered in polynomial time, in spite of there being exponentially many possible worlds*!

Moreover, these algorithms can be made very efficient. For example, given a Codd table with N rows, at most M possible versions for rows with missing values and K as the number of neighbors used in the KNN algorithm, we show that answering both queries only take $\mathcal{O}\left(N \cdot M \cdot (\log(N \cdot M) + K \cdot \log N)\right)$. For Q1 in the binary classification case, we can even do $\mathcal{O}(N \cdot M)$! This makes it possible to efficiently answer both queries for the KNN classifier, a result that is both *surprising* (at least to us), *new*, and *technically non-trivial*.

The result described in the previous chapter is not only of theoretical interest, but also has an interesting empirical implication — *intuitively, the notion of certain prediction provides us a way to measure the relative importance of different variables*

in the Codd table to the downstream classification accuracy. Inspired by this intuition, we study the efficacy of CP in the imporant application of "data cleaning for machine learning (DC for ML)" [15], [38]. Based on the CP framework, we develop a novel algorithm CPClean that prioritizes manual cleaning efforts given a dirty dataset.

Data cleaning (DC) is often an important prerequisite step in the entire pipeline of an ML application. Unfortunately, most existing work considers DC as a standalone exercise without considering its impact on downstream ML applications (exceptions include exciting seminal work such as ActiveClean [15] and BoostClean [38]). Studies have shown that such *oblivious* data cleaning may not necessarily improve downstream ML models' performance [39]; worse yet, it can sometimes even degrade ML models' performance due to Simpson's paradox [15]. We propose a novel "DC for ML" framework built on top of certain predictions. In the following discussion, we assume a standard setting for building ML models, where we are given a training set D_{train} and a validation set D_{val} that are drawn independently from the same underlying data distribution. We assume that D_{train} may contain missing information whereas D_{val} is complete.

The intuition of our framework is as follows. When the validation set is sufficiently large, if Q1 returns `true` for every data example t in D_{val}, then with high probability cleaning D_{train} will not have impact on the model accuracy. In this case we can immediately finish without any human cleaning effort. Otherwise, some data examples cannot be CP'ed, and our goal is then to clean the data such that all these examples can be CP'ed. *Why is this sufficient*? The key observation is that, as long as a tuple t can be CP'ed, the prediction will remain the same regardless of further cleaning efforts. That is, even if we clean the whole D_{train}, the prediction for t (made by the classifier using the *clean* D_{train}) will remain the same, simply because the final clean version is one of the possible worlds of D_{train} that has been included in the definition of CP!

To *minimize* the number of tuples in D_{train} being cleaned until all data examples in D_{val} are CP'ed, we further propose a novel optimization algorithm based on the principle of *sequential information maximization* [40], exploiting the counts in Q2 for each example in D_{val} that cannot be certainly predicted. The optimization algorithm is *iterative*: Each time we pick the next example in D_{train} (to be cleaned) based on its potential impact on the "degree of certainty" of D_{train} after cleaning (see Section Chapter 3 for more details).

PRIOR WORK ON QUERYING INCOMPLETE INFORMATION. This work is heavily inspired by the database literature of handling incomplete information [41], consistent query answering [42]–[44], and probabilistic databases [45]. While these work targets SQL analytics, our proposed consistent prediction query targets ML analytics.

PRIOR WORK ON LEARNING OVER INCOMPLETE DATA. The statistics and ML community have also studied the problem of learning over incomplete data. Many studies operate under certain missingness assumption (e.g., missing completeness at random) and reason about the performance of downstream classifiers in terms of asymptotic properties and in terms of different imputation strategies [46]. In this work, we focus more on the algorithmic aspect of this problem and try to understand how to enable more efficient manual cleaning of the data. Another flavor of work aims at developing ML models that are robust to certain types of noises, and multiple imputation [47] is such a method that is most relevant to us. Our CP framework can be seen as an extreme case of multiple imputation (i.e, by trying all possible imputations) with efficient implementation (in KNN), which also enables novel manual cleaning for ML use cases.

Recently, Khosravi et al. [48] explored a similar semantics as ours, but for Logistic Regression models. In the work presented here, we focus on efficient algorithms for nearest neighbor classifiers.

PRIOR WORK ON ANALYTICS-DRIVEN DATA CLEANING. The research on data cleaning (DL) has been thriving for many years. Many data cleaning works focus on performing standalone cleaning without considering how cleaned data is used by downstream analytics. We refer readers to a recent survey on this topic [49].

As data cleaning itself is an expensive process that usually needs human involvement eventually (e.g., to confirm suggested repairs), the DB community is starting to work on analytics-driven cleaning methods. SampleClean [50] targets the problem of answering SQL aggregate queries when the input data is dirty by cleaning a sample of the dirty dataset, and at the same time, providing statistical guarantees on the query results. ActiveClean [15] is an example of cleaning data intelligently for convex ML models that are trained using gradient descent methods. As discussed before, while both ActiveClean and our proposal assume the use of a human cleaning oracle, they are incomparable as they are targeting different ML

models. BoostClean [38] automatically selects from a predefined space of cleaning algorithms, using a hold-out validation set via statistical boosting. We show that our proposal significantly outperforms BoostClean under the same space of candidate repairs.

3.2 Preliminaries: Relational Queries over Incomplete Information

This work is inspired by the *algorithmic* and *theoretical* foundations of running *relational queries over incomplete information* [41]. In traditional database theory, there are multiple ways of representing incomplete information, starting from the Codd table, or the conditional table (c-table), all the way to the recently studied probabilistic conditional table (pc-table) [45]. Over each of these representations of incomplete information, one can define the corresponding semantics of a relational query. In this work, we focus on the weak representation system built upon the Codd table, as illustrated in Figure 3.2. Given a Codd table T with constants and n variables over domain $\mathcal{D}_v$ (each variable only appears once and represents the incomplete information at the corresponding cell), it represents $|\mathcal{D}_v|^n$ many *possible worlds* $rep(T)$, and a query $\mathtt{Q}$ over T can be defined as returning the *certain answers* that *always* appear in the answer of $\mathtt{Q}$ over each possible world:

$$sure(\mathtt{Q}, T) = \cap\{\mathtt{Q}(I) | I \in rep(T)\}.$$

Another line of work with similar spirit is *consistent query answering*, which was first introduced in the seminal work by Arenas, Bertossi, and Chomicki [42]. Specifically, given an inconsistent database instance D, it defines a set of repairs $\mathcal{R}_D$, each of which is a consistent database instance. Given a query $\mathtt{Q}$, a tuple t is a *consistent answer* to $\mathtt{Q}$ if and only if t appears in *all answers* of $\mathtt{Q}$ evaluated on every consistent instance $D' \in \mathcal{R}_D$.

Both lines of work lead to a similar way of thinking in an effort to reason about data processing over incomplete information, i.e., *to reason about certain/consistent answers over all possible instantiations of incompleteness and uncertainty.*

3.3 The Certain Prediction (CP) Framework

In this section, we describe the *certain prediction* (CP) framework, which is a natural extension of the notion of *certain answer* for query processing over Codd tables [41] to machine learning. We first describe our data model and then introduce two CP queries.

DATA MODEL We focus on standard supervised ML settings:

1. Feature Space $\mathcal{X}$: without loss of generality, we assume that every data example is drawn from a domain $\mathcal{X} = \mathbb{D}^d$, i.e., a d dimensional space of data type $\mathbb{D}$.

2. Label Space $\mathcal{Y}$: we assume that each data example can be classified into one of the labels in $\mathcal{Y}$.

3. Training Set $D_{train} \subseteq \mathcal{X} \times \mathcal{Y}$ is drawn from an *unknown* distribution $\mathcal{P}_{\mathcal{X},\mathcal{Y}}$.

4. Test Set $D_{test} \subseteq \mathcal{X}$ (Validation Set D_{val}) is drawn from the marginal distribution $\mathcal{P}_{\mathcal{X}}$ of the joint distribution $\mathcal{P}_{\mathcal{X},\mathcal{Y}}$.

5. Training Algorithm $\mathcal{A}$: A training algorithm $\mathcal{A}$ is a functional that maps a given training set D_{train} to a function $\mathcal{A}_{D_{train}} : \mathcal{X} \mapsto \mathcal{Y}$. Given a test example $t \in D_{test}$, $\mathcal{A}_{D_{train}}(t)$ returns the prediction of the trained classifier on the test example t.

INCOMPLETE INFORMATION IN THE TRAINING SET In this work, we focus on the case in which there is incomplete information in the training set. We define an *incomplete training set* as follows.

Our definition of an incomplete training set is very similar to a block tuple-independent probabilistic database [45]. However, we do assume that there is no uncertainty on the label and we do not have access to the probability distribution of each tuple.

Definition 1 (Incomplete Dataset)**.** *An* incomplete dataset

$$\mathcal{D} = \{(\mathcal{C}_i, y_i) : i = 1, ..., N\}$$

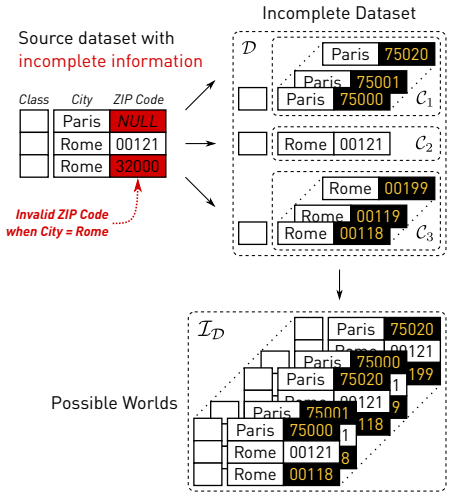

FIGURE 3.3: Example of a dataset with incomplete information, its representation as an *incomplete dataset*, and the induced set of *possible worlds*.

is a finite set of N pairs where each $\mathcal{C}_i = \{x_{i,1}, x_{i,2}, ...\} \subset \mathcal{X}$ is a finite number of possible feature vectors of the i-th data example and each $y_i \in \mathcal{Y}$ is its corresponding class label.

According to the semantics of $\mathcal{D}$, the i-th data example can take any of the values from its corresponding *candidate set* $\mathcal{C}_i$. The space of all possible ways to assign values to all data points in $\mathcal{D}$ is captured by the notion of possible worlds. Similar to a block tuple-independent probabilistic database, an incomplete dataset can define a set of *possible worlds*, each of which is a dataset without incomplete information.

Definition 2 (Possible Worlds)**.** *Let $\mathcal{D} = \{(\mathcal{C}_i, y_i) : i = 1, ..., N\}$ be an incomplete dataset. We define the set of possible worlds $\mathcal{I}_\mathcal{D}$, given the incomplete dataset $\mathcal{D}$, as*

$$\mathcal{I}_\mathcal{D} \;=\; \left\{D = \{(x'_i, y'_i)\} : |D| = |\mathcal{D}| \wedge \forall i.\ x'_i \in \mathcal{C}_i \wedge y'_i = y_i\right\}.$$

In other words, a *possible world* represents one complete dataset D that is generated from $\mathcal{D}$ by replacing every candidate set $\mathcal{C}_i$ with one of its candidates $x_j \in \mathcal{C}_i$. The set of all distinct datasets that we can generate in this way is referred to as the *set of possible worlds*. If we assume that $\mathcal{D}$ has N data points and the size of each $\mathcal{C}_i$ is bounded by M, we can see $|\mathcal{I}_\mathcal{D}| = \mathcal{O}(M^N)$.

Figure 3.3 provides an example of these concepts. As we can see, our definition of incomplete dataset can represent both possible values for missing cells and possible repairs for cells that are considered to be potentially incorrect.

CONNECTIONS TO DATA CLEANING. In the work presented here, we use data cleaning as one application to illustrate the practical implication of the CP framework. In this setting, each possible world can be thought of as one possible *data repair* of the dirty/incomplete data. These repairs can be generated in an arbitrary way, possibly depending on the entire dataset [4], or even some external domain knowledge [51]. Attribute-level data repairs could also be generated independently and merged together with Cartesian products.

We will further apply the assumption that any given incomplete dataset $\mathcal{D}$ is *valid*. That is, for every data point i, we assume that there exists a *true value* x_i^* that is unknown to us, but is nevertheless included in the candidate set $\mathcal{C}_i$. This is a commonly used assumption in data cleaning [49], where automatic cleaning algorithms are used to generate a set of candidate repairs, and humans are then asked to pick one from the given set. We call $D_{\mathcal{D}}^*$ the *true possible world*, which contains the true value for each tuple. When $\mathcal{D}$ is clear from the context, we will also write D^*.

3.3.1 Certain Prediction (CP)

When we train an ML model over an incomplete dataset, we can define its semantics in a way that is very similar to how people define the semantics for data processing over probabilistic databases — we denote $\mathcal{A}_{D_i}$ as the classifier that was trained on the possible world $D_i \in \mathcal{I}_{\mathcal{D}}$. Given a test data point $t \in \mathcal{X}$, we say that it can be *certainly predicted* (CP'ed) if all classifiers trained on all different possible worlds agree on their predictions:

Definition 3 (Certain Prediction (CP))**.** *Given an incomplete dataset $\mathcal{D}$ with its set of possible worlds $\mathcal{I}_{\mathcal{D}}$ and a data point $t \in \mathcal{X}$, we say that a label $y \in \mathcal{Y}$ can be* certainly predicted *with respect to a learning algorithm $\mathcal{A}$ if and only if*

$$\forall D_i \in \mathcal{I}_{\mathcal{D}}, \mathcal{A}_{D_i}(t) = y.$$

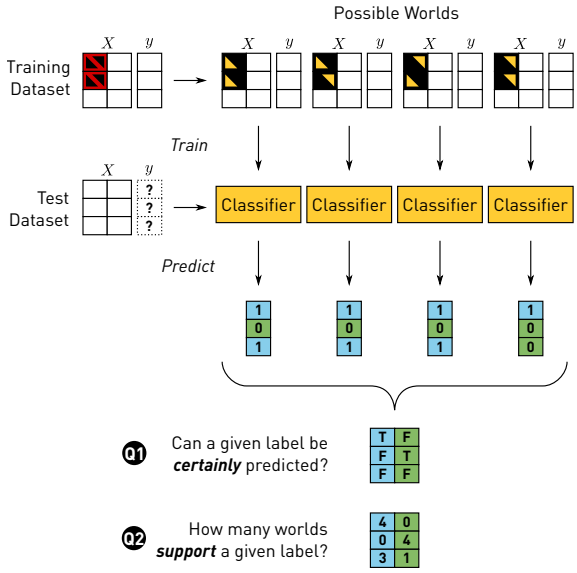

FIGURE 3.4: Illustration of certain prediction, and two queries: checking query (Q1) and counting query (Q2).

CONNECTIONS TO DATABASES. The intuition behind this definition is rather natural from the perspective of database theory. In the context of Codd table, each `NULL` variable can take values in its domain, which in turn defines exponentially many possible worlds [41]. Checking whether a tuple is in the answer of some query `Q` is to check whether such a tuple is in the result of each possible world.

TWO PRIMITIVE CP QUERIES. Given the notion of *certain prediction*, there are two natural queries that we can ask. The query $Q1$ represents a *decision problem* that checks if a given label can be predicted in *all* possible worlds. The query $Q2$ is an extension of that and represents a *counting problem* that returns the number of possible worlds that support each prediction outcome. Figure 3.4 illustrates both queries and we formally define them as follows.

Definition 4 (Q1: Checking)**.** *Given a data point $t \in \mathcal{X}$, an incomplete dataset $\mathcal{D}$ and a class label $y \in \mathcal{Y}$, we define a query that checks* if all possible world *permits y to be predicted:*

$$Q1(\mathcal{D}, t, y) := \begin{cases} \texttt{true}, & \text{if } \forall D_i \in \mathcal{I}_{\mathcal{D}}, \mathcal{A}_{D_i}(t) = y; \\ \texttt{false}, & \text{otherwise.} \end{cases} \tag{3.1}$$

Definition 5 (Q2: Counting)**.** *Given a data point $t \in \mathcal{X}$, an incomplete dataset $\mathcal{D}$ and a class label $y \in \mathcal{Y}$, we define a query that returns the* number of possible worlds *that permit y to be predicted:*

$$Q2(\mathcal{D}, t, y) := |\{D_i \in \mathcal{I}_{\mathcal{D}} : \mathcal{A}_{D_i}(T) = y\}|. \tag{3.2}$$

COMPUTATIONAL CHALLENGE. If we do not make any assumption about the learning algorithm $\mathcal{A}$, we have no way of determining the predicted label $y = \mathcal{A}_{D_i}(t)$ except for running the algorithm on the training dataset. Therefore, for a general classifier treated as a black box, answering both $Q1$ and $Q2$ requires us to apply a brute-force approach that iterates over each $D_i \in \mathcal{I}_{\mathcal{D}}$, produces $\mathcal{A}_{D_i}$, and predicts the label. Given an incomplete dataset with N data examples each of which has M clean candidates, the computational cost of this naive algorithm for both queries would thus be $\mathcal{O}(M^N)$.

This is not surprising. However, as we will see later in this chapter, for certain types of classifiers, such as K-Nearest Neighbor classifiers, we are able to design efficient algorithms for both queries.

CONNECTIONS TO PROBABILISTIC DATABASES. Our definition of certain prediction has strong connection to the theory of probabilistic database [45] — in fact, Q2 can be seen as a natural definition of evaluating an ML classifier over a block tuple-independent probabilistic database with uniform prior.

Nevertheless, unlike traditional relational queries over a probabilistic database, our "query" is an ML model that has very different structure. As a result, despite the fact that we are inspired by many seminal works in probabilistic database [52]–[54], they are not applicable to our settings and we need to develop new techniques.

CONNECTIONS TO DATA CLEANING. It is easy to see that, if $Q1$ returns `true` on a test example t, obtaining more information (by cleaning) for the original training set will not change the prediction on t at all! This is because the true possible world D^* is one of the possible worlds in $\mathcal{I}_{\mathcal{D}}$. Given a large enough test set, if Q_1 returns `true` for all test examples, cleaning the training set in this case might not improve the quality of ML models at all!

Of course, in practice, it is unlikely that all test examples can be CP'ed. In this more realistic case, Q_2 provides a "softer" way than Q_1 to measure the *degree of certainty/impact*. As we will see later, we can use this as a principled proxy of the impact of data cleaning on downstream ML models, and design efficient algorithms to prioritize which uncertain cell to clean in the training set.

3.4 Formal Definition of the Data Cleaning Problem

In this section, we show how to use the proposed CP framework to design an effective data cleaning solution, called CPClean, for the important application of data cleaning for ML. We assume as input a dirty training set $\mathcal{D}_{train}$ with unknown ground truth D^* among all possible worlds $\mathcal{I}_{\mathcal{D}}$. Our goal is to select a possible world D from $\mathcal{I}_{\mathcal{D}}$, such that the classifier trained on $\mathcal{A}_D$ has the same validation accuracy as the classifier trained on the ground truth world $\mathcal{A}_{D^*}$.

CLEANING MODEL. Given a dirty dataset $\mathcal{D} = \{(\mathcal{C}_i, y_i)\}_{i\in[N]}$, in this work, we focus on the scenario in which the candidate set $\mathcal{C}_i$ for each data example is created by automatic data cleaning algorithms or a predefined noise model. For each uncertain data example $\mathcal{C}_i$, we can ask a human to provide its true value $x_i^* \in \mathcal{C}_i$. Our goal is to find a good strategy to prioritize which dirty examples to be cleaned. That is, a cleaning strategy of T steps can be defined as

$$\pi \in [N]^T,$$

which means that in the first iteration, we clean the example π_1 (by querying human to obtain the ground truth value of $\mathcal{C}_{\pi_1}$; in the second iteration, we clean the example π_2; and so on. Applying a cleaning strategy π will generate a *partially cleaned* dataset $\mathcal{D}_\pi$ in which all cleaned candidate sets $\mathcal{C}_{\pi_i}$ are replaced by $\{x^*_{\pi_i}\}$.

PRINCIPLE: ENTROPY MINIMIZATION The theory of submodular functions enables the analysis for the performance of the most informative selection policies. Several studies have applied such theory to identify near optimal solutions to information gathering problem, and show that exploiting information measures is beneficial [55]–[60]. In the context of data cleaning, the mutual information violates the adaptive submodularity condition. We, in this work, leverage the theory of *sequential information maximization* introduced by [40], and adapt its information gathering strategy for data cleaning.

FORMAL CLEANING PROBLEM FORMULATION. The question we need to address is *"What is a successful cleaning strategy?"* Given a validation set D_{val}, the view of CPClean is that a successful cleaning strategy π should be the one that produces a partially cleaned dataset $\mathcal{D}_\pi$ in which all validation examples $t \in D_{val}$ can be certainly predicted. In this case, picking any possible world defined by $\mathcal{D}_\pi$, i.e., $\mathcal{I}_{\mathcal{D}_\pi}$, will give us a dataset that has the same accuracy, on the validation set, as the ground truth world D^*. This can be defined precisely as follows.

We treat each candidate set $\mathcal{C}_i$ as a random variable $\mathbf{c}_i$, taking values in $\{x_{i,1}, ..., x_{i,M}\}$. We write $\mathbf{D} = \{(\mathbf{c}_i, y_i)\}_{i\in[N]}$. Given a cleaning strategy π we can define the conditional entropy of the classifier prediction on the validation set as

$$\mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val})|\mathbf{c}_{\pi_1}, ..., \mathbf{c}_{\pi_T}) \coloneqq \frac{1}{|D_{val}|} \sum_{t\in D_{val}} \mathcal{H}(\mathcal{A}_{\mathbf{D}}(t)|\mathbf{c}_{\pi_1}, ..., \mathbf{c}_{\pi_T}). \tag{3.3}$$

Naturally, this gives us a principled objective for finding a "good" cleaning strategy that minimizes the human cleaning effort:

$$\begin{aligned} &\min_\pi \quad \dim(\pi) \\ &s.t., \quad \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val})|\mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}) = 0. \end{aligned}$$

If we are able to find a cleaning strategy in which

$$\mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val})|\mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}) = 0,$$

we know that this strategy would produce a partially cleaned dataset $\mathcal{D}_\pi$ on which all validation examples can be CP'ed. Note that we can use the query $Q2$ to compute this conditional entropy:

$$\mathcal{H}(\mathcal{A}_{\mathbf{D}}(t)|\mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}) \ = -\sum_{l\in\mathcal{Y}} \frac{Q2(\mathcal{D}_\pi, t, y)}{|\mathcal{D}_\pi|} \log \frac{Q2(\mathcal{D}_\pi, t, y)}{|\mathcal{D}_\pi|}$$

CONNECTIONS TO ACTIVECLEAN. The idea of prioritizing human cleaning effort for downstream ML models is not new — ActiveClean [15] explores an idea with a similar goal. However, there are some important differences between our framework and ActiveClean. The most crucial one is that our framework relies on *consistency* of predictions instead of the *gradient*, and therefore, we do not need labels for the validation set and our algorithm can be used in ML models that cannot be trained by gradient-based methods. The KNN classifier is one such example. Since both frameworks essentially measure some notion of "local sensitivity," it is interesting future work to understand how to combine them.

3.5 Algorithmic Approaches for Answering CP Queries

In this section we cover the algorithmic approaches for answering the CP queries described in Section 3.3.1. We will first briefly cover two relatively straightforward approaches: the brute-force approach which computes the exact answers and the Monte-Carlo sampling approach which computes approximate answers. These approaches make no assumptions regarding the type of machine learning model used to make predictions. On the other hand, we will describe in detail the algorithmic approaches we developed which assume that the K-nearest neighbor model is used to make predictions.

3.5.1 Brute-force Algorithms

The brute-force approach for answering both Q1 and Q2 essentially rely on enumerating the entire set of possible worlds $\mathcal{I}_{\mathcal{D}}$. It can be summarized using the following procedure:

Procedure 1 (Exact Approach for Answering Q1 and Q2)**.** *We iterate over all $D_i \in \mathcal{I}_{\mathcal{D}}$ and use it to train a classifier h_i. Depending on the query we are trying to answer, we perform the following steps:*

- Q1 – *check whether $h_i(\mathbf{x}) =$ is the same for all $D_i \in \mathcal{I}_{\mathcal{D}}$.*
- Q2 – *compute $\mathcal{D}_y$ for each $y \in \mathcal{Y}$ based on h_i and use $\mathcal{D}_y$ to generate answers based on different semantics of Q2.*

COMPUTATIONAL COMPLEXITY. It is clear that this approach is bounded by the size of the set of possible worlds $\mathcal{I}_{\mathcal{D}}$. Based on Definition 2 we can deduce that the complexity of answering both Q1 and Q2 is $O(M^N)$ for a dataset that has N data examples and at most M candidates per example.

3.5.2 Monte Carlo Sampling Algorithms

The basic idea is to take independent random samples from $\mathcal{D}$ and then estimate the tail probability (i.e., error rate) using techniques based on the central limit theorem (CLT).

Procedure 2 (Approximate Approach for Answering Q1 and Q2)**.** *We select p random samples $\mathcal{S} = \{D_1, D_2, ...D_p\}$ from $\mathcal{D}$ independently. We then train the corresponding classifier h_1, ..., h_p to make predictions for* $\mathbf{x}$.

We can answer Q1 and Q2 based on the results $\mathcal{H} = \{h_i(\mathbf{x})\}_{i=1}^{p}$:

- *Q1* – check whether $h_1(\mathbf{x}) = \cdots = h_p(\mathbf{x})$;
- *Q2* – compute $\mathcal{D}_y$ for each $y \in \mathcal{Y}$ based on $\mathcal{H}$ and use $\mathcal{D}_y$ to generate answers based on different semantics of Q2.

Clearly, if $\mathbf{x}$ is not consistently classifiable on $\mathcal{S}$, then it is also not consistently classifiable on $\mathcal{D}$. On the other hand, when $\mathbf{x}$ is consistently classifiable on $\mathcal{S}$, the following result guarantees that, with high probability and high confidence, $\mathbf{x}$ is also consistently classifiable on $\mathcal{D}$, as long as $\mathcal{S}$ is sufficiently large.

Theorem 1. *Let $\mathcal{S} = \{D_1, D_2, ...D_p\}$ be p independent samples from $\mathcal{D}$. If $\mathbf{x}$ can be consistently classified w.r.t $\mathcal{S}$ and h, then with probability at least $1 - \delta$, we have*

$$Pr(h_{D \overset{i.i.d}{\sim} \mathcal{D}}(\boldsymbol{x}) = \hat{y}) \geq 1 - \epsilon,$$

where $\epsilon = \sqrt{\frac{-\ln \delta}{2p}}$, $\hat{y} = h_{D \in \mathcal{S}}(\mathbf{x})$.

By Theorem 1, we provide some concrete examples regarding the sample size

$$p = \ln(1/\delta)/2\epsilon^2$$

for various ϵ and δ:

K	$\lvert\mathcal{Y}\rvert$	Query	Alg.	Complexity in $O(-)$	Section
1	2	Q1/Q2	SS	$NM \log NM$	3.1.2
K	2	Q1	MM	NM	3.2
K	$\lvert\mathcal{Y}\rvert$	Q1/Q2	SS	$NM(\log(NM) + K^2 \log N)$	3.1.3

FIGURE 3.5: Summary of results (K and $|\mathcal{Y}|$ are constants).

- $\epsilon = 0.1$ and $\delta = 0.1$, then $p = 116$;
- $\epsilon = 0.05$ and $\delta = 0.05$, then $p = 600$;
- $\epsilon = 0.01$ and $\delta = 0.01$, then $p = 23,205$.

3.5.3 PTIME Algorithms using the KNN Model

Given our definition of certain prediction, not surprisingly, both queries are hard if we do not assume any structure of the classifier. In this section, we focus on a specific classifier that is popularly used in practice, namely the K-Nearest Neighbor (KNN) classifier. As we will see, for a KNN classifier, we are able to answer both CP queries in *polynomial* time, even though we are reasoning over *exponentially* many possible worlds!

K-NEAREST NEIGHBOR (KNN) CLASSIFIERS. A textbook KNN classifier works in the following way, as illustrated in Figure 3.6(a): Given a training set $D = \{(x_i, y_i)\}$ and a test example t, we first calculate the similarity between t and each x_i: $s_i = \kappa(x_i, t)$. This similarity can be calculated using different kernel functions κ such as linear kernel, RBF kernel, etc. Given all these similarity scores $\{s_i\}$, we pick the top K training examples with the largest similarity score: $x_{\sigma_1}, ..., x_{\sigma_K}$ along with corresponding labels $\{y_{\sigma_i}\}_{i\in[K]}$. We then take the majority label among $\{y_{\sigma_i}\}_{i\in[K]}$ and return it as the prediction for the test example t.

KNN OVER INCOMPLETE DATASETS. When the KNN classifier makes predictions over an incomplete dataset $\mathcal{D}$, those predictions should (at least implicitly) consider the entire set of possible worlds $\mathcal{I}_\mathcal{D}$. Since each possible world $D \in \mathcal{I}_\mathcal{D}$ is made up independently chosen candidates of each tuple, we can compute the

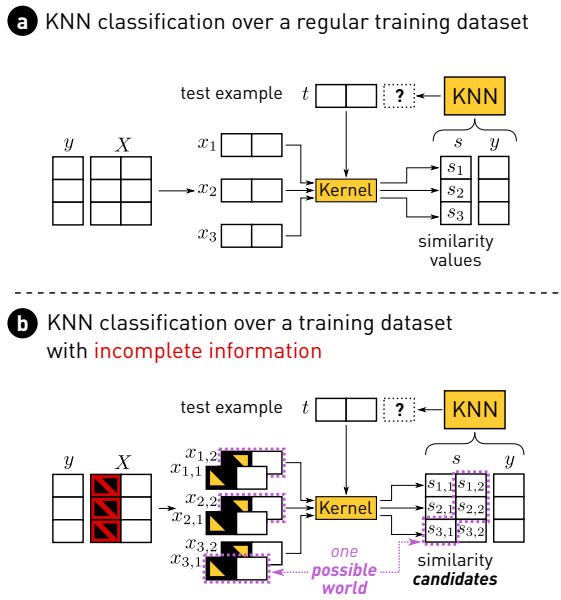

FIGURE 3.6: Illustration of KNN classifiers.

similarity score per each candiate. Therefore, we can see that in Figure 3.6(b) we have similarity candiates instead of similarity values. Our modified KNN algorithm then takes these similarity candidates and produces either a certain prediction if $Q1$ (according to Equation 3.1) would return return `true`. Otherwise, the modified KNN would return or NULL.

SUMMARY OF RESULTS. In this chapter, we focus on designing efficient algorithms to support a KNN classifier for both CP queries. In general, all these results are based on two algorithms, namely SS (SortScan) and MM (MinMax). SS is a generic algorithm that can be used to answer both queries, while MM can only be used to answer $Q1$. However, on the other hand, MM permits lower complexity than SS when applicable. Figure 3.5 summarizes the result.

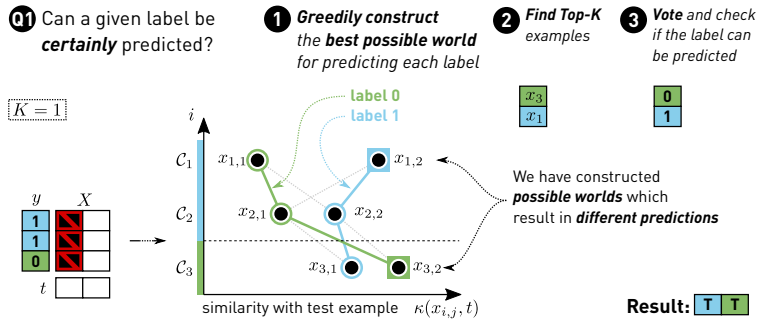

FIGURE 3.7: Illustration of MM when $K = 1$ for $Q1$.

3.6 Min-Max Algorithm for Answering the Checking Query

One can do significantly better for Q_1 in certain cases. Instead of using the SS algorithm, we can develop an algorithm that deals with the binary classification case ($|\mathcal{Y}| = 2$) with time complexity $\mathcal{O}\left(N \cdot M + (N \log K + K)\right)$.

This algorithm, illustrated in Figure 3.7, relies on a key observation that for each label l, we can greedily construct a possible world that has the best chance of predicting label l. We call this possible world the l-extreme world and construct it by selecting from each candidate set $\mathcal{C}_i$ either the candidate most similar to the test example t when $y_i = l$, or the candidate least similar to t when $y_i \neq l$. We can show that the l-extreme world predicts label l if and only if there exists a possible world that predicts label l. This means we can use it as a condition for checking the possibility of predicting label l. Since the construction of the l-extreme world can be done in $\mathcal{O}\left(N \cdot M\right)$ time, this leads us to a efficient algorithm for $Q1$.

We first describe the key idea behind the MM algorithm, and then describe the MM algorithm which is listed in Algorithm 1.

ALGORITHM OUTLINE. We are given an incomplete dataset $\mathcal{D} = \{(\mathcal{C}_i, y_i) : i = 1, ..., N\}$, a test data point $t \in \mathcal{X}$ and a class label $l \in \mathcal{Y}$. The MM algorithm answers the checking query $Q1(\mathcal{D}, t, l)$ for K-NN with similarity kernel κ by constructing the l-expreme possible world $E_{l,\mathcal{D}}$ defined as:

$$E_{l,\mathcal{D}} = \{(M_i, y_i) : (\mathcal{C}_i, y_i) \in \mathcal{D}\}$$
$$M_i = \begin{cases} \arg\max_{x_{i,j} \in \mathcal{C}_i} \kappa(x_{i,j}, t), & \text{if } y_i = l, \\ \arg\min_{x_{i,j} \in \mathcal{C}_i} \kappa(x_{i,j}, t), & \text{otherwise} \end{cases} \tag{3.4}$$

The answer to $Q1(\mathcal{D}, t, l)$ is obtained by checking if: (1) K-NN trained over $E_{l,\mathcal{D}}$ predicts l, and (2) for all other labels $l' \in \mathcal{Y} \setminus \{l\}$, K-NN trained over $E_{l',\mathcal{D}}$ does not predict l. Figure 3.7 depicts this algorithm for an example scenario.

KEY IDEA For *binary classification* ($|\mathcal{Y}| = 2$), we have the following observation — given a possible world $D = \{(x_{i,j_{i,D}}, y_i)\}$ that produces prediction $l \in \mathcal{Y}$ with a top-K set $Top(K, D, t)$, consider a different possible world, which we call the *l-extreme world* of D as $E_{l,D}$. In $E_{l,D}$, we replace, for all candidate sets with $y_i = l$, the $x_{i,j_{i,D}}$ candidate in D with the candidate in the candidate set $\mathcal{C}_i$ that is most similar to the test example

$$j_{i,E_{l,D}} = \arg\max_j \kappa(x_{i,j}, t)$$

and replace, for all candidate sets with $y_i \neq l$, the $x_{i,j_{i,D}}$ candidate in D with the candidate in the candidate set $\mathcal{C}_i$ that is least similar to the test example

$$j_{i,E_{l,D}} = \arg\min_j \kappa(x_{i,j}, t).$$

We have

$$D \text{ predicts } l \implies E_{l,D} \text{ predicts } l.$$

To see why, note that (1) replacing all candidate values for candidate set whose label $y_i \neq l$ by something less similar to the test example t will only make it more likely to predict l; (2) replacing all candidate values for candidate set whose label $y_i = l$ by something more similar to t will only make it more likely to predict l.

Another powerful observation is that for all possible worlds D they all have the *same* l-extreme worlds $E_{l,D}$ since the construction of the latter only relies on the most and least similar items in each candidate sets. We can then write E_l as the l-extreme world for all possible world D. We now have

$$\exists D.\ D \text{ predicts } l \implies E_l \text{ predicts } l,$$

Algorithm 1 Algorithm **MM** for answering Q1 with K-NN.

inputs
    $\mathcal{D}$, incomplete dataset;
    t, target data point.
outputs
    r, Boolean vector, s.t. $r[y] = Q1(\mathcal{D}, t, y), \forall y \in \mathcal{Y}$.
$S \leftarrow \texttt{kernel}(\mathcal{D}, x_t)$
for $i \in 1, ..., |\mathcal{D}|$ **do**
    $s_i^{\min} \leftarrow \min\{S_{i,j}\}_{j=1}^{M}$, $s_i^{\max} \leftarrow \max\{S_{i,j}\}_{j=1}^{M}$;
end for
for $l \in \mathcal{Y}$ **do**
    $s \leftarrow \texttt{zeros}(|\mathcal{D}|)$;
    for $i \in 1, ..., |\mathcal{D}|$ **do**
        $s[i] \leftarrow s_i^{\max}$ **if** $(y_i = l)$ **else** $s_i^{\min}$;
    end for
    $I_K \leftarrow \texttt{argmax\_k}(s, K)$;
    $v \leftarrow \texttt{vote}(\{y_i : i \in I_K\})$;
    if $\texttt{argmax}(v) = l$ **then**
        $r[l] \leftarrow \texttt{true}$;
    else
        $r[l] \leftarrow \texttt{false}$;
    end if
end for
return r;

and, trivially

$$E_l \text{ predicts } l \implies \exists D.\ D \text{ predicts } l,$$

by simply taking $D = E_l$. As a result,

$$\exists D.\ D \text{ predicts } l \Leftrightarrow E_l \text{ predicts } l.$$

One can use this observation to check whether $Q1(\mathcal{D}, t, l)$ evaluates to true: this is equivalent to checking whether there exists any possible world D that predicts a label $l' \neq l$. To achieve this, we can simply check the l'- extreme world $E_{l'}$.

PROOF IN APPENDIX. This idea might look simple and natural, however, a formal proof is actually quite engaged (e.g., without a formal proof, it is not immediately clear why this algorithm cannot handle cases in which $|\mathcal{Y}| > 2$). We leave the full, formal proof to the appendix of this thesis.

EFFICIENT ALGORITHM. The above intuition gives us a very efficient algorithm to answer the query $Q1(\mathcal{D}, t, l)$, as illustrated in Algorithm 1. We first calculate the similarity matrix (line 1), compute the extreme similarities that we use later (lines 2-3), and then try to construct the l-extreme world for each $l \in \mathcal{Y}$ (lines 4-7). We then calculate the top-K set of the l-extreme world (line 8) and tally the labels to get the prediction in the l-extreme world (line 9).

To answer the query $Q1(\mathcal{D}, t, l)$ (lines 10-13), we check all l'-extreme worlds where $l' \neq l$ to see if any of these l'-extreme worlds predicts their corresponding l'. If yes, then $Q1(\mathcal{D}, t, l) = \texttt{false}$; otherwise, $Q1(\mathcal{D}, t, l) = \texttt{true}$.

(Complexity) We analyze the complexity of Algorithm 1 as follows:

- We first precompute the similarity matrix, as well as the minimum and maximum similarities, both of which can be done in $\mathcal{O}(N \cdot M)$ time.
- The outer loop is executed $|\mathcal{Y}|$ times.
- The optimal world construction loop (lines 6-7) is executed N times. In each iteration we retrieve the precomputed maximal or minimal values.
- The `argmax_k` function implemented as a heap requires $\mathcal{O}(N \log K)$ steps.
- The `vote` function requires $\mathcal{O}(K)$ steps. The `argmax` takes $\mathcal{O}(|\mathcal{Y}|)$, although these two steps can be implemented jointly and run in $\mathcal{O}(K)$ time.

The time complexity is therefore $\mathcal{O}\left(N \cdot M + |\mathcal{Y}| \cdot (N \log K + K)\right)$.

3.7 Sort-Scan Algorithm for Answering the Counting Query

We now describe the SS algorithm. The idea behind SS is that we can calculate the similarity between all candidates $\cup_i \mathcal{C}_i$ in an incomplete dataset and a test example t. Without loss of generaility, assume that $|\mathcal{C}_i| = M$, this leads to $N \times M$ similarity scores $s_{i,j}$. We can then sort and scan these similarity scores.

The core of the SS algorithm is a dynamic programming procedure. We will first describe a set of basic building blocks of this problem, and then introduce a

simplified version of SS for the special case of $K = 1$ and $|\mathcal{Y}| = 2$, to explain the intuition behind SS. We follow this by the general version of the SS algorithm.

3.7.1 Two Building Blocks

In our problem, we can construct two building blocks efficiently. We start by articulating the settings precisely. In the next section, we will use these two building blocks for our SS algorithm.

SETUP. We are given an incomplete dataset $\mathcal{D} = \{(\mathcal{C}_i, y_i)\}$. Without loss of generality, we assume that each $\mathcal{C}_i$ only contains M elements, i.e., $|\mathcal{C}_i| = M$. We call $\mathcal{C}_i = \{x_{i,j}\}_{j\in[M]}$ the i^{th} incomplete data example, and $x_{i,j}$ the j^{th} candidate value for the i^{th} incomplete data example. This defines M^N many possible worlds:

$$\mathcal{I}_{\mathcal{D}} = \{D = \{(x_i^D, y_i^D)\} : |D| = |\mathcal{D}| \wedge y_i^D = y_i \wedge x_i^D \in \mathcal{C}_i\}.$$

We use $x_{i,j_{i,D}}$ to denote the candidate value for the i^{th} data point in D. Given a test example t, we can calculate the similarity between each candidate value $x_{i,j}$ and t: $s_{i,j} = \kappa(x_{i,j}, t)$. We call these values *similarity candidates*, as shown in Figure 3.6 (b). We assume that there are no ties in these similarities scores (we can always break a tie by favoring a smaller i and j or a pre-defined random order).

Furthermore, given a candidate value $x_{i,j}$, we count, for each candidate set, how many candidate values are less similar to the test example than $x_{i,j}$. This gives us what we call the *similarity tally* α. For each candidate set $\mathcal{C}_n$, we have

$$\alpha_{i,j}[n] = \sum\nolimits_{m=1}^{M} \mathbb{I}[s_{n,m} \leq s_{i,j}].$$

Example 1. *In Figure 3.8 we can see an example of a similarity tally $\alpha_{2,2}$ with respect to the data point $x_{2,2}$. For i^{th} incomplete data example, it contains the number of candidate values $x_{i,j} \in \mathcal{C}_i$ that have the similarity value no greater than $s_{2,2}$. Visually, in Figure 3.8, this represents all the candidates that lie left of the vertical yellow line. We can see that only one candidate from $\mathcal{C}_1$, two candidates from $\mathcal{C}_2$, and none of the candidates from $\mathcal{C}_3$ satisfy this property. This gives us $\alpha_{2,2}[1] = 1$, $\alpha_{2,2}[2] = 2$, and $\alpha_{2,2}[3] = 0$.*

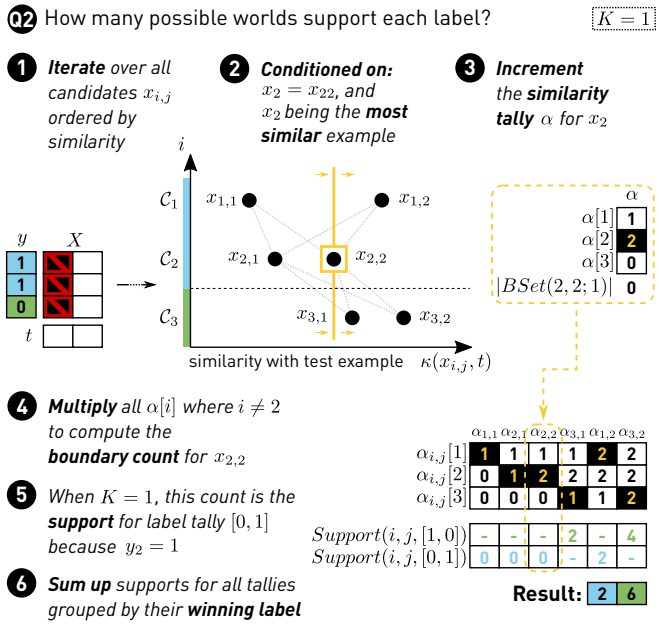

FIGURE 3.8: Illustration of SS when $K = 1$ for $Q2$.

KNN OVER A SINGLE POSSIBLE WORLD. Given one possible world D, running a KNN classifier to get the prediction for a test example t involves multiple stages. First, we obtain *Top-K Set*, the set of K examples in D that are in the K-nearest neighbor set

$$Top(K, D, t) \subseteq [N],$$

which has the following property

$$|Top(K, D, t)| = K,$$

$$\forall i, i' \in [N]. \quad i \in Top(K, D, t) \wedge i' \notin Top(K, D, t) \implies s_{i,j_{i,D}} > s_{i',j_{i',D}}.$$

Given the top-K set, we then tally the corresponding labels by counting how many examples in the top-K set support a given label. We call it the *label tally* γ^D:

$$\gamma^D \in \mathbb{N}^{|\mathcal{Y}|} : \gamma_l^D = \sum\nolimits_{i \in Top(K,D,t)} \mathbb{I}[l = y_i].$$

Finally, we pick the label with the largest count:

$$y_D^* = \arg\max_l \gamma_l^D.$$

Example 2. *For $K = 1$, the Top-K Set contains only one element x_i which is most similar to t. The label tally then is a $|\mathcal{Y}|$-dimensional binary vector with all elements being equal to zero except for the element corresponding to the label y_i being equal to one. Clearly, there are $|\mathcal{Y}|$ possible such label tally vectors.*

BUILDING BLOCK 1: BOUNDARY SET. The first building block answers the following question: *Out of all possible worlds that picked the value $x_{i,j}$ for $\mathcal{C}_i$, how many of them have $x_{i,j}$ as the least similar item in the Top-K set?* We call all possible worlds that satisfy this condition the *Boundary Set* of $x_{i,j}$:

$$BSet(i, j; K) = \{D : j_{i,D} = j \wedge i \in Top(K, D, t) \wedge i \notin Top(K - 1, D, t)\}.$$

We call the size of the boundary set the *Boundary Count*.

We can enumerate all $\binom{N}{(K-1)}$ possible configurations of the top-(K-1) set to compute the boundary count. Specifically, let $\mathcal{S}(K-1, [N])$ be all subsets of $[N]$ with size $K-1$. We have

$$|BSet(i, j; K)| = \sum_{S \in \mathcal{S}(K-1,[N])} \mathbb{1}\{i \notin S\} \cdot \left(\prod_{n \notin S} \alpha_{i,j}[n] \right) \cdot \left(\prod_{n \in S} (M - \alpha_{i,j}[n]) \right). \quad (3.5)$$

The idea behind this is the following — we enumerate all possible settings of the top-(K-1) set: $\mathcal{S}(K-1, [N])$. For each specific top-(K-1) setting S, every candidate set in S needs to pick a value that is more similar than $x_{i,j}$, while every candidate set not in S needs to pick a value that is less similar than $x_{i,j}$. Since the choices of value between different candidate sets are independent, we can calculate this by multiplying different entries of the similarity tally vector α.

We observe that calculating the boundary count for a value $x_{i,j}$ can be efficient when K is small. For example, if we use a 1-NN classifier, the only S that we consider is the empty set, and thus, the boundary count merely equals $\prod_{n \in [N], n \neq i} \alpha_{i,j}[n]$.

Example 3. *We can see this, in Figure 3.8 from Step 3 to Step 4, where the size of the boundary set $|BSet(2, 2; 1)|$ is computed as the product over elements of α, excluding $\alpha[2]$. Here, the boundary set for $x_{2,2}$ is actually empty. This happens because both candidates from $\mathcal{C}_3$ are more similar to t than $x_{2,2}$ is, that is, $\alpha_{2,2}[3] = 0$. Consequently, since every possible world must contain one element from $\mathcal{C}_3$, we can see that $x_{2,2}$ will never be in the Top-1, which is why its boundary set contains zero elements.*

If we had tried to construct the boundary set for $x_{3,1}$, we would have seen that it contains two possible worlds. One contains $x_{2,1}$ and the other contains $x_{2,2}$, because both are less similar to t than $x_{3,1}$ is, so they cannot interfere with its Top-1 position. On the other hand, both possible worlds have to contain $x_{1,1}$ because selecting $x_{1,2}$ would prevent $x_{3,1}$ from being the Top-1 example.

BUILDING BLOCK 2: LABEL SUPPORT. To get the prediction of a KNN classifier, we can reason about the label tally vector γ, and not necessarily the specific configurations of the top-K set. It answers the following question: *Given a specific configuration of the label tally vector γ, how many possible worlds in the boundary set of $x_{i,j}$ support this γ?* We call this the *Support* of the label tally vector γ:

$$Support(i, j, \gamma) = |\{D : \gamma^D = \gamma \wedge D \in BSet(i, j; K)\}|.$$

Example 4. *For example, when $K = 3$ and $|\mathcal{Y}| = 2$, we have 4 possible label tallies: $\gamma \in \{[0, 3], [1, 2], [2, 1], [3, 0]\}$. Each tally defines a distinct partition of the boundary set of $x_{i,j}$ and the size of this partition is the support for that tally. Note that one of these tallies always has support 0, which happens when $\gamma_l = 0$ for the label $l = y_i$, thus excluding $x_{i,j}$ from the top-K set.*

For $K = 1$, a label tally can only have one non-zero value that is equal to 1 only for a single label l. Therefore, all the elements in the boundary set of $x_{i,j}$ can support only one label tally vector that has $\gamma_l = 1$ where $l = y_i$. This label tally vector will always have the support equal to the boundary count of $x_{i,j}$.

Calculating the support can be done with dynamic programming. First, we can partition the whole incomplete dataset into $|\mathcal{Y}|$ many subsets, each of which only contains incomplete data points (candidate sets) of the same label $l \in |\mathcal{Y}|$:

$$\mathcal{D}_l = \{(\mathcal{C}_i, y_i) : y_i = l \wedge (\mathcal{C}_i, y_i) \in \mathcal{D}\}.$$

Clearly, if we want a possible world D that supports the label tally vector γ, its top-K set needs to have γ_1 candidate sets from $\mathcal{D}_1$, γ_2 candidate sets from $\mathcal{D}_2$, and so on. *Given that $x_{i,j}$ is on the boundry, how many ways do we have to pick γ_l many*

candidate sets from $\mathcal{D}_l$ in the top-K set? We can represent this value as $C_l^{i,j}(\gamma_l, N)$, with the following recursive structure:

$$C_l^{i,j}(c,n) = \begin{cases} C_l^{i,j}(c, n-1), & \text{if } y_n \neq l, \\ C_l^{i,j}(c-1, n-1), & \text{if } x_n = x_i, \text{otherwise} \\ \alpha_{i,j}[n] \cdot C_l^{i,j}(c, n-1) + (M - \alpha_{i,j}[n]) \cdot C_l^{i,j}(c-1, n-1). \end{cases}$$

This recursion defines a process in which one scans all candidate sets from $(\mathcal{C}_1, y_1)$ to $(\mathcal{C}_N, y_N)$. At candidate set $(\mathcal{C}_n, y_n)$:

1. If y_n is not equal to our target label l, the candidate set $(\mathcal{C}_n, y_n)$ will not have any impact on the count.

2. If x_n happens to be x_i, this will not have any impact on the count as x_i is always in the top-K set, by definition. However, this means that we have to decrement the number of available slots c.

3. Otherwise, we have two choices to make:

   a) Put $(\mathcal{C}_n, y_n)$ into the top-K set, and there are $(M - \alpha_{i,j}[n])$ many possible candidates to choose from.

   b) Do not put $(\mathcal{C}_n, y_n)$ into the top-K set, and there are $\alpha_{i,j}[n]$ many possible candidates to choose from.

It is clear that this recursion can be computed as a dynamic program in $\mathcal{O}(N \cdot M)$ time. This DP is defined for $c \in \{0...K\}$ which is the exact number of candidates we want to have in the top-K, and $n \in \{1...N\}$ which defines the subset of examples $x_i : i \in \{1...N\}$ we are considering. The boundary conditions of this DP are $C_l^{i,j}(-1, n) = 0$ and $C_l^{i,j}(c, 0) = 1$.

Given the result of this dynamic programming algorithm for different values of l, we can calculate the support of label tally γ:

$$Support(i, j, \gamma) = \prod\nolimits_{l \in \mathcal{Y}} C_l^{i,j}(\gamma_l, N),$$

which can be computed in $\mathcal{O}(NM|\mathcal{Y}|)$.

Example 5. *If we assume the situation shown in Figure 3.8, we can try for example to compute the value of $Support(3, 1, \gamma)$ where $\gamma = [1, 0]$. We would have*

$C_0^{3,1}(1, N) = 1$ *because* x_3 *(the subset of* D *with label 0) must be in the top-*K*, which happens only when* $x_3 = x_{3,1}$*. On the other hand we would have* $C_1^{3,1}(0, N) = 2$ *because both* x_1 *and* x_2 *(the subset of* D *with label 1) must be out of the top-*K*, which happens when* $x_1 = x_{1,1}$ *while* x_2 *can be either equal to* $x_{2,1}$ *or* $x_{2,2}$*. Their mutual product is equal to* 2*, which we can see below the tally column under* $x_{3,1}$*.*

3.7.2 Simple Case: 1-NN Algorithm for Binary Classification

Given the above two building blocks, it is easy to develop an algorithm for the case $K = 1$ and $|\mathcal{Y}| = 2$. In SS, we use the result of $Q2$ to answer both $Q1$ and $Q2$. Later we will introduce the MM algorithm that is dedicated to $Q1$ only.

We simply compute the number of possible worlds that support the prediction label being 1. We do this by enumerating all possible candidate values $x_{i,j}$. If this candidate has label $y_i = 1$, we count how many possible worlds have $x_{i,j}$ as the top-1 example, i.e., the boundry count of $x_{i,j}$. We have

$$Q2(\mathcal{D}, t, l) = \sum_{i\in[N]} \sum_{j\in[M]} \mathbb{I}[y_i = l] \cdot |BSet(i, j; K = 1)|,$$

which simplifies to

$$Q2(\mathcal{D}, t, l) = \sum_{i\in[N]} \sum_{j\in[M]} \mathbb{I}[y_i = l] \cdot \prod_{n\in[N], n\neq i} \alpha_{i,j}[n].$$

If we pre-compute the whole α matrix, it is clear that a naive implementation would calculate the above value in $\mathcal{O}(N^2M)$. However, as we will see later, we can do much better.

EFFICIENT IMPLEMENTATION. We can design a much more efficient algorithm to calculate this value. The idea is to first sort all $x_{i,j}$ pairs by their similarity to t, $s_{i,j}$, from the smallest to the largest, and then scan them in this order. In this way, we can incrementally maintain the $\alpha_{i,j}$ vector during the scan.

Let (i, j) be the current candidate value being scanned, and (i', j') be the candidate value right before (i, j) in the sort order, we have

$$\alpha_{i,j}[n] = \begin{cases} \alpha_{i',j'}[n] + 1 & \text{if } n = i', \\ \alpha_{i',j'}[n]. \end{cases} \tag{3.6}$$

Therefore, we are able to compute, for each (i, j), its

$$\prod_{n \in [N], n \neq i} \alpha_{i,j}[n] \tag{3.7}$$

in $\mathcal{O}(1)$ time, *without pre-computing the whole* α. This will give us an algorithm with complexity $\mathcal{O}(MN \log MN)$!

Example 6. *In Figure 3.8 we depict exactly this algorithm. We iterate over the candidates* $x_{i,j}$ *in an order of increasing similarity with the test example* t *(Step 1). In each iteration we try to compute the number of possible worlds supporting* $x_{i,j}$ *to be the top-*1 *data point (Step 2). We update the tally vector* α *according to Equation 3.6 (Step 3) and multiply its elements according to Equation 3.7 (Step 4) to obtain the boundary cont. Since* $K = 1$*, the label support for the label* $l = y_i$ *is trivially equal to the boundary count and zero for* $l \neq y_i$ *(Step 5). We can see that the label* 0 *is supported by* 2 *possible worlds when* $x_3 = x_{3,1}$ *and* 4 *possible worlds when* $x_3 = x_{3,2}$*. On the other hand, label* 1 *has non-zero support only when* $x_1 = x_{1,2}$*. Finally, the number of possible worlds that will predict label* l *is obtained by summing up all the label supports in each iteration where* $l = y_i$ *(Step 6). For label* 0 *this number is* $2 + 4 = 6$*, and for label* 1 *it is* $0 + 0 + 0 + 2 = 2$*.*

3.7.3 General Case: K-NN algorithm for Multi-Class Classification

In the general case, the algorithm follows a similar intuition as the case of $K = 1$ and $|\mathcal{Y}| = 2$. We enumerate each possible candidate value $x_{i,j}$. For each candidate value, we enumerate all possible values of the label tally vector; for each such vector, we compute its support. Let Γ be the set of all possible label tally vectors, we have

$$Q2(\mathcal{D}, t, l) = \sum_{i \in [N]} \sum_{j \in [M]} \sum_{\gamma \in \Gamma} \mathbb{I}[l = \arg\max(\gamma)] \cdot \mathit{Support}(i, j, \gamma).$$

We know that there are $|\Gamma| = \mathcal{O}\left(\binom{|\mathcal{Y}|+K-1}{K}\right)$ many possible configurations of the label tally vector, and for each of them, we can compute the support $\mathit{Support}(i, j, \gamma)$

Algorithm 2 Algorithm **SS** for Answering Q2 with K-NN.

1: **inputs**
2: $\mathcal{D}$, incomplete dataset;
3: t, target data point.
4: **outputs**
5: r, integer vector, s.t. $r[y] = Q2(\mathcal{D}, t, y), \forall y \in \mathcal{Y}$.
6: $s \leftarrow \texttt{kernel}(\mathcal{D}, t)$;
7: $\alpha \leftarrow zeros(|\mathcal{D}|)$;
8: $r \leftarrow zeros(|\mathcal{Y}|)$;
9: **for** $(i, j) \in \texttt{argsort}(s)$ **do**
10: $\alpha[i] \leftarrow \alpha[i] + 1$; // (See Equation (3.6))
11: **for** $l \in \mathcal{Y}$ **do**
12: **for** $k \in [K]$ **do**
13: Compute $C_l^{i,j}(k, N)$.
14: **end for**
15: **end for**
16: **for** possible valid tally vectors $\gamma \in \Gamma$ **do**
17: $y_p \leftarrow \texttt{argmax}(\gamma)$;
18: Compute $Support(i, j, \gamma) = \mathbb{I}[\gamma_{y_i} \geq 1] \cdot \prod_{l \in \mathcal{Y}} C_l^{i,j}(\gamma_l, N)$;
19: $r[y_p] \leftarrow r[y_p] + Support(i, j, \gamma)$;
20: **end for**
21: **end for**
22: **return** r;

in $\mathcal{O}(NM|\mathcal{Y}|)$ time. As a result, a naive implementation of the above algorithm would take $\mathcal{O}(N^2M^2|\mathcal{Y}|\binom{|\mathcal{Y}|+K-1}{K})$ time.

EFFICIENT IMPLEMENTATION. We can implement the above procedure in a more efficient way, as illustrated in Algorithm 2. Similar to the case of $K = 1$, we iterate over all values $x_{i,j}$ in the order of increasing similarity (line 4). This way, we are able to maintain, efficiently, the similarity tally vector $\alpha_{i,j}$ (line 5). We then pre-compute the result of $K|\mathcal{Y}|$ many dynamic programming procedures (lines 6-8), which will be used to compute the support for each possible tally vector later. We iterate over all valid label tally vectors, where a valid tally vector $\gamma \in \Gamma$ contains all integer vectors whose entries sum up to K (line 9). For each tally vector, we get its prediction y_p (line 10). We then calculate its support (line 11) and add it to the number of possible worlds with y_p as the prediction (line 12).

(Complexity) We analyze the complexity of Algorithm 2:

- The sorting procedure requires $\mathcal{O}(N \cdot M \log N \cdot M)$ steps as it sorts all elements of S.
- The outer loop iterates over $\mathcal{O}(N \cdot M)$ elements.
- In each inner iteration, we need to compute $|\mathcal{Y}|$ sets of dynamic programs, each of which has a combined state space of size $N \cdot K$.
- Furthermore, in each iteration, we iterate over all possible label assignments, which requires $\mathcal{O}\left(\binom{|\mathcal{Y}|+K-1}{K}\right)$ operations.
- For each label assignment, we need $\mathcal{O}(|\mathcal{Y}|)$ multiplications.

The time complexity is therefore the sum of $\mathcal{O}(N \cdot M \log(N \cdot M))$ and $\mathcal{O}(N \cdot M \cdot (N \cdot K + |\mathcal{Y}| + \binom{|\mathcal{Y}|+K-1}{K}))$.

3.8 Optimizations for the Sort-Scan Algorithm

We can make this even faster by observing that: (1) all the states relevant for each iteration of the outer loop are stored in α, and (2) between two iterations, only one element of α is updated. We can take advantage of these observations to reduce the cost of computing the dynamic program by employing divide-and-conquer. We recursively divide the elements of α into two subsets and maintain the DP result for each subset. The joint result for the two subsets is obtained by a simple sum-of-products formula with $O(K)$ complexity. We can see that this enables us to maintain a binary tree structure of DP results and in each iteration we need to update $O(\log N)$ elements. This enables us to compute the dynamic program in $O(K \log N)$ instead of $O(KN)$ time, which renders the overall complexity as $\mathcal{O}(N \cdot M \cdot (\log(N \cdot M) + K^2 \cdot \log N))$.

3.8.1 Optimization Using Divide and Conquer

ALGORITHM OUTLINE. This version of the algorithm is almost identical to the original SS algorithm described previously, except for the way it computes

the label support. Namely, in the original algorithm we were using the dynamic program $C_l^{i,j}(c,n)$ to return the number of possible worlds in the boundary set $BSet(i,j;K)$ that support having exactly c examples in the top-K. Here, the parameter $n \in \{1...N\}$ denoted that we were only considering the subset of candidate sets $\mathcal{C}_i$ where $i \in \{1...n\}$.

If we observe Algorithm 2, we can see that the dynamic program $C_l^{i,j}(c,n)$ is re-computed in every iteration of the outer loop. However, at the same time we can see that the similarity tally α, which is used to compute the dynamic program, gets only one of its elements updated. To take advantage of that, we apply a divide-and-conquer strategy and redefine the recurrence relation as a tree structure:

$$\begin{aligned} &T_l^{i,j}(c,a,b) = \sum_{k=0}^{c} T_l^{i,j}(k,a,m) \cdot T_l^{i,j}(c-k,a+1,b), \\ &\text{where } m = \lfloor \frac{a+b}{2} \rfloor. \end{aligned} \tag{3.8}$$

To efficiently maintain the dynamic program $T_l^{i,j}$ across iterations over (i,j), we organize it in a binary tree structure. Each node, denoted as $n_{a,b}$, contains a list of values of $T_l^{i,j}(c,a,b)$ for all $c \in \{0...K\}$. Its two children are $n_{a,m}$ and $n_{m+1,b}$ where $m = \lfloor \frac{a+b}{2} \rfloor$. The leaves are nodes $n_{a,a}$ with both indices equal, which get evaluated according to the following base conditions:

1. $T_l^{i,j}(c,a,a) = 1$, if $y_a \neq l$;
   Rationale: Skip examples with label different from l.

2. $T_l^{i,j}(0,i,i) = 0$ and $T_l^{i,j}(1,i,i) = 1$;
   Rationale: The i-th example must be in the top-K, unless it got skipped.

3. $T_l^{i,j}(0,a,a) = \alpha[a]$;
   Rationale: If the a-th example is in the top-K, there are $\alpha[a]$ candidates to choose from.

4. $T_l^{i,j}(1,a,a) = M - \alpha[a]$;
   Rationale: If the a-th example is not in the top-K, there are $M - \alpha[a]$ candidates to choose from.

5. $T_l^{i,j}(c, a, a) = 0$, if $c \notin \{0, 1\}$
   Rationale: Invalid case because an example can either be $(c = 1)$ or not be $(c = 0)$ in the top-K.

The leaf nodes $n_{a,a}$ of this tree correspond to label support coming from individual data examples. The internal nodes $n_{a,b}$ correspond to the label support computed over all leaves in their respective sub-trees. This corresponds to data examples with index $i \in \{a...b\}$. The root node $n_{1,N}$ contains the label support computed over all data examples.

Since between any two consecutive iterations of (i, j) in Algorithm 2 we only update the i-th element of the similarity tally α, we can notice that out of all leaves in our binary tree, only $n_{i,i}$ gets updated. This impacts only $\mathcal{O}(N)$ internal nodes which are direct ancestors to that leaf. If we update only those nodes, we can avoid recomputing the entire dynamic program. The full algorithm is listed in Algorithm 3.

COMPLEXITY. We analyze the complexity of Algorithm 3:

- The sorting procedure requires $\mathcal{O}(N \cdot M \log N \cdot M)$ steps as it sorts all elements of S.
- The tree initialization can be performed eagerly in $\mathcal{O}(KN)$ time, or lazily in constant amortized time.
- The outer loop iterates over $\mathcal{O}(N \cdot M)$ elements.
- In each inner iteration, we update $\mathcal{O}(\log N)$ nodes. Each node maintains support values for all $c \in \{1...K\}$ and each one takes $\mathcal{O}(K)$ to recompute. Therefore, the tree update can be performed in $\mathcal{O}(K^2 \log N)$ time.
- Furthermore, in each iteration, we iterate over all possible label assignments, which requires $\mathcal{O}\left(\binom{|\mathcal{Y}|+K-1}{K}\right)$ operations.
- For each label assignment, we need $\mathcal{O}(|\mathcal{Y}|)$ multiplications.

This renders the final complexity to be the sum of $\mathcal{O}(N \cdot M \cdot (\log(N \cdot M) + K^2 \cdot \log N))$ and $\mathcal{O}(N \cdot M \cdot (|\mathcal{Y}| + \binom{|\mathcal{Y}|+K-1}{K}))$. When $|\mathcal{Y}|$ and K are relatively small constants, this reduces to $\mathcal{O}(N \cdot M \cdot \log(N \cdot M))$.

Algorithm 3 Algorithm **SS-DC** for Q2 with K-NN.

inputs
    $\mathcal{D}$, incomplete dataset;
inputs t, target data point.
outputs
    r, integer vector, s.t. $r[y] = Q2(\mathcal{D}, t, y), \forall y \in \mathcal{Y}$.
$s \leftarrow \texttt{kernel}(\mathcal{D}, t)$;
$\alpha \leftarrow zeros(|\mathcal{D}|)$;
$r \leftarrow zeros(|\mathcal{Y}|)$;
for $l \in \mathcal{Y}$ **do**
    for $k \in [K]$ **do**
        Initialize the tree $T_l(k, 1, N)$.
    end for
end for
for $(i, j) \in \texttt{argsort}(s)$ **do**
    $\alpha[i] \leftarrow \alpha[i] + 1$;
    Update the leaf node $T_{y_i}(c, i, i)$ and its ancestors for all $c \in \{1...K\}$.
    for possible valid tally vectors $\gamma \in \Gamma$ **do**
        $y_p \leftarrow \texttt{argmax}(\gamma)$;
        Compute $Support(i, j, \gamma) = \mathbb{I}[\gamma_{y_i} \geq 1] \cdot \prod_{l \in \mathcal{Y}} T_l(\gamma_l, 1, N)$;
        $r[y_p] \leftarrow r[y_p] + Support(i, j, \gamma)$;
    end for
end for
return r;

3.8.2 Optimization for Multiple Invocations

The goal of this optimization is to efficiently handle the scenario described in Section 3.9 where we introduce our **CPClean** data repair method. In this method we apply invocations to $Q2$ multiple times in order to compute the information gain after cleaning individual data examples. Specifically, given an incomplete training dataset $\mathcal{D}$ with N candidate sets each consisting of up to M possible candidate feature vectors, we iterate over all $n \in \{1...N\}$ and $m \in \{1...M\}$ and compute $Q2$ for an altered incomplete dataset $\mathcal{D}^{(n,m)}$ obtained from $\mathcal{D}$ by replacing the candidate set $\mathcal{C}_n = \{x_{n,1}, x_{n,2}, ...\}$ with $\mathcal{C}_n^{(n,m)} = \{x_{n,m}\}$. Consequently, we end up invoking the same algorithm $\mathcal{O}(N \cdot M)$ times with little change in $\mathcal{D}'$ between consecutive invocations.

ALGORITHM OUTLINE. This version of the algorithm takes advantage of the structure of the dynamic program $C_l^{i,j}(c, n)$. Namely, that $C_l^{i,j}(c, n)$ contains the support for label l given that $x_i = x_{i,j}$ computed over the first n data examples (i.e. over all $\mathcal{C}_i$ where $i \in \{1...n\}$). We notice that if we simply reverse the order by which we visit the data examples, we can end up with a dynamic program $\bar{C}_l^{i,j}(c, n)$ which is computed over all $\mathcal{C}_i$ where $i \in \{n...N\}$. Here we have an equivalence between $C_l^{i,j}(c, N)$ and $\bar{C}_l^{i,j}(c, 1)$.

Furthermore, since in our scenario we are dealing with the incomplete dataset $\mathcal{D}^{(n,m)}$, we know that the vector α we use in our SS algorithm will contain at position n one of two possible values $\alpha[n] = 0$ or $\alpha[n] = 1$, depending on whether or not $x_{i,j}$ comes before $x_{n,m}$ in the similarity-based sort order.

Putting these things together, we notice that we can compute $C_l^{i,j}(c, N)$ for $\mathcal{D}^{(n,m)}$ as follows:

$$C_l^{i,j}(c, N) = \sum_{k=0}^{b(c)} C_l^{i,j}(k, n-1) \cdot \bar{C}_l^{i,j}(b(c) - k, n+1)$$

Given that we have c slots to fill, we are partitioning $\mathcal{D}^{(n,m)}$ into data examples with index strictly less than n and data examples with index strictly greater than n. We then simply consider all possible ways to split the available slots among these two partitions. When computing $C_l^{i,j}$, we know $x_{i,j}$ is the K-th most similar data example and $\mathcal{C}_n^{(n,m)} = \{x_{n,m}\}$ has only one candidate. Therefore, we know that the n-th data example is either surely going to be in the top-K, or it is surely going to

Algorithm 4 Algorithm **SS-BDP** for Q2 with K-NN.

inputs
  $\mathcal{D}'$, incomplete dataset obtained from $\mathcal{D}$ by replacing $\mathcal{C}_n$ with $\mathcal{C}'_n = \{x_{n,m}\}$;
  t, target data point;
  precomputed DP matrices $C_l^{i,j}$ and $\bar{C}_l^{i,j}$ for all $i \in \{1...N\}$, $j \in \{1...M\}$ and $l \in \mathcal{Y}$.
outputs
  r, integer vector, s.t. $r[y] = Q2(\mathcal{D}, t, y), \forall y \in \mathcal{Y}$.
$s \leftarrow \texttt{kernel}(\mathcal{D}, t)$;
$r \leftarrow zeros(|\mathcal{Y}|)$;
for $(i, j) \in \texttt{argsort}(s)$ **do**
  for $l \in \mathcal{Y}$ **do**
    for $k \in [K]$ **do**
      $b \leftarrow k - \mathbb{I}[s_{i,j} \leq s_{n,m}]$;
      $\hat{C}_l^{i,j}(k) \leftarrow \sum_{c=0}^{b} C_l^{i,j}(c, n-1) \cdot \bar{C}_l^{i,j}(b-c, n+1)$;
    end for
  end for
  for possible valid tally vectors $\gamma \in \Gamma$ **do**
    $y_p \leftarrow \texttt{argmax}(\gamma)$;
    Compute $Support(i, j, \gamma) = \mathbb{I}[\gamma_{y_i} \geq 1] \cdot \prod_{l \in \mathcal{Y}} \hat{C}_l^{i,j}(\gamma_l)$;
    $r[y_p] \leftarrow r[y_p] + Support(i, j, \gamma)$;
  end for
end for
return r;

be out, depending on whether its similarity score is higher than the one of $x_{i,j}$. We use $b(c) = c - \mathbb{I}[s_{i,j} \leq s_{n,m}]$ to represent our budget of available slots to fill given the mentioned condition.

The strategy of our algorithm is to precompute the dynamic programs $C_l^{i,j}$ and $\bar{C}_l^{i,j}$ for all $i \in \{1...N\}$, $j \in \{1...M\}$ and $l \in \mathcal{Y}$. Then we use these precomputed dynamic programs to efficiently compute the label support. The full algorithm, which we refer to as *bidirectional dynamic program* (BDP), is listed in Algorithm 4.

COMPLEXITY. In terms of complexity, Algorithm 4 makes savings by avoiding the computation of the dynamic programs. Each invocation therefore takes $\mathcal{O}(N \cdot M \log(N \cdot M))$ assuming K and $|\mathcal{Y}|$ are relatively small constants. If we use caching to avoid sorting the similarity matrix each time, we end up with complexity $\mathcal{O}(N \cdot$

M). Of course, precomputing the dynamic programs still takes as much time as for Algorithm 2, which is $\mathcal{O}(N \cdot M \log(N \cdot M) + N^2 \cdot M)$.

However, the real savings come with multiple invocations. Namely, to estimate information gain, we need to invoke the same algorithm $\mathcal{O}(N \cdot M)$ times, which would make take $\mathcal{O}(N^2 \cdot M^2 \log(N \cdot M) + N^3 \cdot M^2)$ time without our optimization. With our optimization it takes only $\mathcal{O}(N^2 \cdot M^2)$ time.

3.8.3 Polynomial Time Solution for Multi-Class Classification

We have seen that the previously described version of the SS algorithm gives an efficient polynomial solution for $Q2$, but only for a relatively small number of classes $|\mathcal{Y}|$. When $|\mathcal{Y}| \gg 1$, the $\mathcal{O}(\binom{|\mathcal{Y}|+K-1}{K})$ factor of the complexity starts to dominate. For a very large number of classes (which is the case for example in the popular ImageNet dataset), running this algorithm becomes practically infeasible. In this section we present a solution for $Q2$ which is polynomial in $|\mathcal{Y}|$.

The main source of computational complexity in Algorithm 3 is in the for-loop starting at line 10. Here we iterate over all possible tally vectors γ, and for each one we compute the label tally support $Support(i, j, \gamma)$ (line 12) and add it to the resulting sum (line 13) which is selected according to the winning label with the largest tally in γ (line 11).

The key observation is that, for l to be the winning label, one only needs to ensure that no other label has a larger label tally. In other words, label l will be predicted whenever $\gamma_l > \gamma_{l'}$ for all $l \neq l'$, regardless of the actual tallies of all l'. Therefore, we found that we can group all the label tally vectors according to this predicate. To achieve this, we define the following recurrence:

$$D_{Y,c}(j,k) = \sum_{n=0}^{\min\{c,k\}} T_{Y_j}(n,1,N) \cdot D_{Y,c}(j+1, k-l). \tag{3.9}$$

Here Y is the list of all labels in $\mathcal{Y} \setminus \{l\}$ and $T^{i,j}_{Y_j}(n,1,N)$ is the label support for label Y_j, as described in the previous section. The semantics of $D_{Y,c}(j,k)$ is the number of possible worlds where the top-K contains at most k examples with labels $l' \in Y_{0\ldots j}$ and no label has tally above c. We can see that $D_{Y,c}(j,k)$ can

also be computed as a dynamic program with base conditions $D_{Y,c}(|Y|, 0) = 1$ and $D_{Y,c}(|Y|, k) = 0$ for $k > 0$.

Algorithm 5 Algorithm **SS-DC-MC** for Q2 with K-NN.

1: **inputs**
2: $\mathcal{D}$, incomplete dataset; t, target data point.
3: **outputs**
4: r, integer vector, s.t. $r[y] = Q2(\mathcal{D}, t, y), \forall y \in \mathcal{Y}$.
5: $s \leftarrow \texttt{kernel}(\mathcal{D}, t)$;
6: $\alpha \leftarrow zeros(|\mathcal{D}|)$;
7: $r \leftarrow zeros(|\mathcal{Y}|)$;
8: **for** $l \in \mathcal{Y}$ **do**
9: **for** $k \in [K]$ **do**
10: Initialize the tree $T_l(k, 1, N)$.
11: **end for**
12: **end for**
13: **for** $(i, j) \in \texttt{argsort}(s)$ **do**
14: $\alpha[i] \leftarrow \alpha[i] + 1$;
15: Update the leaf node $T_{y_i}(c, i, i)$ and its ancestors for all $c \in \{1...K\}$.
16: **for** $l \in \mathcal{Y}$ **do**
17: $Y \leftarrow [\mathcal{Y} \setminus \{l\}]$;
18: **for** $c \in \{1...K\}$ **do**
19: Compute $D_{Y,c}(|\mathcal{Y}| - 1, K - c - 1)$ using dynamic programming;
20: $r[y_p] \leftarrow r[y_p] + T_l(c, 1, N) \cdot D_{Y,c}(|\mathcal{Y}| - 1, K - c - 1)$;
21: **end for**
22: **end for**
23: **end for**
24: **return** r;

In terms of performance, the complexity of Algorithm 5, compared to Algorithm 3 has one more major source of time complexity, which is the computation of the dynamic program $D_{Y,c}$ which takes $\mathcal{O}(|\mathcal{Y}| \cdot K^2)$ time. Since the for loops in lines 10 and 12 take $O(|\mathcal{Y}|)$ and $O(K)$ time respectively, the overall complexity of the algorithm becomes $O(M \cdot N \cdot (\log(M \cdot N) + K^2 \log N + |\mathcal{Y}|^2 \cdot K^3)$.

3.9 The CPClean Algorithm

In this section, we switch from the algorithmic framework for answering CP queries to constructing the algorithms for solving the data cleaning problem as defined in Section 3.4. Finding the solution to this problem is, not surprisingly, NP-hard [61]. Here we take the view of sequential information maximization introduced by [40] and adapt the respective greedy algorithm for this problem. We first describe the algorithm, and then review the theoretical analysis of its behavior.

PRINCIPLE: SEQUENTIAL INFORMATION MAXIMIZATION. Our goal is to find a cleaning strategy that *minimizes* the *conditional entropy* as fast as possible. An equivalent view of this is to find a cleaning strategy that *maximizes* the *mutual information* as fast as possible.

Given the current T-step cleaning strategy $\pi_1, ..., \pi_T$, our goal is to greedily find the *next* data example to clean $\pi_{T+1} \in [N]$ that minimizes the entropy conditioned on the partial observation as fast as possible:

$$\pi_{T+1} = \arg\min_{i \in [N]} \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val}) | \mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}, \mathbf{c}_i = x^*_i).$$

PRACTICAL ESTIMATION. The question thus becomes how to estimate

$$\mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val}) | \mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}, \mathbf{c}_i = x^*_i)?$$

The challenge is that when we are trying to decide which example to clean, we do not know the ground truth for item i, x^*_i. As a result, we need to assume some priors on how likely each candidate value $x_{i,j}$ is the ground truth x^*_i. In practice, we find that a uniform prior already works well; this leads to the following expected value:

$$\frac{1}{M} \sum_{j \in [M]} \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val}) | \mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}, \mathbf{c}_i = x_{i,j}). \tag{3.10}$$

The above term can thus be calculated by invoking the $Q2$ query.

CPCLEAN. The pseudocode for CPClean is shown in Algorithm 6. The algorithm starts with an empty cleaning strategy (line 1). In each iteration, given the current cleaning strategy $\pi_1, ..., \pi_T$, we compute the expected value of entropy conditioned

Algorithm 6 Algorithm CPClean.

1: **inputs**
2: $\mathcal{D}$, incomplete training set;
3: D_{val}, validation set.
4: **outputs**
5: D, a dataset in $\mathcal{I}_{\mathcal{D}}$ s.t. $\mathcal{A}_D$ and $\mathcal{A}_{D^*}$ have same validation accuracy
6: $\pi \leftarrow []$
7: **for** $T = 0$ to $N - 1$ **do**
8: **if** D_{val} all CP'ed **then**
9: **break**
10: **end if**
11: $min\_entropy \leftarrow \infty$
12: **for** $i \in [N] \backslash \pi$ **do**
13: $entropy = \frac{1}{M} \sum_{j \in [M]} \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{val}) | \mathbf{c}_{\pi_1} = x^*_{\pi_1}, ..., \mathbf{c}_{\pi_T} = x^*_{\pi_T}, \mathbf{c}_i = x_{i,j})$
14: **if** $entropy < min\_entropy$ **then**
15: $\pi_{T+1} \leftarrow i$, $min\_entropy \leftarrow entropy$
16: **end if**
17: **end for**
18: $\mathbf{x}^*_{\pi_{T+1}} \leftarrow$ obtain the ground truth of $\mathcal{C}_{\pi_{T+1}}$ by human
19: **end for**
20: **return** Any world $D \in \mathcal{I}_{\mathcal{D}_\pi}$

on cleaning one extra training example (lines 6-7). We select the next example to clean π_{T+1} that minimizes the entropy (lines 8-9). We then ask a human to clean the selected example (line 10). The greedy algorithm terminates when all validation examples become CP'ed (line 3). Finally, we return any world D among all possible partially cleaned worlds $\mathcal{I}_{\mathcal{D}_\pi}$ (line 12). Since all the validation examples are CP'ed with $\mathcal{I}_{\mathcal{D}_\pi}$, classifier trained on any world in $\mathcal{I}_{\mathcal{D}_\pi}$, including the unknown ground truth world D^*, has the same validation accuracy. Therefore, $\mathcal{A}_D$ has the same validation accuracy as $\mathcal{A}_{D^*}$.

Example 7. *Figure 3.9 shows an example of how* CPClean *selects the next data example to clean in each iteration via sequential information maximization. Assume there are two dirty examples, $\mathcal{C}_1$ and $\mathcal{C}_2$, in the training set and each example has two candidate repairs. Therefore, there are four possible clean versions after cleaning the next data point, based on which data point is selected to be cleaned and which candidate repair is the ground truth. For example, the first table at step 1 shows the clean version after cleaning $\mathcal{C}_1$ if $x_{1,1}$ is the ground truth. Assume that we have two*

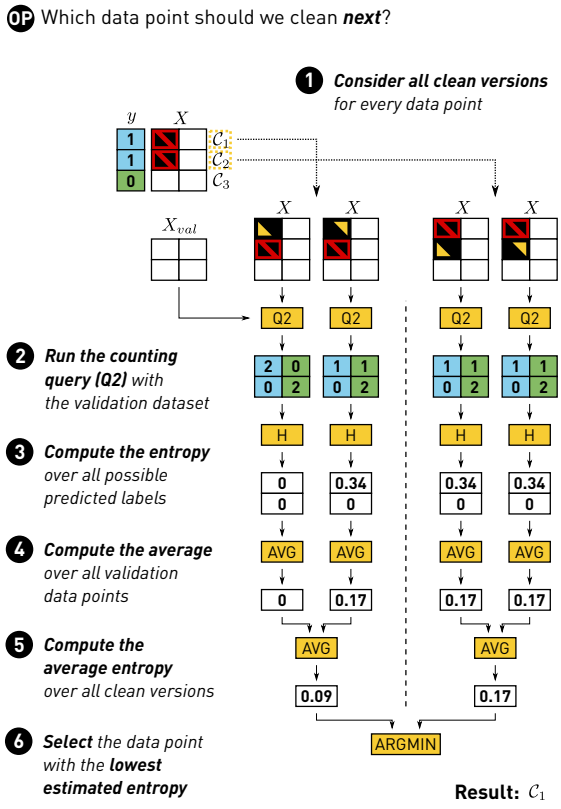

FIGURE 3.9: CPClean via sequential info. maximization.

validation examples. We run the counting query (Q2) on each possible version w.r.t. each validation example as shown in step 2. Then we can compute the entropy of predictions on validation examples as shown in step 3 and 4. The results show that if $\mathcal{C}_1$ *is selected to be cleaned, the entropy may become* 0 *or* 0.17 *depending on which candidate repair is the ground truth. We assume that each of the two candidate repairs has 50% chance to be the ground truth. Therefore, the expected entropy after cleaning* $\mathcal{C}_1$ *is* $(0 + 0.17)/2 = 0.09$ *(step 5). Similarly, we compute the expected entropy after cleaning* $\mathcal{C}_2$ *as* 0.17*. Since* $\mathcal{C}_1$ *has a lower expected entropy, we select* $\mathcal{C}_1$ *to clean.*

COMPLEXITY OF CPCLEAN. In each iteration of Algorithm 6, we need to (1) automatically select a tuple; and (2) ask human to clean the selected tuple. To select

a tuple, we need to first check whether $|D_{val}|$ are all CP'ed (line 3), which invokes the $Q1$ query $O(|D_{val}|)$ times. If not all D_{val} are CP'ed, we need to compute expected value of entropy $O(N)$ times (line 6). Computing the expected value of entropy (line 7) needs to invoke the $Q2$ query $O(M|D_{val}|)$ times. Therefore, when the downstream ML model is KNN, using our SS algorithm for $Q1$ and $Q2$, the complexity for selecting a tuple at each iteration is $O(N^2M^2|D_{val}| \times (\log(MN) + K \log N))$. The quadratic complexity in tuple selection is acceptable in practice, since human involvement is generally considered to be the most time consuming part in practical data cleaning [49].

THEORETICAL GUARANTEE. The theoretical analysis of this algorithm, while resembling that of [40], is non-trivial. We provide the main theoretical analysis here and leave the proof to the appendix.

Corollary 2. *Let the optimal cleaning policy that minimizes the cleaning effort while consistently classifying the test examples be denoted by $D_{Opt} \subseteq D_{train}$ with limited cardinality t, such that*

$$D_{Opt} = \underset{D_\pi \subseteq \mathcal{D}_{train},\ |D_\pi| \leq t}{\arg\max} I(\mathcal{A}_{\mathbf{D}}(D_{val}); D_\pi).$$

The sequential information maximization strategy follows a near optimal strategy where the information gathering satisfies

$$I(\mathcal{A}_{\mathbf{D}}(D_{val}); \mathbf{c}_{\pi_1}, ..., \mathbf{c}_{\pi_T}) \geq I(\mathcal{A}_{\mathbf{D}}(D_{val}); D_{Opt})(1 - \exp(-T/\theta t'))$$

where

$$\theta = \left(\max_{v \in \mathcal{D}_{train}} I(\mathcal{A}_{\mathbf{D}}(D_{val}); v) \right)^{-1}$$
$$t' = t \min\{\log |\mathcal{Y}|, \log M\}, \quad \mathcal{Y} : \textit{label space}, \ M : |\mathcal{C}_i|.$$

The above result, similarly as in [40], suggests that data cleaning is guaranteed to achieve near-optimal information gathering up to a logarithmic factor $\min(\log |\mathcal{Y}|, \log M)$ when leveraging the sequential information strategy.

Dataset	Error Type	#Examples	#Features	Missing rate
BabyProduct [62]	real	3042	7	11.8%
Supreme [63]	synthetic	3052	7	20%
Bank [64]	synthetic	3192	8	20%
Puma [64]	synthetic	8192	8	20%

TABLE 3.1: Datasets characteristics

3.10 Evaluation of CPClean

We now conduct an extensive set of experiments to evaluate the effectiveness of **CPClean** at the task of labor efficient data repair. We compare it with other data cleaning approaches in the context of K-nearest neighbor classifiers.

3.10.1 Experimental Setup

HARDWARE AND PLATFORM. All our experiments were performed on a machine with a 2.20GHz Intel Xeon(R) Gold 5120 CPU.

DATASETS. One main challenge of evaluating data cleaning solutions is the lack of datasets with ground truth, and hence most data cleaning work resort to synthetic error injection. This is especially true in the context of incomplete information: a dataset with missing values is not likely to come with ground truth. In this work, besides three datasets with synthetic errors, we manage to find one dataset with *real* missing values, where we are able to obtain the *ground truth* via manual Googling. We summarize all datasets in Table 4.1.

The BabyProduct dataset contains various baby products of different categories (e.g., bedding, strollers). Since the dataset was scraped from websites using Python scripts [62], many records have missing values, presumably due to extractor errors. We designed a classification task to predict whether a given baby product has a high price or low price based on other attributes (e.g. weight, brand, dimension, etc), and we selected a subset of product categories whose price difference is not so high so as to make the classification task more difficult. For records with missing brand attribute, we then perform a Google search using the product title to obtain

the product brand. For example, one record titled "*Just Born Safe Sleep Collection Crib Bedding in Grey*" is missing the product brand, and a search reveals that the brand is "Just Born."

We also use three datasets (Supreme, Bank, Puma), originally with no missing values, to inject synthetic missing values. Our goal is to inject missing values in the most realistic way possible and also to ensure that the missing values can have a large impact on classification accuracy. We follow the popular "Missing Not At Random" assumption [65], where the probability of missing may be higher for more sensitive/important attributes. For example, high income people are more likely to not report their income in a survey. We first assess the relative importance of each feature in a classification task (by measuring the accuracy loss after removing a feature), and use the relative feature importance as the relative probability of a feature missing. We can then inject missing values into a dataset for any given missing rate (we use 20% in our experiment).

MODEL. We use a KNN classifier with K=3 and use Euclidean distance as the similarity function. For each dataset, we randomly select 1,000 examples as the validation set and 1,000 examples as the test set. The remaining examples are used as the training set.

CLEANING ALGORITHMS COMPARED. We compare the following approaches for handling missing values in the training data.

- *Ground Truth*: This method uses the ground-truth version of the dirty data, and shows the performance upper-bound.
- *Default Cleaning*: This is the default and most commonly used way for cleaning missing values in practice, namely, missing cells in a numerical column are filled in using the mean value of the column, and those in a categorical column are filled using the most frequent value of that column.
- *CPClean*: This is our proposal, which needs a candidate repair set $\mathcal{C}_i$ for each example with missing values. For missing cells in numerical columns, we consider five candidate repairs: the minimum value, the 25-th percentile, the mean value, the 75-th percentile and the maximum value of the column. For missing cells in categorical columns, we also consider five candidate repairs: the top 4 most frequent categories and a dummy category named "other category". If a record i

has multiple missing values, then the Cartesian product of all candidate repairs for all missing cells forms $\mathcal{C}_i$. We simulate human cleaning by picking the candidate repair that is closest to the ground truth.

- *HoloClean*: This is the state-of-the-art probabilistic data cleaning method [4]. As a weakly supervised machine learning system, it leverages multiple signals (e.g. quality rules, value correlations, reference data) to build a probabilistic model for imputing and cleaning data. Note that the focus of HoloClean is to find the most likely fix for a missing cell in a dataset without considering how the dataset is used by downstream classification tasks.

- *BoostClean*: This is the state-of-the-art automatic data cleaning method for ML [38]. At a high level, it selects, from a predefined set of cleaning methods, the one that has the maximum validation accuracy on the validation set. To ensure fair comparison, we use the same cleaning method as in CPClean, i.e., the predefined cleaning methods include cleaning a numerical column with missing values using 25-th percentile, the mean value, etc. We also use the same validation set as in CPClean.

- *RandomClean*: While CPClean uses the idea of sequential information maximization to select which examples to clean, *RandomClean* simply selects an example randomly to clean.

PERFORMANCE MEASURES. Besides the cleaning effort spent, we are mainly concerned with the test accuracy of models trained on datasets cleaned by different cleaning methods. Instead of reporting exact test accuracies for all methods, we only report them for *Ground Truth* and *Default Cleaning*, which represents the upper bound and the lower bound, respectively. For other methods, we report the percentage of closed gap defined as:

$$\textit{gap closed by } X = \frac{\text{accuracy(X) - accuracy(}\textit{Default Cleaning}\text{)}}{\text{accuracy(}\textit{Ground Truth}\text{) - accuracy(}\textit{Default Cleaning}\text{)}}.$$

3.10.2 Experimental Results

MODEL ACCURACY COMPARISON. Table 3.2 shows the end-to-end performance of our method and other automatic cleaning methods. We can see that the missing values exhibit different degress of impact on these datasets (the gap between

Dataset	*Ground Truth*	*Default Cleaning*	*BoostClean*	*HoloClean*	*CPClean* (Early Termination)		*CPClean* (Partial Cleaning)	
	Test Accuracy	**Test Accuracy**	**Gap Closed**	**Gap Closed**	**Gap Closed**	**Examples Cleaned**	**Gap Closed**	**Examples Cleaned**
BabyProduct	0.668	0.589	1%	1%	99%	64%	72%	20%
Supreme	0.968	0.877	12%	-4%	100%	15%	100%	20%
Bank	0.643	0.558	20%	11%	102%	93%	52%	20%
Puma	0.794	0.747	28%	-64%	102%	63%	40%	20%

TABLE 3.2: End-to-End Performance Comparison

Ground Truth and *Default Cleaning*). We can also observe that *HoloClean*, the state-of-the-art standalone data cleaning approach performs poorly — the gap closed can even be negative. This suggests that performing data cleaning on a data without considering how it is used later may not necessarily improve downstream model performance. On the other hand, we observe that *BoostClean* shows a consistently positive impact on model performance by using the validation set to pick the most useful cleaning method. In all cases, *CPClean* is able to close 100% of gap without manual cleaning of all dirty data. In fact, on Supreme, *CPClean* only requires the manual cleaning of 15% of missing records to close 100% gap. We can also see from Table 3.2 that, by cleaning only 20% of all dirty data, i.e., terminating the cleaning process at 20% mark even if not all validation examples are CP'ed, *CPClean* is able to close 66% gap on average.

IMPACT OF EARLY TERMINATION. If users have a limited cleaning budget, they may choose to terminate *CPClean* early. To study the effectiveness of *CPClean* in prioritizing cleaning effort, we compare it with *RandomClean* that randomly picks an example to clean at each iteration. The results for *RandomClean* are the average of 20 runs.

The red lines in Figure 3.10 show the percentage of CP'ed examples in the validation set as more and more examples are cleaned. As we can see, *CPClean* (solid red line) dramatically outperforms the *RandomClean* (dashed red line) both in terms of the number of training examples cleaned so that all validation examples are CP'ed and in terms of the rate of convergence. For example, for Supreme, *CPClean* requires the cleaning of 15% examples while *RandomClean* requires cleaning almost all training examples.

The blue lines in Figure 3.10 show the percentage of gap closed for the test set accuracy. Again, we can observe that *CPClean* significantly outperforms *Random-*

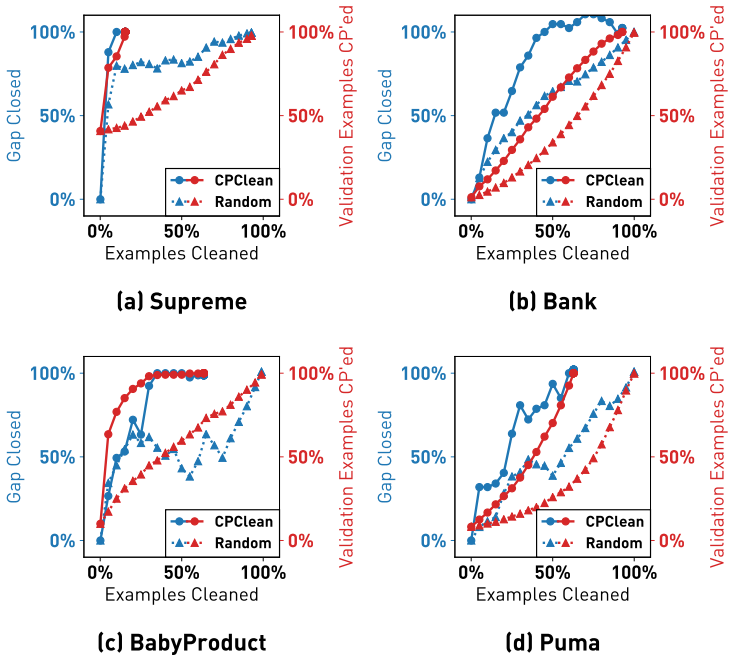

FIGURE 3.10: Comparison with Random Cleaning

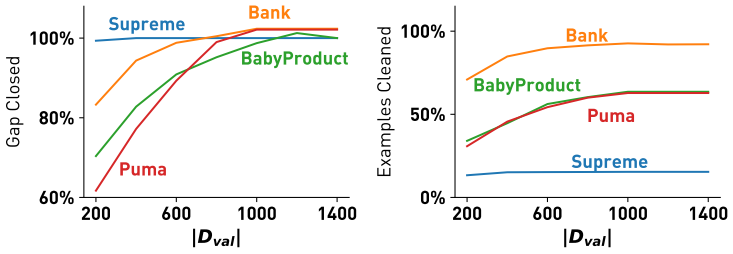

FIGURE 3.11: Varying size of D_{val}.

Clean. For example, with 50% of data cleaned in Bank, *RandomClean* only closes about 65% of the gap, whereas *CPClean* closes almost 100% of the gap.

IMPACT OF THE VALIDATION SET SIZE. We vary the size of D_{val} to understand how it affects the result. As shown in Figure 3.11, as the size of validation set increases, both the test accuracy gap closed and the cleaning effort spent first

increase and then become steady. This is because, when the validation set is small, it is easier to make all validation examples CP'ed (hence the smaller cleaning effort). However, a small validation set may not be representative of some unseen test set, and hence may not close the accuracy gap on test set. In all cases, we observe that 1K validation set is sufficiently large and further increasing it does not improve the performance.

3.11 Conclusion

In this chapter, we focused on the problem of understanding the impact of incomplete information on training downstream ML models. We presented a formal study of this impact by extending the notion of *Certain Answers for Codd tables*, which has been explored by the database research community for decades, into the field of machine learning, by introducing the notion of *Certain Predictions* (CP). We developed efficient algorithms to analyze the impact via CP primitives, in the context of nearest neighor classifiers.

As an application, we further proposed a novel "DC for ML" framework called **CPClean** that uses *information gain* as a measure of importance. We show how this measure can be computed by using the CP primitives as building blocks. We evaluate this measure of importance by measuring its impact on the efficiency of the data cleaning process. We find that it often significantly outperforms existing techniques in accuracy, with mild manual cleaning effort. In Section 1.1.3, we gave an example of how this method can be integrated in a systematic way into the ease.ml workflow.

4

REPAIR OF WRONG DATA GUIDED BY SHAPLEY VALUES

Previously, we have seen our method for performing repairs over missing data. We will not switch to the problem of repairing wrong data. That is, with data errors that are not easily detectable using some automatic data validator. We are again interested in optimizing the data debugging process by minimizing the effort involved while maximizing the model quality improvement. We introduce ease.ml/datascope, a system component designed for optimizing the process of repairing wrong data. In Figure 4.1 we can see its place in the overall ease.ml workflow.

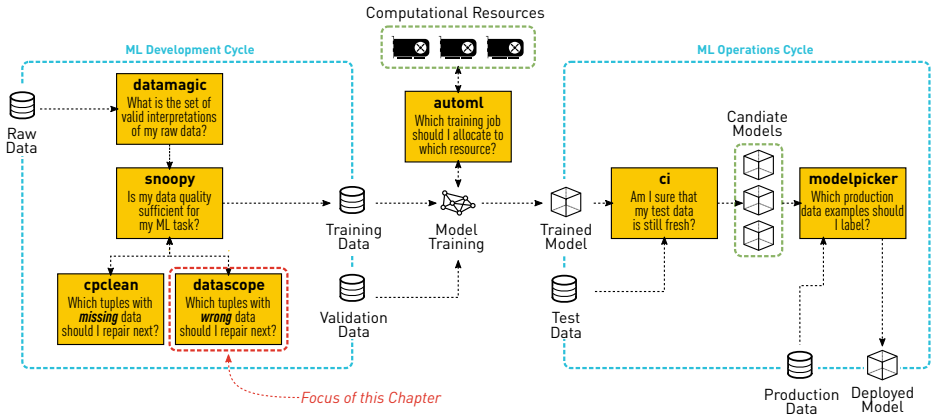

FIGURE 4.1: In this chapter we focus on repairing tuples with *wrong data*.

For this we will use the Shapley value, a well known concept from game theory which has also been successfully applied to the problem of explainability in machine learning as well as for data repairs. Unfortunately, this value is notoriously difficult to compute given that its definition involves a sum over an exponentially large space. Encouragingly, prior work has found it to be possible to compute this value under certain assumptions.

```
# Data loading
train_data = pd.read_csv("...")
test_data = pd.read_csv("...")
side_data = pd.read_csv("...")
# Data integration
train_data = train_data.join(side_data, on="item_id")
test_data = test_data.join(side_data, on="item_id")
# Declaratively defined (nested) feature encoding pipeline
pipeline = Pipeline([
    ('features', ColumnTransformer([
    (StandardScaler(), ["num_att1", "num_att2"]),
    (Pipeline([SimpleImputer(), OneHotEncoder()])),
        ["cat1", "cat2"]),
    (HashingVectorizer(n_features=100), "text_att1")])),
    # ML model for learning
    ('learner', SVC())])
# Train and evaluate model
pipeline.fit(train_data, train_data.label)
print(pipeline.score(test_data, test_data.label))

# ********************* DATASCOPE *********************
# Run data importance computation over pipeline
(train_data_with_imp, side_data_with_imp) = \
    DataScope.debug(pipeline, sklearn.metrics.accuracy)
# *****************************************************
```

LISTING 4.1: A simplified illustration of the core functionality enabled by DataScope — given an end-to-end ML pipeline (Line 1-19), and a utility (e.g., `sklearn.metrics.accuracy`), DataScope computes the Shapley value of each training example as its *importance* with respect to the given utility.

In this chapter, we are interested in efficient ways to compute the Shapley value in the context of end-to-end machine learning pipelines which are made up of feature extraction operators and machine learning models. We will developed a theoretical framework which includes tracking information about the provenance of data as it passes through a pipeline and compiling this information into decision diagrams. We will then show how this framework can be leveraged to computing the Shapley value. Finally, we empirically evaluate the usafulness of the Shapley value as an importance measure for guiding our process of repairing wrong data. In this chapter we will focus specifically on repairing wrong labels, which is a common data quality issue in machine learning settings.

4.1 Background and Related Work

Last decade has witnessed the rapid advancement of machine learning (ML), which brought along the advancement of *machine learning systems* [66]. Thanks to these advancements, training a machine learning model has never been easier today for practitioners — distributed learning over hundreds of devices [67]–[71], tuning hyper-parameters and selecting the best model [72]–[74], all of which become much more systematic and less mysterious. Moreover, all major cloud service providers now support AutoML and other model training and serving services.

DATA-CENTRIC CHALLENGES AND OPPORTUNITIES. Despite these great advancements, when it comes to building better machine learning applications, a new collection of challenges starts to emerge. One observation getting great attention recently is that *the quality of a model is often a reflection of the quality of the underlying training data.* As a result, often the most practical and efficient way of improving ML model quality is to improve data quality. As a result, recently, researchers have studied how to conduct data cleaning [15], [24], data debugging [34]–[36], [75]–[77], and data acquisition [5], specifically for the purpose of improving an ML model.

DATA DEBUGGING VIA DATA IMPORTANCE. In this chapter, we focus on the fundamental problem of reasoning about the *importance of training examples with respect to some utility function (e.g., validation accuracy, fairness, etc.) of the trained ML model.* There has been intensive recent interest in developing methods for reasoning about data importance. These efforts can be categorized into two different views. The *Leave-One-Out (LOO)* view of this problem tries to calculate, given a training set $\mathcal{D}$, the importance of a data example $x \in \mathcal{D}$ modeled as the *utility* decrease after removing this data example: $U(\mathcal{D}) - U(\mathcal{D} \backslash x)$. To scale-up this process over a large dataset, researchers have been developing approximation methods such as the *influence function* for a diverse set of ML models [75]. On the other hand, the *Expected-Improvement (ExpI)* view of this problem tries to calculate such a utility decrease over *all possible subsets of* $\mathcal{D}$. Intuitively, this line of work models data importance as an "expectation" over all possible subsets/sub-sequences of $\mathcal{D}$, instead of trying to reason about it solely on a single training set. One popular approach is to use Shapley value [34], [36], [77], a concept in game theory that has been applied to data importance and data valuation [35].

SHAPLEY-BASED DATA IMPORTANCE. In this chapter, we do not champion one view over the other (i.e., LOO vs. ExpI). We scope ourselves and only focus on Shapley-based methods since previous work has shown applications that can only use Shapley-based methods because of the favorable properties enforced by the Shapley value. Furthermore, taking expectations can sometimes provide a more reliable importance measure [35] than simply relying on a single dataset. Nevertheless, we believe it is important for future ML systems to support both and we hope this work can inspire future research in data importance for both the LOO and ExpI views.

One key challenge of Shapley-based data importance is its computational complexity — in the worst case, it needs to enumerate *exponentially* many subsets. There have been different ways to *approximate* this computation, either with MCMC [77] and group testing [34] or proxy models such as K-nearest neighbors (KNN) [36]. One surprising result is that Shapley-based data importance can be calculated efficiently (in *polynomial* time) for KNN classifiers [36], and using this as a proxy for other classifiers performs well over a diverse range of tasks [35].

DATA IMPORTANCE OVER PIPELINES. Existing methods for computing Shapley values [34]–[36], [77] are designed to directly operate on a single numerical input dataset for an ML model, typically in matrix form. However, in real-world ML applications, this data is typically generated on the fly from multiple data sources with an ML pipeline. Such pipelines often take multiple datasets as input, and transform them into a single numerical input dataset with relational operations (such as joins, filters, and projections) and common feature encoding techniques, often based on nested estimator/transformer pipelines, which are integrated into popular ML libraries such as scikit-learn [78], SparkML [79], or Google TFX [7]. It is an open problem how to apply Shapley-value computation in such a setup.

Listing 4.1 shows a toy example of such an end-to-end ML pipeline, which includes relational operations from pandas for data preparation (lines 3-9), a nested estimator/transformer pipeline for encoding numerical, categorical, and textual attributes as features (lines 12-16), and an ML model from scikit-learn (line 18). The code loads the data, splits it temporally into training and test datasets, 'fits' the pipeline to train the model, and evaluates the predictive quality on the test dataset. This leads us to the key question:

Can we efficiently compute Shapley-based data importance over such an end-to-end ML pipeline with both data processing and ML training?

PRIOR WORK ON ML EXPLAINABILITY. Understanding model predictions and handling problems when they arise has been an important area. In recent years, this topic has gained even more traction and is better known under the terms *explainability* and *interpretability* [80]–[82]. The goal is to understand why a model makes some specific prediction for a data example. Some approaches use surrogate models to produce explanations [83], [84]. In computer vision, saliency maps have gained prominence [85], [86]. Saliency maps can be seen as a type of a *feature importance* approach to explainability, although they also aim at interpreting the internals of a model. Other feature importance frameworks have also been developed [87], [88], some of which focus on Shapley-based feature importance [89].

Another approach to model interpretability can be referred to as *data importance* (i.e. using training data examples to explain predictions). This can have broader applications, including data valuation [90]. One important line of work expresses data importance in terms of *influence functions* [75], [76], [91], [92]. Another line of work expresses data importance using Shapley values. Some apply Monte Carlo methods for efficiently computing it [77] and some take advantage of the KNN model [34], [35]. The KNN model has also been used for computing an entropy-based importance method targeted specifically for the data-cleaning application [24].

PRIOR WORK ON ANALYZING DATA PROCESSING PIPELINES. The data management community has been studying how to analyze data importance for data-processing pipelines through various forms of query analysis. Some broader approaches include: causal analysis of interventions to queries [93], and reverse data management [94]. Methods also differ in terms of the target of their explanation, i.e., the query output being explained. Some methods target queried relations [95]. Others target specific predicates that make up a query [96], [97]. Finally, some methods target specific tuples in input relations [98].

A prominent line of work employs *data provenance* as a means to analyze data processing pipelines [99]. *Provenance semiring* represents a theoretical framework for dealing with provenance [100]. This framework gives us as a theoretical foundation to develop algorithms for analyzing ML pipelines. However, this framework is not guaranteed to lead us to tractable solutions. Some types of queries have been shown

to be quite difficult [101], [102]. In this work, we demonstrate tractability of a concrete class of pipelines comprised of both a set of feature extraction operators, as well as an ML model.

Recent research under the umbrella of "mlinspect" [103]–[105] details how to compute data provenance over end-to-end ML pipelines similar to the ones in the focus of this work, based on lightweight code instrumentation.

PRIOR WORK ON END-TO-END PIPELINE ANALYSIS. Joint analysis of machine learning pipelines is a relatively novel, but nevertheless, an important field [106]. Systems such as Data X-Ray can debug data processing pipelines by finding groups of data errors that might have the same cause [107]. Some work has been done in the area of end-to-end pipeline compilation to tensor operations [108]. A notable piece of work leverages influence functions as a method for analyzing pipelines comprising of a model and a post-processing query [9]. This work also leverages data provenance as a key ingredient. In general, there are indications that data provenance is going to be a key ingredient of future ML systems [105], [109], something that our own system depends upon.

4.2 Preliminaries

In this section we describe several concepts from existing research that we use as basis for our contributions. Specifically, (1) we present the definition of machine learning pipelines and their semantics, and (2) we describe decision diagrams as a tool for compact representation of Boolean functions.

4.2.1 End-to-end ML Pipelines

An end-to-end ML application consists of two components: (1) a feature extraction pipeline, and (2) a downstream ML model. To conduct a joint analysis over one such end-to-end application, we leave the precise definition to Section 4.3.2. One important component in our analysis relies on the *provenance* of the feature extraction pipeline, which we will discuss below.

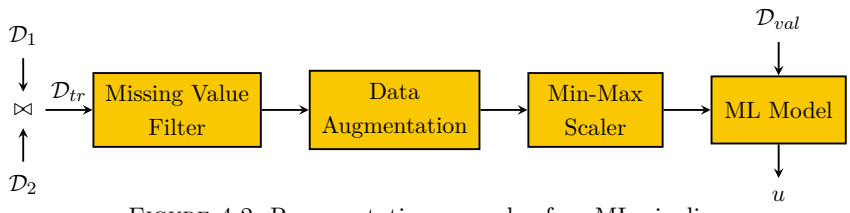

FIGURE 4.2: Representative example of an ML pipeline.

We take a traditional data processing view in our treatment of machine learning (feature extraction) pipelines. A pipeline f is composed of one or more data processing operators. Each operator takes one or more datasets $\mathcal{D}$ and produces an output dataset. Datasets are made up of tuples taken from some data domain $\mathbb{D}$. We assume a data model with named attributes, where each attribute a is simply a function that maps tuples t to values $a(t)$.

In the machine learning setting, data is pulled from one or more sources and transformed with a feature processing pipeline f. The output of this pipeline is a training dataset $\mathcal{D}_{tr}$ that is used for training a model. We focus on the supervised learning setting where each training tuple $t \in \mathcal{D}_{tr}$ has an attribute y that corresponds to its label $y(t) \in \mathcal{Y}$ taken from a set of class labels $\mathcal{Y}$. We also have an unlabeled validation dataset $\mathcal{D}_{val}$ for which the trained model predicts the label. Thus, a machine learning model is also a data processing operator that takes a labeled training dataset and an unlabeled validation dataset in order to produces predicted labels.

PROVENANCE TRACKING. We are given a source training dataset $\mathcal{D}_{tr}$, which is a set of data examples represented as tuples t with named attributes. Input examples (tuples) in $\mathcal{D}_{tr}$ are transformed by a feature processing pipeline before being turned into a processed training dataset $\mathcal{D}_{tr}^{f} := f(\mathcal{D}_{tr})$, which is directly used to train the model. To enable ourselves to compute the importance of examples in $\mathcal{D}_{tr}$, it is useful to relate the presence of tuples in $\mathcal{D}_{tr}^{f}$ in the training dataset with respect to the presence of tuples in $\mathcal{D}_{tr}$. In other words, we need to know the *provenance* of training tuples. In this work, we rely on the well-established theory of provenance semirings [100] to describe such provenance.

We define a set of variables A where a variable $a_t \in A$ is associated with every tuple t in the training dataset $\mathcal{D}_{tr}$. We define *value assignment* $v : A \to \mathbb{B}$ to describe whether a given tuple t appears in $\mathcal{D}_{tr}$ — by setting $v(a_t) = 0$, we "exclude" t from

$\mathcal{D}_{tr}$ and by setting $v(a_t) = 1$, we "include" t in $\mathcal{D}_{tr}$. Let $\mathcal{V}_A$ be the set of all possible such value assignments ($|\mathcal{V}_A| = 2^{|A|}$). We use $\mathcal{D}_{tr}[v] = \{t \in \mathcal{D}_{tr} | v(a_t) \neq 0\}$ to denote a subset of training examples, only containing tuples t whose corresponding variable a_t is set to 1 according to v.

To describe the association between $\mathcal{D}_{tr}$ and its transformed version $\mathcal{D}^f_{tr}$, we annotate each potential tuple in $\mathcal{D}^f_{tr}$ with an attribute $p : \mathbb{D} \to \mathbb{B}[A]$ containing its *provenance polynomial* [100], which is a logical formula with variables in A and binary coefficients (e.g. $a_1 + a_2 \cdot a_3$) — $p(t)$ is true only if tuple t appears in $\mathcal{D}^f_{tr}$. For such polynomials, an *addition* corresponds to a *union* operator in the ML pipeline, and a *multiplication* corresponds to a *join* operator in the pipeline. Figure 4.4 illustrates some examples of the association between a_t and $p(t)$.

Given a value assignment $v \in \mathcal{V}_A$, we can define an evaluation function $\text{eval}_v\ \phi$ that returns the *evaluation* of a provenance polynomial ϕ under the assignment v. Given a value assignment v, we can obtain the corresponding *transformed dataset* by evaluating all provenance polynomials of its tuples, as such:

$$\mathcal{D}^f_{tr}[v] := \{t \in \mathcal{D}^f_{tr} \mid \text{eval}_v(p(t)) \neq 0\}. \tag{4.1}$$

Intuitively, $\mathcal{D}^f_{tr}[v]$ corresponds to the result of applying the feature transformation f over a subset of training examples that only contains tuples t whose corresponding variable a_t is set to 1.

Using this approach, given a feature processing pipeline f and a value assignment v, we can obtain the transformed training set $\mathcal{D}^f_{tr}[v] = f(\mathcal{D}_{tr}[v])$.

4.2.2 Additive Decision Diagrams (ADD's)

KNOWLEDGE COMPILATION. Our approach of computing the Shapley value will rely upon being able to construct functions over Boolean inputs $\phi : \mathcal{V}_A \to \mathcal{E}$, where $\mathcal{E}$ is some finite *value set*. We require an elementary algebra with $+$, $-$, $\cdot$ and $/$ operations to be defined for this value set. Furthermore, we require this value set to contain a *zero element* 0, as well as an *invalid element* ∞ representing an undefined result (e.g. a result that is out of bounds). We then need to count the number of value assignments $v \in \mathcal{V}_A$ such that $\phi(v) = e$, for some specific value $e \in \mathcal{E}$. This is referred to as the *model counting* problem, which is #P complete for arbitrary logical formulas [110], [111]. For example, if $A = \{a_1, a_2, a_3\}$, we can define

$\mathcal{E} = \{0, 1, 2, 3, \infty\}$ to be a value set and a function $\phi(v) := v(a_1) + v(a_2) + v(a_3)$ corresponding to the number of variables in A that are set to 1 under some value assignment $v \in \mathcal{V}_A$.

Knowledge compilation [112] has been developed as a well-known approach to tackle this model counting problem. It was also successfully applied to various problems in data management [113]. One key result from this line of work is that, if we can construct certain polynomial-size data structures to represent our logical formula, then we can perform model counting in polynomial time. Among the most notable of such data structures are *decision diagrams*, specifically binary decision diagrams [114], [115] and their various derivatives [116]–[118]. For our purpose in this chapter, we use the *additive decision diagrams* (ADD), as detailed below.

ADDITIVE DECISION DIAGRAMS (ADD). We define a simplified version of the *affine algebraic decision diagrams* [117]. An ADD is a directed acyclic graph defined over a set of nodes $\mathcal{N}$ and a special *sink node* denoted as $\boxdot$. Each node $n \in \mathcal{N}$ is associated with a variable $a(n) \in A$. Each node has two outgoing edges, $c_L(n)$ and $c_H(n)$, that point to its *low* and *high* child nodes, respectively. For some value assignment v, the low/high edge corresponds to $v(a) = 0/v(a) = 1$. Furthermore, each low/high edge is associated with an increment w_L/w_H that maps edges to elements of $\mathcal{E}$.

The semantics of an entire ADD is that each path from the root node to sink node $\boxdot$ corresponds to some value assignment $v \in \mathcal{V}_A$. This allows us to represent a logical formula ϕ where the value $\phi(v)$ is computed by *adding* together all increments along the path determined by v. Figure 4.3a shows an example ADD with one path highlighted in red.

Note that each node $n \in \mathcal{N}$ represents the root of a subgraph and defines a Boolean function. Given some value assignment $v \in \mathcal{V}_A$ we can evaluate this function by constructing a path starting from n and at each step moving towards the low or high child depending on whether the corresponding variable is assigned 0 or 1. The value of the function is the result of adding all the edge increments together.

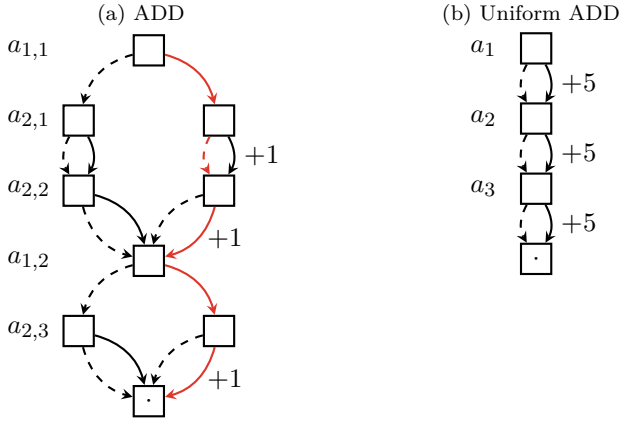

FIGURE 4.3: (a) An ordered and full ADD for computing $\phi(v) := v(a_{1,1}) \cdot \big(v(a_{2,1}) + v(a_{2,2})\big) + v(a_{1,2}) \cdot v(a_{2,3})$. (b) A uniform ADD for computing $\phi(v) := 5 \cdot (v(a_1) + v(a_2) + v(a_3))$.

Figure 4.3a presents an example ADD with one path highlighted in red. Formally, we can define the evaluation of the function defined by the node n as follows:

$$\text{eval}_v(n) := \begin{cases} 0, & \text{if } n = \boxdot, \\ w_L(n) + \text{eval}_v(c_L(n)) & \text{if } v(x(n)) = 0, \\ w_H(n) + \text{eval}_v(c_H(n)) & \text{if } v(x(n)) = 1. \end{cases} \tag{4.2}$$

In our work we focus specifically on ADD's that are *full* and *ordered*. A diagram is full if every path from root to sink encounters every variable in A exactly once. For example, in Figure 4.3a we see a full diagram over the set of variables $A = \{a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}, a_{2,3}\}$. If any of the variables in A has no node associated with it, then the diagram is not considered full. On the other hand, an ADD is ordered when on each path from root to sink variables always appear in the same order. For this purpose, we define $\pi : A \to \{1, ..., |A|\}$ to be a permutation of variables that assigns each variable $a \in A$ an index. For example, in Figure 4.3a, the variable order is $\pi = \{a_{1,1} \mapsto 1, a_{1,2} \mapsto 4, a_{2,1} \mapsto 2, a_{2,2} \mapsto 3, a_{2,3} \mapsto 5\}$. It is possible, for example, to swap the two nodes on the left side that correspond to $a_{2,1}$ and $a_{2,2}$. This, however, makes the diagram unordered, which dramatically complicates certain operations (e.g. the diagram summation operation that we will describe shortly).

MODEL COUNTING. We define a model counting operator

$$\text{count}_e(n) := \left|\left\{v \in \mathcal{V}_{A[\leq \pi(a(n))]} \mid \text{eval}_v(n) = e\right\}\right|, \tag{4.3}$$

where $A[\leq \pi(a(n))]$ is the subset of variables in A that include $a(n)$ and all variables that come before it in the permutation π. For an ordered and full ADD, $\text{count}_e(n)$ satisfies the following recursion:

$$\text{count}_e(n) := \begin{cases} 1, & \text{if } e = 0 \text{ and } n = \boxdot, \\ 0, & \text{if } e = \infty \text{ or } n = \boxdot, \\ \text{count}_{e-w_L(n)}(c_L(n)) + \text{count}_{e-w_H(n)}(c_H(n)), & \text{otherwise.} \end{cases} \tag{4.4}$$

The above recursion can be implemented as a dynamic program with computational complexity $O(|\mathcal{N}| \cdot |\mathcal{E}|)$.

Figure 4.3b shows a special case of a full and ordered ADD, which we call a *uniform* ADD. It is structured as a chain with one node per variable, where all low increments equal zero and all high increments equal some constant $E \in \mathcal{E}$. For this type of ADD we can perform model counting in constant time, assuming that we have a precomputed table of factorials of size $O(|\mathcal{N}|)$ that allows us to compute binomial coefficients in constant time. The count_e operator for a uniform ADD can be defined as

$$\text{count}_e(n) := \begin{cases} \binom{\pi(a(n))}{e/E}, & \text{if } e \bmod E = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

Intuitively, if we observe the uniform ADD shown in Figure 4.3b, we see that the result of an evaluation must be a multiple of 5. For example, to evaluate to 10, the evaluation path must pass a *high* edge exactly twice. Therefore, in a 3-node ADD with root node n_R, the result of $\text{count}_{10}(n_R)$ will be exactly $\binom{3}{2}$.

SPECIAL OPERATIONS ON ADD'S. Given an ADD with node set $\mathcal{N}$, we define two operations that will become useful later on when constructing diagrams for our specific scenario:

1. *Variable restriction*, denoted as $\mathcal{N}[a_i \leftarrow V]$, which restricts the domain of variables A by forcing the variable a_i to be assigned the value V. This operation removes every node $n \in \mathcal{N}$ where $a(n) = a_i$ and rewires all incoming edges to point to the node's high or low child, depending on whether $V = 1$ or $V = 0$.

2. *Diagram summation*, denoted as $\mathcal{N}_1 + \mathcal{N}_2$, where $\mathcal{N}_1$ and $\mathcal{N}_2$ are two ADD's over the same (ordered) set of variables A. It starts from the respective root nodes n_1 and n_2, and produces a new node $n := n_1 + n_2$. We then apply the same operation to child nodes. Therefore, $c_L(n_1 + n_2) := c_L(n_1) + c_L(n_2)$ and $c_H(n_1 + n_2) := c_H(n_1) + c_H(n_2)$. Also, for the increments, we can define $w_L(n_1 + n_2) := w_L(n_1) + w_L(n_2)$ and $w_H(n_1 + n_2) := w_H(n_1) + w_H(n_2)$.

4.3 Data Importance over ML Pipelines

We first recap the problem of computing data importance for ML pipelines in Section 4.3.1, formalise the problem in Section 4.3.2, and outline core technical efficiency and scalability issues afterwards. We will describe the DataScope approach in Section 4.4 and our theoretical framework in Section 4.5.

4.3.1 Data Importance for ML Pipelines

In real-world ML, one often encounters data-related problems in the input training set (e.g., wrong labels, outliers, biased samples) that lead to sub-optimal quality of the user's model. As illustrated in previous work [34]–[36], [75]–[77], many data debugging and understanding problems hinge on the following fundamental question:

> *Which data examples in the training set are most important for the model utility ?*

A common approach is to model this problem as computing the *Shapley value* of each data example as a measure of its importance to a model, which has been applied to a wide range of use cases [34]–[36], [77]. However, this line of work focused solely on ML model training but ignored the *data pre-processing pipeline* prior to model training, which includes steps such as feature extraction, data augmentation, etc. This significantly limits its applications to real-world scenarios, most of which consist of a non-trivial data processing pipeline [119]. In this work, we take the first step in applying Shapley values to debug end-to-end ML pipelines.

4.3.2 Formal Problem Definition

We first formally define the core technical problem.

ML PIPELINES. Let $\mathcal{D}_e$ be an input training set for a machine learning task, potentially accompanied by additional relational side datasets $\mathcal{D}_{s_1}, \ldots, \mathcal{D}_{s_k}$. We assume the data to be in a *star* database schema, where each tuple from a side dataset $\mathcal{D}_{s_i}$ (the "dimension" tables) can be joined with multiple tuples from $\mathcal{D}_e$ (the "fact" table). Let f be a feature extraction pipeline that transforms the relational inputs $\mathcal{D}_{tr} = \{\mathcal{D}_e, \mathcal{D}_{s_1}, \ldots, \mathcal{D}_{s_k}\}$ into a set of training tuples $\{t_i = (x_i, y_i)\}_{i\in[m]}$ made up of feature and label pairs that the ML training algorithm $\mathcal{A}$ takes as input. Note that $\mathcal{D}_e$ represents `train_data` in our toy example in Listing 4.1, $\mathcal{D}_s$ represents `side_data`, while f refers to the data preparation operations from lines 6-14, and the model $\mathcal{A}$ corresponds to the support vector machine `SVC` from line 16.

After feature extraction and training, we obtain an ML model $\mathcal{A} \circ f(\mathcal{D}_{tr})$. We can measure the *quality* of this model in various ways, e.g., via validation accuracy or a fairness metric. Let $\mathcal{D}_v$ be a given set of relational validation data with the same schema as $\mathcal{D}_e$. Applying f to $\mathcal{D}_{val} = \{\mathcal{D}_v, \mathcal{D}_{s_1}, \ldots, \mathcal{D}_{s_k}\}$ produces a set of validation tuples $\{t_i = (\tilde{x}_i, \tilde{y}_i)\}_{i\in[p]}$ made up of feature and label pairs, on which we can derive predictions with our trained model $\mathcal{A} \circ f(\mathcal{D}_{tr})$. Based on this, we define a utility function u, which measures the performance of the predictions:

$$u(\mathcal{A} \circ f(\mathcal{D}_{tr}), f(\mathcal{D}_{val})) \mapsto [0, 1]. \tag{4.6}$$

For readability, we use the following notation in cases where the model $\mathcal{A}$ and pipeline f are clear from context:

$$u(\mathcal{D}_{tr}, \mathcal{D}_{val}) := u(\mathcal{A} \circ f(\mathcal{D}_{tr}), f(\mathcal{D}_{val})). \tag{4.7}$$

ADDITIVE UTILITIES. We will focus on *additive utilities* that cover the most important set of utility functions in practice (e.g., validation loss, validation accuracy, various fairness metrics, etc.). A utility function u is *additive* if there exists a *tuple-wise* utility u_T such that u can be rewritten as

$$u(\mathcal{D}_{tr}, \mathcal{D}_{val}) = w \cdot \textstyle\sum_{t_{val}\in f(\mathcal{D}_{val})} u_T\Big(\Big(\mathcal{A} \circ f(\mathcal{D}_{tr})\Big)(t_{val}), t_{val}\Big). \tag{4.8}$$

Here, w is a scaling factor only relying on $\mathcal{D}_{val}$. The tuple-wise utility $u_T : (y_{pred}, t_{val}) \mapsto [0,1]$ takes a validation tuple $t_{val} \in \mathcal{D}_{val}$ as well as a class label $y_{pred} \in \mathcal{Y}$ predicted by the model for t_{val}. It is easy to see that some popular utilities, such as validation accuracy, are additive, e.g., the accuracy utility is simply defined by plugging $u_T(y_{pred}, (x_{val}, y_{val})) := \mathbb{1}\{y_{pred} = y_{val}\}$ into Equation 4.8.

EXAMPLE: FALSE NEGATIVE RATE AS AN ADDITIVE UTILITY. Apart from accuracy that represents a trivial example of additive utility, we can show many more complex utilities are additive and can therefore be decomposed according to Equation 4.8. As an example, we use *false negative rate (FNR)*, that is, the portion of examples with a positive true label that the model wrongly labeled as negative. It can be formally defined as

$$u(\mathcal{D}_{tr}, \mathcal{D}_{val}) := \frac{\sum_{t_{val} \in f(\mathcal{D}_{val})} \mathbb{1}\{(\mathcal{A} \circ f(\mathcal{D}_{tr}))(t_{val}) = 0\} \mathbb{1}\{y(t_{val}) = 1\}}{|\{t_{val} \in \mathcal{D}_{val} \,:\, y(t_{val}) = 1\}|}. \quad (4.9)$$

In the above expression we can see that the denominator only depends on $\mathcal{D}_{val}$, which means it can be interpreted as the scaling factor w. We can easily see that the expression in the numerator neatly fits the structure of Equation 4.8 as long as we we define u_T as $u_T(y_{pred}, (x_{val}, y_{val})) := \mathbb{1}\{y_{pred} = 0\} \mathbb{1}\{y_{val} = 1\}$. Similarly, we are able to represent various other utilities, including false positive rate, true positive rate (i.e. recall), true negative rate (i.e. specificity), etc. We describe an additional example in Section 4.4.3.

SHAPLEY VALUE. The Shapley value, denoting the importance of an input tuple t_i for the ML pipeline, is defined as

$$\varphi_i = \frac{1}{|\mathcal{D}_{tr}|} \sum_{S \subseteq \mathcal{D}_{tr} \setminus \{t_i\}} \binom{n-1}{|S|}^{-1} \left(u(S \cup \{t_i\}, \mathcal{D}_{val}) - u(S, \mathcal{D}_{val})\right).$$

Intuitively, the *importance* of t_i over a subset $S \subseteq \mathcal{D}_{tr} \setminus \{t_i\}$ is measured as the difference between the utility $u \circ \mathcal{A} \circ f(S \cup \{t_i\})$ *with* t_i and the utility $u \circ \mathcal{A} \circ f(S)$ *without* t_i. The Shapley value takes the average of all such possible subsets $S \subseteq \mathcal{D}_{tr} \setminus \{t_i\}$, which allows it to have a range of desired properties that significantly benefit data debugging tasks, often leading to more effective data debugging mechanisms compared to other leave-one-out methods.

4.3.3 Prior Work and Challenges

All previous research focuses on the scenario in which there is no ML pipeline f (i.e., one directly works with the vectorised training examples $\{t_i\}$). Even in this case, computing Shapley value is tremendously difficult since its complexity for general ML model is #P-hard. To accommodate this computational challenge, previous work falls into two categories:

1. *Monte Carlo Shapley*: One natural line of work tries to estimate Shapley value with Markov Chain Monte Carlo (MCMC) approaches. This includes vanilla Monte Carlo sampling, group testing [34], [120], and truncated Monte Carlo sampling [77].
2. *KNN Shapley*: Even the most efficient Monte Carlo Shapley methods need to train multiple ML models (i.e., evaluate $\mathcal{A}$ multiple times) and thus exhibit long running time for datasets of modest sizes. Another line of research proposes to approximate the model $\mathcal{A}$ using a simpler proxy model. Specifically, previous work shows that Shapley values can be computed over K-nearest neighbors (KNN) classifiers in PTIME [35] and using KNN classifiers as a proxy is very effective in various real-world scenarios [36].

In this work, we face an even harder problem given the presence of an ML pipeline f in addition to the model $\mathcal{A}$. Nevertheless, as a baseline, it is important to realize that all Monte Carlo Shapley approaches [34], [77] can be directly extended to support our scenario. This is because most, if not all, Monte Carlo Shapley approaches operate on *black-box functions* and thus, can be used directly to handle an end-to-end pipeline $\mathcal{A} \circ f$.

CORE TECHNICAL PROBLEM. Despite the existence of such a Monte Carlo baseline, there remain challenges on scalability and speed — in our experiments in Section 4.6, it is not uncommon for such a Monte Carlo baseline to take a full hour to compute Shapley values even on a small dataset with only 1,000 examples. To bring data debugging and understanding into practice, we are in dire need for a more efficient and scalable alternative. Without an ML pipeline, using a KNN proxy model has been shown to be orders of magnitude faster than its Monte Carlo counterpart [35] while being equally, if not more, effective on many applications [36].

As a consequence, we focus on the following question: *Can we similarly use a KNN classifier as a proxy when dealing with end-to-end ML pipelines*? Today's KNN Shapley algorithm heavily relies on the structure of the KNN classifier. The presence of an ML pipeline will drastically change the underlying algorithm and time complexity — in fact, for many ML pipelines, computation of Shapley value is #P-hard even for KNN classifiers.

4.4 The DataScope Approach

We summarize our main theoretical contribution in Section 4.4.1, followed by the characteristics of ML pipelines to which these results are applicable (Section 4.4.2). We further discuss how we can approximate many real-world pipelines as *canonical pipelines* to make them compatible with our algorithmic approach (Section 4.4.3). We defer the details of our theoretical results to Section 4.5.

4.4.1 Overview

The key technical contribution of this chapter is a novel algorithmic framework that covers a large sub-family of ML pipelines whose KNN Shapley can be computed in PTIME. We call these pipelines *canonical pipelines*.

Theorem 3. *Let $\mathcal{D}_{tr}$ be a set of n training tuples, f be an ML pipeline over $\mathcal{D}_{tr}$, and $\mathcal{A}_{knn}$ be a K-nearest neighbor classifier. If f can be expressed as an Additive Decision Diagram (ADD) with polynomial size, then computing*

$$\varphi_i = \frac{1}{n} \sum_{S \subseteq \mathcal{D}_{tr} \setminus \{t_i\}} \binom{n-1}{|S|}^{-1} \left(u \circ \mathcal{A}_{knn} \circ f(S \cup \{t_i\}) - u \circ \mathcal{A}_{knn} \circ f(S)\right)$$

is in PTIME for all additive utilities u.

We leave the details about this Theorem to Section 4.5. This theorem provides a sufficient condition under which we can compute Shapley values for KNN classifiers over ML pipelines. We can instantiate this general framework with concrete types of ML pipelines.

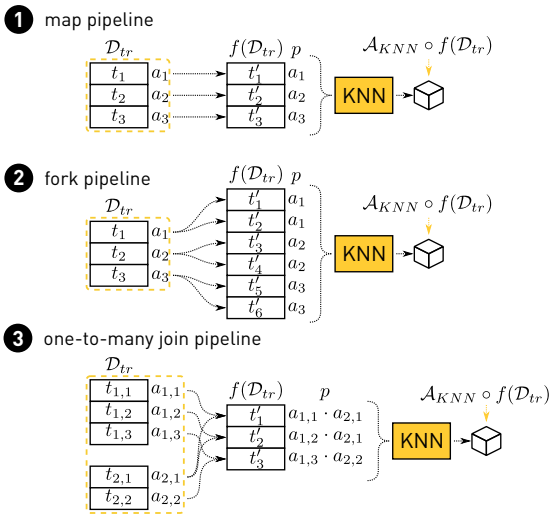

FIGURE 4.4: Three types of canonical pipelines over which Shapley values can be computed in PTIME.

4.4.2 Canonical ML Pipelines

As a prerequisite for an efficient Shapley-value computation over pipelines, we need to understand how the removal of an input tuple t_i from $\mathcal{D}_{tr}$ impacts the featurised training data $f(\mathcal{D}_{tr})$ produced by the pipeline. In particular, we need to be able to reason about the difference between $f(\mathcal{D}_{tr})$ and $f(\mathcal{D}_{tr} \setminus \{t_i\})$, which requires us to understand the *data provenance* [100], [121] of the pipeline f. In the following, we summarise three common types of pipelines (illustrated in Figure 4.4), to which we refer as *canonical pipelines*. We will show that Shapley values over these pipelines can be computed in PTIME in Section 4.5.

MAP PIPELINES. These are a family of pipelines that satisfy the condition in Theorem 3, in which the feature extraction f has the following property: each input training tuple t_i is transformed into a unique output training example t'_i with a tuple-at-a-time transformation function h_f: $t_i \mapsto t'_i = h_f(t_i)$. Map pipelines are the standard case for supervised learning, where each tuple of the input data is encoded as a feature vector for the model's training data. The provenance polynomial for

the output t'_i is $p(t'_i) = a_i$ in this case, where a_i denotes the presence of t_i in the input to the pipeline f.

FORK PIPELINES. They are a superset of Map pipelines, which requires that for each output example t_j, there exists a *unique* input tuple t_i, such that t_j is generated by applying a tuple-at-a-time transformation function h_f over t_i: $t_j = h_f(t_i)$. As illustrated in Figure 4.4(b), the output examples t_1 and t_2 are both generated from the input example t_1. Fork pipelines also satisfy the condition in Theorem 3. Fork pipelines typically originate from data augmentation operations for supervised learning, where multiple variants of a single tuple of the input data are generated (e.g., various rotations of an image in computer vision), and each copy is encoded as a feature vector for the model's training data. The provenance polynomial for an output t_j is again $p(t_j) = a_i$ in this case, where a_i denotes the presence of t_i in the input to the pipeline f.

ONE-TO-MANY JOIN PIPELINES. These are a superset of Fork pipelines, which rely on the star-schema structure of the relational inputs. Given the relational inputs $\mathcal{D}_e$ ("fact table") and $\mathcal{D}_s$ ("dimension table"), we require that, for each output example t_k, there exist *unique* input tuples $t_i \in \mathcal{D}_e$ and $t_j \in \mathcal{D}_s$ such that t_k is generated by applying a tuple-at-a-time transformation function h_f over the join pair (t_i, t_j): $t_k = h_f(t_i, t_j)$. One-to-Many Join pipelines also satisfy the condition in Theorem 3. Such pipelines occur when we have multiple input datasets in supervised learning, with the "fact" relation holding data for the entities to classify (e.g., emails in a spam detection scenario), and the "dimension" relations holding additional side data for these entities, which might result in additional helpful features. The provenance polynomial for an output t_k is $p(t_k) = a_i \cdot a_j$ in this case, where a_i and a_j denote the presence of t_i and t_j in the input to the pipeline f. Note that the polynomials states that both t_i and t_j must be present in the input at the same time (otherwise no join pair can be formed from them).

DISCUSSION. We note that this classification of pipelines assumes that the relational operations applied by the pipeline are restricted to the positive relational algebra (SPJU: Select, Project, Join, Union), where the pipeline applies no aggregations, and joins the input data according to the star schema. In our experience, this covers a lot of real-world use cases in modern ML infrastructures, where the ML pipeline consumes pre-aggregated input data from so-called "feature stores," which

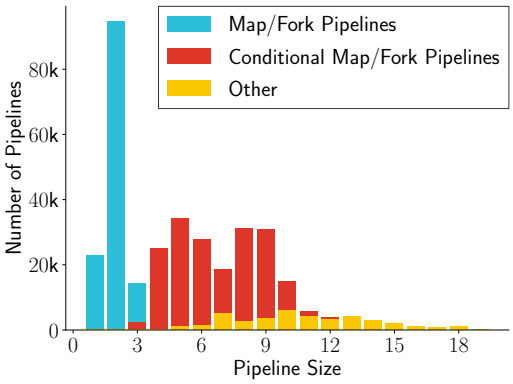

FIGURE 4.5: A majority of real-world ML pipelines [119] either already exhibit a canonical map-fork pipeline pattern, or are easily convertible to it using our approximation scheme.

is naturally modeled in a star schema. Furthermore, pipelines in the real-world operate on relational datasets using dataframe semantics [122], where unions and projections do not deduplicate their results, which (together with the absence of aggregations) has the effect that there are no additions present in provenance polynomials of the outputs of our discussed pipeline types. This pipeline model has also been proven helpful for interactive data distribution debugging [103], [104].

4.4.3 Approximating Real-World ML Pipelines

In practice, an ML pipeline f and its corresponding ML model $\mathcal{A}$ will often not directly give us a canonical pipeline whose Shapley value can be computed in PTIME. The reasons for this are twofold: (i) there might be no technique known to compute the Shapley value in PTIME for the given model; (ii) the estimator/transformer operations for feature encoding in the pipeline require global aggregations (e.g., to compute the mean of an attribute for normalising it). In such cases, each output depends on the whole input, and the pipeline does not fit into one of the canonical pipeline types that we discussed earlier.

As a consequence, we *approximate* an ML pipeline into a canonical pipeline in two ways.

APPROXIMATING $\mathcal{A}$. The first approximation follows various previous efforts summarised as KNN Shapley before, and has been shown to work well, if not better, in a diverse range of scenarios. In this step, we approximate the pipeline's ML model with a KNN classifier $\mathcal{A} \mapsto \mathcal{A}_{knn}$.

APPROXIMATING THE ESTIMATOR/TRANSFORMER STEPS IN f. In terms of the pipeline operations, we have to deal with the global aggregations applied by the estimators for feature encoding. Common feature encoding and dimensionality reduction techniques often base on a `reduce-map` pattern over the data:

$$op(\mathcal{D}) = \texttt{map}(\texttt{reduce}(\mathcal{D}), \mathcal{D}).$$

During the `reduce` step, the estimator computes some global statistics over the dataset — e.g., the estimator for `MinMaxScaling` computes the minimum and maximum of an attribute, and the `TFIDF` estimator computes the inverse document frequencies of terms. The estimator then generates a transformer, which applies the `map` step to the data, transforming the input dataset based on the computed global statistics, e.g., to normalise each data example based on the computed minimum and maximum values in the case of `MinMaxScaling` .

The global aggregation conducted by the `reduce` step is often the key reason that we cannot compute Shapley value in PTIME over a given pipeline — such a global aggregation requires us to enumerate all possible subsets of data examples, each of which corresponds to a potentially different global statistic. Fortunately, we also observe, and will validate empirically later, that the results of these global aggregations are relatively stable given different subsets of the data, especially in cases where what we want to compute is the *difference* when a single example is added or removed. The approximation that we conduct is to reuse the result of the `reduce` step computed over the whole dataset $\mathcal{D}_{tr}$ for a subset $\mathcal{D} \subset \mathcal{D}_{tr}$:

$$op(\mathcal{D}) = \texttt{map}(\texttt{reduce}(\mathcal{D}), \mathcal{D}) \mapsto op^*(\mathcal{D}) = \texttt{map}(\texttt{reduce}(\mathcal{D}_{tr}), \mathcal{D}). \tag{4.10}$$

In the case of scikit-learn, this means that we reuse the transformer generated by fitting the estimator on the whole dataset. Once all estimators op in an input pipeline are transformed into their approximate variant op^*, a large majority of realistic pipelines become canonical pipelines of a `map` or `fork` pattern.

STATISTICS OF REAL-WORLD PIPELINES. A natural question is how common these families of pipelines are in practice. Figure 4.5 illustrates a case study that we conducted over 500K real-world pipelines provided by Microsoft [119]. We divide pipelines into three categories: (1) "pure" map/fork pipelines, based on our definition of canonical pipelines; (2) "conditional" map/fork pipelines, which are comprised of a reduce operator that can be effectively approximated using the scheme we just described; and (3) other pipelines, which contain complex operators that cannot be approximated. We observe that a vast majority of pipelines we encountered in our case study fall into the first two categories that we can effectively approximate using our canonical pipelines framework.

DISCUSSION: WHAT IF THESE TWO APPROXIMATIONS FAIL? Computing Shapley values for a generic pipeline $(\mathcal{A}, f)$ is #P-hard, and by approximating it into $(\mathcal{A}_{knn}, f^*)$, we obtain an efficient PTIME solution. This drastic improvement on complexity also means that *we should expect that there exist scenarios under which this* $(\mathcal{A}, f) \mapsto (\mathcal{A}_{knn}, f^*)$ *approximation is not a good approximation.*

How often would this failure case happen in practice? When the training set is large, as illustrated in many previous studies focusing on the KNN proxy, we are confident that the $\mathcal{A} \mapsto \mathcal{A}_{knn}$ approximation should work well in many practical scenarios except those relying on some very strong global properties that KNN does not model (e.g., global population balance). As for the $f \mapsto f^*$ approximation, we expect the failure cases to be rare, especially when the training set is large. In our experiments, we have empirically verified these two beliefs, which were also backed up by previous empirical results on KNN Shapley [36].

What should we do when such a failure case happens? In such situations, we will resort to the Monte Carlo baseline, which will be orders of magnitude slower but should provide a backup alternative. It is an interesting direction to further explore the limitations of both approximations and develop more efficient Monte Carlo methods.

APPROXIMATING ADDITIVE UTILITIES: EQUALIZED ODDS DIFFERENCE. We show how slightly more complex utilities can also be represented as additive, with a little approximation, similar to the one described above. We demonstrate

this using the "equalized odds difference" utility, a commonly used measure of (un)fairness [123], [124] that we also use in our experiments. It is defined as

$$u(\mathcal{D}_{tr}, \mathcal{D}_{val}) := \max\{TPR_\Delta(\mathcal{D}_{tr}, \mathcal{D}_{val}), FPR_\Delta(\mathcal{D}_{tr}, \mathcal{D}_{val})\}. \tag{4.11}$$

Here, TPR_Δ and FPR_Δ are *true positive rate difference* and *false positive rate difference*, respectively. We assume that each tuple $t_{tr} \in f(\mathcal{D}_{tr})$ and $t_{val} \in f(\mathcal{D}_{val})$ have some sensitive feature g (e.g. ethnicity) with values taken from some finite set $\{G_1, G_2, ...\}$, that allows us to partition the dataset into *sensitive groups*. We can define TPR_Δ and FPR_Δ as

$$\begin{aligned} TPR_\Delta(\mathcal{D}_{tr}, \mathcal{D}_{val}) &:= \max_{G_i \in G} TPR_{G_i}(\mathcal{D}_{tr}, \mathcal{D}_{val}) - \min_{G_j \in G} TPR_{G_j}(\mathcal{D}_{tr}, \mathcal{D}_{val}), \text{ and} \\ FPR_\Delta(\mathcal{D}_{tr}, \mathcal{D}_{val}) &:= \max_{G_i \in G} FPR_{G_i}(\mathcal{D}_{tr}, \mathcal{D}_{val}) - \min_{G_j \in G} FPR_{G_j}(\mathcal{D}_{tr}, \mathcal{D}_{val}). \end{aligned} \tag{4.12}$$

For some sensitive group G_i, we define TPR_{G_i} and FPR_{G_i} as

$$\begin{aligned} &TPR_{G_i}(\mathcal{D}_{tr}, \mathcal{D}_{val}) := \\ &\frac{\sum_{t_{val} \in f(\mathcal{D}_{val})} \mathbb{1}\{(\mathcal{A} \circ f(\mathcal{D}_{tr}))(t_{val}) = 1\}\mathbb{1}\{y(t_{val}) = 1\}\mathbb{1}\{g(t_{val}) = G_i\}}{|\{t_{val} \in \mathcal{D}_{val} \ : \ y(t_{val}) = 1 \wedge g(t_{val}) = G_i\}|}, \text{ and} \\ &FPR_{G_i}(\mathcal{D}_{tr}, \mathcal{D}_{val}) := \\ &\frac{\sum_{t_{val} \in f(\mathcal{D}_{val})} \mathbb{1}\{(\mathcal{A} \circ f(\mathcal{D}_{tr}))(t_{val}) = 1\}\mathbb{1}\{y(t_{val}) = 0\}\mathbb{1}\{g(t_{val}) = G_i\}}{|\{t_{val} \in \mathcal{D}_{val} \ : \ y(t_{val}) = 0 \wedge g(t_{val}) = G_i\}|}. \end{aligned}$$

For a given training dataset $\mathcal{D}_{tr}$, we use Equation 4.11 to decide whether TPR_Δ or FPR_Δ is going to be the dominant metric. Similarly, given that choice, we can determine a pair of sensitive groups (G_{max}, G_{min}) that ends up being selected as minimal and maximal in Equation 4.12. Similarly to the conversion shown in Equation 4.10, we can treat these two steps as a `reduce` operation over the whole dataset. Then, if we assume that this intermediate result will remain stable over subsets of $\mathcal{D}_{tr}$, we can approximatly represent the equalized odds difference utility as an additive utility.

As an example, let us assume that we have determined that TPR_Δ dominates over FPR_Δ, and similarly that the pair of sensitive groups (G_{max}, G_{min}) will end up being selected in Equation 4.12. Then, our tuple-wise utility u_T and the scaling factor w become

$$\begin{aligned} u_T(y_{pred}, t_{val}) &:= TPR_{G_{max},T}(y_{pred}, t_{val}) - TPR_{G_{min},T}(y_{pred}, t_{val}), \\ w &:= 1/|\{t_{val} \in \mathcal{D}_{val} \ : \ y(t_{val}) = 1 \wedge g(t_{val}) = G_i\}|, \end{aligned}$$

where

$$TPR_{G_i,T}(y_{pred}, t_{val}) := \mathbb{1}\{y_{pred} = 1\}\mathbb{1}\{y(t_{val}) = 1\}\mathbb{1}\{g(t_{val}) = G_i\}.$$

A similar approach can be taken to define u_T and w for the case when FPR_Δ dominates over TPR_Δ.

4.5 Algorithm Framework: KNN Shapley Over Data Provenance

We now provide details for our theoretical results that are mentioned in Section 4.4. We present an algorithmic framework that efficiently computes the Shapley value over the KNN accuracy utility (defined in Equation 4.7 when $\mathcal{A}$ is the KNN model). Our framework is based on the following key ideas: (1) the computation can be reduced to computing a set of *counting oracles*; (2) we can develop PTIME algorithms to compute such counting oracles for the canonical ML pipelines, by translating their *provenance polynomials* into an Additive Decision Diagram (ADD).

4.5.1 Counting Oracles

We now unpack Theorem 3. Using the notations of data provenance introduced in Section 4.2, we can rewrite the definition of the Shapley value as follows, computing the value of tuple t_i, with the corresponding varible $a_i \in A$:

$$\varphi_i = \frac{1}{|A|} \sum_{v \in \mathcal{V}_{A\setminus\{a_i\}}} \frac{u(\mathcal{D}_{tr}^f[v[a_i \leftarrow 1]]) - u(\mathcal{D}_{tr}^f[v[a_i \leftarrow 0]])}{\binom{|A|-1}{|\text{supp}(v)|}}. \tag{4.13}$$

Here, $v[a_i \leftarrow X]$ represents the same value assignment as v, except that we enforce $v(a_i) = X$ for some constant X. Moreover, the support $\text{supp}(v)$ of a value assignment v is the subset of variables in A that are assigned value 1 according to v.

NEAREST NEIGHBOR UTILITY. When the downstream classifier is a K-nearest neighbor classifier, we have additional structure of the utility function $u(-)$ that we can take advantage of. Given a data example t_{val} from the validation dataset, the hyperparameter K controlling the size of the neighborhood and the set of class labels

$\mathcal{Y}$, we formally define the KNN utility $u_{t_{val},K,\mathcal{Y}}$ as follows. Given the transformed training set $\mathcal{D}_{tr}^{f}$, let σ be a scoring function that computes, for each tuple $t \in \mathcal{D}_{tr}^{f}$, its similarity with the validation example t_{val}: $\sigma(t, t_{val})$. In the following, we often write $\sigma(t)$ whenever t_{val} is clear from the context. We also omit σ when the scoring function is clear from the context. Given this scoring function σ, the KNN utility can be defined as follows:

$$u_{t_{val},K,\mathcal{Y}}(\mathcal{D}) := u_T\left(\text{argmax}_{y\in\mathcal{Y}}\left(\text{tally}_{y,\text{top}_K\mathcal{D}_{tr}^{f}}(\mathcal{D}_{tr}^{f})\right), t_{val}\right) \tag{4.14}$$

where $\text{top}_K\mathcal{D}_{tr}^{f}$ returns the tuple t which ranks at the K-th spot when all tuples in $\mathcal{D}_{tr}^{f}$ are ordered by decreasing similarity σ. Given this tuple t and a class label $y \in \mathcal{Y}$, the $\text{tally}_{y,t}$ operator returns the number of tuples with similarity score greater or equal to t that have label y. We assume a standard majority voting scheme where the predicted label is selected to be the one with the greatest tally ($\arg\max_y$). The accuracy is then computed by simply comparing the predicted label with the label of the validation tuple t_{val}.

Plugging the KNN accuracy utility into Equation 4.13, we can augment the expression for computing φ_i as

$$\begin{aligned}\varphi_i = \frac{1}{|A|}\sum_{v\in\mathcal{V}_{A\setminus\{a_i\}}}\sum_{\alpha=1}^{|A|}&\mathbb{1}\{\alpha = |\text{supp}(v)|\}\binom{|A|-1}{\alpha}^{-1}\\ &\cdot\sum_{t,t'\in\mathcal{D}_{tr}^{f}}\mathbb{1}\{t = \text{top}_K\mathcal{D}_{tr}^{f}[v[a_i\leftarrow 0]]\}\\ &\quad\cdot\mathbb{1}\{t' = \text{top}_K\mathcal{D}_{tr}^{f}[v[a_i\leftarrow 1]]\}\\ &\cdot\sum_{\gamma,\gamma'\in\Gamma}\mathbb{1}\{\gamma = \text{tally}_t\mathcal{D}_{tr}^{f}[v[a_i\leftarrow 0]]\}\\ &\quad\cdot\mathbb{1}\{\gamma' = \text{tally}_{t'}\mathcal{D}_{tr}^{f}[v[a_i\leftarrow 1]]\}\\ &\quad\cdot u_\Delta(\gamma,\gamma')\end{aligned} \tag{4.15}$$

where $\text{tally}_t\mathcal{D} = (\text{tally}_{c_1,t}\mathcal{D}...\text{tally}_{y_{|\mathcal{Y}|},t}\mathcal{D})$ returns a tally vector $\gamma \in \Gamma \subset \mathbb{N}^{|\mathcal{Y}|}$ consisting of the tallied occurrences of each class label $y \in \mathcal{Y}$ among tuples with similarity to t_{val} greater than or equal to that of the boundary tuple t. Let Γ be all possible tally vectors (corresponding to all possible label "distributions" over top-K).

Here, the innermost utility gain function is formally defined as $u_\Delta(\gamma,\gamma') := u_\Gamma(\gamma') - u_\Gamma(\gamma)$, where u_Γ is defined as

$$u_\Gamma(\gamma) := u_T(\text{argmax}_{y\in\mathcal{Y}}\gamma, t_{val}).$$

Intuitively, $u_\Delta(\gamma, \gamma')$ measures the utility difference between two different label distributions (i.e., tallies) of top-K examples: γ and γ'. $u_T(y, t_{val})$ is the tuple-wise utility for a KNN prediction (i.e., $\text{argmax}_{y \in \mathcal{Y}} \gamma$) and validation tuple t_{val}, which is the building block of the *additive utility*. The correctness of Equation 4.15 comes from the observation that for any distinct $v \in \mathcal{V}_{A \setminus \{a_i\}}$, there is a unique solution to all indicator functions $\mathbb{1}$. Namely, there is a single t that is the K-th most similar tuple when $v(a_i) = 0$, and similarly, a single t' when $v(a_i) = 1$. Given those *boundary tuples* t and t', the same goes for the *tally vectors*: given $\mathcal{D}^f_{tr}[v[a_i \leftarrow 0]]$ and $\mathcal{D}^f_{tr}[v[a_i \leftarrow 1]]$, there exists a unique γ and γ'.

We can now define the following *counting oracle* that computes the sum over value assignments, along with all the predicates:

$$\begin{aligned} \omega_{t,t'}(\alpha, \gamma, \gamma') := \sum_{v \in \mathcal{V}_{A \setminus \{a_i\}}} & \cdot \mathbb{1}\{\alpha = |\text{supp}(v)|\} \\ & \cdot \mathbb{1}\{t = \text{top}_K \mathcal{D}^f_{tr}[v[a_i \leftarrow 0]]\} \\ & \cdot \mathbb{1}\{t' = \text{top}_K \mathcal{D}^f_{tr}[v[a_i \leftarrow 1]]\} \\ & \cdot \mathbb{1}\{\gamma = \text{tally}_t \mathcal{D}^f_{tr}[v[a_i \leftarrow 0]]\} \\ & \cdot \mathbb{1}\{\gamma' = \text{tally}_t \mathcal{D}^f_{tr}[v[a_i \leftarrow 1]]\}. \end{aligned} \tag{4.16}$$

Using counting oracles, we can simplify Equation 4.15 as:

$$\varphi_i = \frac{1}{N} \sum_{t,t' \in \mathcal{D}^f_{tr}} \sum_{\alpha=1}^{N} \binom{N-1}{\alpha}^{-1} \sum_{\gamma, \gamma' \in \Gamma} u_\Delta(\gamma, \gamma') \omega_{t,t'}(\alpha, \gamma, \gamma'). \tag{4.17}$$

We see that the computation of φ_i will be in PTIME if we can compute the counting oracles $\omega_{t,t'}$ in PTIME (ref. Theorem 3). As we will demonstrate next, this is indeed the case for the canonical pipelines that we focus on in this work.

4.5.2 Counting Oracles for Canonical Pipelines

We start by discussing how to compute the counting oracles using ADD's in general. We then study the canonical ML pipelines in particular and develop PTIME algorithms for them.

4.5.2.1 *Counting Oracle using ADD's*

We use Additive Decision Diagram (ADD) to compute the counting oracle $\omega_{t,t'}$ (Equation 4.16). An ADD represents a Boolean function $\phi : \mathcal{V}_A \to \mathcal{E} \cup \{\infty\}$ that maps value assignments $v \in \mathcal{V}_A$ to elements of some set $\mathcal{E}$ or a special invalid element ∞ (see Section 4.2.2 for more details). For our purpose, we define $\mathcal{E} := \{1, ..., |A|\} \times \Gamma \times \Gamma$, where Γ is the set of label tally vectors. We then define a function over Boolean inputs $\phi_{t,t'} : \mathcal{V}_A[a_i = 0] \to \mathbb{N}$ as follows:

$$\begin{aligned}
\phi_{t,t'}(v) &:= \begin{cases} \infty, & \text{if } t \notin \mathcal{D}\big[v[a_i \leftarrow 0]\big], \\ \infty, & \text{if } t' \notin \mathcal{D}\big[v[a_i \leftarrow 1]\big], \\ (\alpha, \gamma, \gamma'), & \text{otherwise,} \end{cases} \\
\alpha &:= |\text{supp}(v)|, \\
\gamma &:= \text{tally}_t \mathcal{D}\big[v[a_i \leftarrow 0]\big], \\
\gamma' &:= \text{tally}_{t'} \mathcal{D}\big[v[a_i \leftarrow 1]\big].
\end{aligned} \tag{4.18}$$

If we can construct an ADD with a root node $n_{t,t'}$ that computes $\phi_{t,t'}(v)$, then the following equality holds:

$$\omega_{t,t'}(\alpha, \gamma, \gamma') = \text{count}_{(\alpha,\gamma,\gamma')}(n_{t,t'}). \tag{4.19}$$

Given that the complexity of model counting is $O(|\mathcal{N}| \cdot |\mathcal{E}|)$ (see Equation 4.4) and the size of $\mathcal{E}$ is polynomial in the size of data, we have

Theorem 4. *If we can represent the $\phi_{t,t'}(v)$ in Equation 4.18 with an ADD of size polynomial in $|A|$ and $|\mathcal{D}^f_{tr}|$, we can compute the counting oracle $\omega_{t,t'}$ in time polynomial of $|A|$ and $|\mathcal{D}^f_{tr}|$.*

4.5.2.2 *Compiling Provenance Polynomials into Decision Diagrams*

In this section we examine the method of compiling a provenance-tracked dataset $f(\mathcal{D})$ that results from a pipeline f. The crux of the method is defined in Algorithm 7 which is an algorithm that takes a dataset $\mathcal{D}$ with provenance tracked over a set of variables $\mathcal{X}$ and a boundary tuple $t \in \mathcal{D}$.

Assuming that all provenance polynomials are actually a single conjunction of variables, and that the tally is always a sum over those polynomials, it tries to

Algorithm 7 Compiling a provenance-tracked dataset into an ADD.

1: **function** CompileADD
2: **inputs**
3: $\mathcal{D}$, provenance-tracked dataset;
4: $\mathcal{X}$, set of variables;
5: t, boundary tuple;
6: **outputs**
7: $\mathcal{N}$, nodes of the compiled ADD;
8: **begin**
9: $\mathcal{N} \leftarrow \{\}$
10: $\mathcal{P} \leftarrow \{(x_1, x_2) \in \mathcal{X} \ : \exists t \in \mathcal{D}, x_1 \in p(t) \ \wedge \ x_2 \in p(t)\}$
11: $\mathcal{X}_L \leftarrow$ GetLeafVariables($\mathcal{P}$)
12: **for** $\mathcal{X}_C \in$ GetConnectedComponents($\mathcal{P}$) **do**
13: $\mathcal{N}' \leftarrow$ ConstructADDTree($\mathcal{X}_C \setminus \mathcal{X}_L$)
14: $\mathcal{X}' \leftarrow \mathcal{X}_C \setminus \mathcal{X}_L$
15: $\mathcal{D}' \leftarrow \{t' \in \mathcal{D} \ : \ p(t') \cup \mathcal{X}_C \neq \emptyset\}$
16: **for** $v \in \mathcal{V}_{\mathcal{X}'}$ **do**
17: $\mathcal{N}_C \leftarrow$ ConstructADDChain($\mathcal{X}_C \cap \mathcal{X}_L$)
18: **for** $n \in \mathcal{N}_C$ **do**
19: $v' \leftarrow v \cup \{x(n) \rightarrow 1\}$
20: $a_H(n) \leftarrow |\{t' \in \mathcal{D}' \ : \ \text{eval}_{v'} p(t') = 1 \ \wedge \ \sigma(t') \geq \sigma(t)\}|$
21: **end for**
22: $\mathcal{N}' \leftarrow$ AppendToADDPath($\mathcal{N}'$, $\mathcal{N}_C$, v)
23: **end for**
24: $\mathcal{N} \leftarrow$ AppendToADDRoot($\mathcal{N}$, $\mathcal{N}'$)
25: **end for**
26: **for** $x' \in p(t)$ **do**
27: **for** $n \in \mathcal{N}$ **where** $x(n) = x'$ **do**
28: $a_L(n) \leftarrow \infty$
29: **end for**
30: **end for**
31: **return** $\mathcal{N}$
32: **end function**

perform factoring by determining if there are any variables that can be isolated. This is achieved by first extracting variables that appear only once (Line 11) separating the total sum into components that don't share any variables (Line 12). Then for the variables that cannot be isolated (because they appear in polynomials in multiple tuples with multiple different variables) we form a group which will be treated as one binary vector and based on the value of that vector we would take a specific path in the tree. We thus take the group of variables and call the ConstructADDTree function to construct an ADD tree (Line 13).

Every path in this tree corresponds to one value assignment to the variables in that tree. Then, for every path we call the ConstructADDChain to build a chain made up of the isolated variables and call AppendToADDPath to append them to the leaf of that path (Line 22). For each variable in the chain we also define an increment that is defined by the number of tuples that will be more similar than the boundary

tuple t and also have their provenance polynomial "supported" by the path. We thus construct a segment of the final ADD made up of different components. We append this segment to the final ADD using the AppendToADDRoot function. We don't explicitly define these functions but we illustrate their functionality in Figure 4.6.

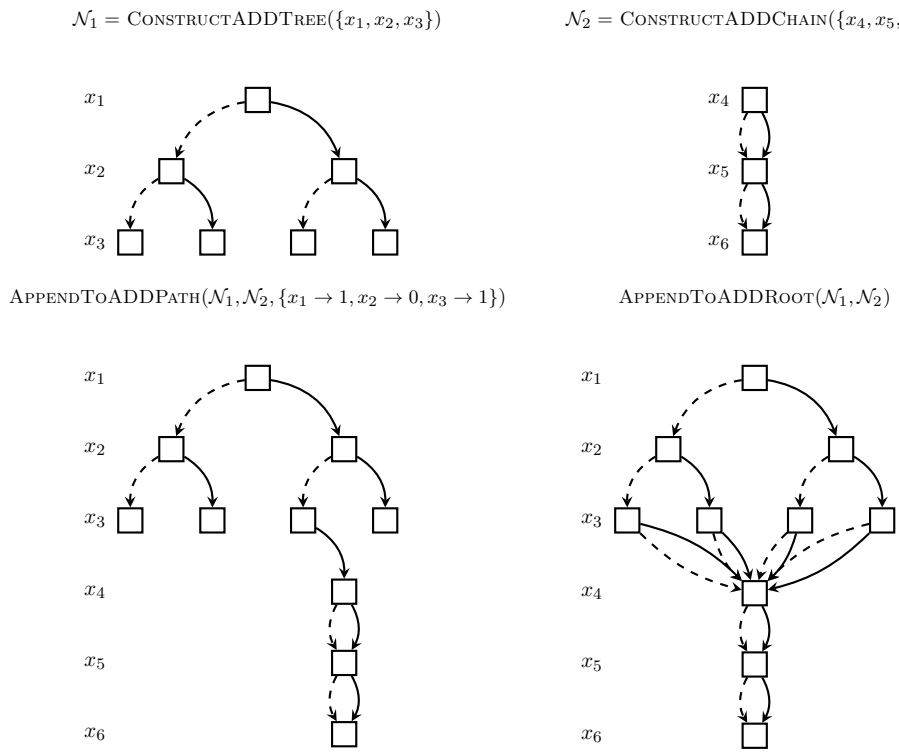

FIGURE 4.6: An example of a ADD compilation functions.

4.5.3 Computing the Shapley Value for ML Pipelines

We can now examine different types of canonical pipelines and see how their structures are reflected onto the ADD's. In summary, we can construct an ADD with polynomial size for canonical pipelines and therefore, by Theorem 4, the computation of the corresponding counting oracles is in PTIME.

ONE-TO-MANY JOIN PIPELINE. In a *star* database schema, this corresponds to a *join* between a *fact* table and a *dimension* table, where each tuple from the dimension table can be joined with multiple tuples from the fact table. It can be represented by an ADD similar to the one in Figure 4.3a.

Corollary 5. *For the K-NN accuracy utility and a one-to-many* join *pipeline, which takes as input two datasets, $\mathcal{D}_F$ and $\mathcal{D}_D$, of total size $|\mathcal{D}_F| + |\mathcal{D}_D| = N$ and outputs a joined dataset of size $O(N)$, the Shapley value can be computed in $O(N^4)$ time.*

We present the proof in Section A.4.2 in the appendix.

FORK PIPELINE. The key characteristic of a pipeline f that contains only *fork* or *map* operators is that the resulting dataset $f(\mathcal{D})$ has provenance polynomials with only a single variable. This is due to the absence of joins, which are the only operator that results in provenance polynomials with a combination of variables.

Corollary 6. *For the K-NN accuracy utility and a* fork *pipeline, which takes as input a dataset of size N and outputs a dataset of size M, the Shapley value can be computed in $O(M^2N^2)$ time.*

We present the proof in Section A.4.3 in the appendix.

MAP PIPELINE. A *map* pipeline is similar to *fork* pipeline in the sense that every provenance polynomial contains only a single variable. However, each variable now can appear in a provenance polynomial of *at most* one tuple, in contrast to *fork* pipeline where a single variable can be associated with *multiple* tuples. This additional restriction results in the following corollary:

Corollary 7. *For the K-NN accuracy utility and a* map *pipeline, which takes as input a dataset of size N, the Shapley value can be computed in $O(N^2)$ time.*

We present the proof in Section A.4.4 in the appendix.

4.5.4 Special Case: 1-Nearest-Neighbor Classifiers

We can significantly reduce the time complexity for 1-NN classifiers, an important special case of K-NN classifiers that is commonly used in practice. For each validation tuple t_{val}, there is always *exactly* one tuple that is most similar to t_{val}. Below we illustrate how to leverage this observation to construct the counting oracle. In the following, we assume that a_i is the variable corresponding to the tuple for which we hope to compute Shapley value.

Let ϕ_t represent the event when t is the top-1 tuple:

$$\phi_t := p(t) \wedge \bigwedge_{\substack{t' \in f(\mathcal{D}_{tr}) \\ \sigma(t') > \sigma(t)}} \neg p(t'). \tag{4.20}$$

For Equation 4.20 to be *true* (i.e. for tuple t to be the top-1), all tuples t' where $\sigma(t') > \sigma(t)$ need to be *absent* from the pipeline output. Hence, for a given value assignment v, all provenance polynomials that control those tuples, i.e., $p(t')$, need to evaluate to false.

We now construct the event

$$\phi_{t,t'} := \phi_t[a_i/\mathsf{false}] \wedge \phi_{t'}[a_i/\mathsf{true}],$$

where $\phi_t[a_i/\mathsf{false}]$ means to substitue all appearances of a_i in ϕ_t to false. This event happens only if if t is the top-1 tuple when a_i is false and t' is the top-1 tuple when a_i is true. This corresponds to the condition that our counting oracle counts models for. Expanding $\phi_{t,t'}$, we obtain

$$\phi_{t,t'} := \Big(p(t) \wedge \bigwedge_{\substack{t'' \in f(\mathcal{D}_{tr}) \\ \sigma(t'') > \sigma(t)}} \neg p(t'')\Big)[a_i/\mathsf{false}] \wedge \Big(p(t') \wedge \bigwedge_{\substack{t'' \in f(\mathcal{D}_{tr}) \\ \sigma(t'') > \sigma(t')}} \neg p(t'')\Big)[a_i/\mathsf{true}]. \tag{4.21}$$

Note that $\phi_{t,t'}$ can only be *true* if $p(t')$ is true when a_i is true and $\sigma(t) < \sigma(t')$. As a result, all provenance polynomials corresponding to tuples with a higher similarity score than that of t need to evaluate to false. Therefore, the only polynomials that can be allowed to evaluate to true are those corresponding to tuples with lower similarity score than t. Based on these observations, we can express the counting oracle for different types of ML pipelines.

MAP PIPELINE. In a *map* pipeline, the provenance polynomial for each tuple $t \in f(\mathcal{D}_{tr})$ is defined by a single distinct variable $a_t \in A$. Furthermore, from the definition of the counting oracle (Equation 4.16), we can see that each $\omega_{t,t'}$ counts the value assignments that result in support size α and label tally vectors γ and γ'. Given our observation about the provenance polynomials that are allowed to be set to true, we can easily construct an expression for counting valid value assignments. Namely, we have to choose exactly α variables out of the set $\{t'' \in \mathcal{D} \ : \ \sigma(t'') < \sigma(t)\}$, which corresponds to tuples with lower similarity than that of t. This can be constructed using a *binomial coefficient*. Furthermore, when $K = 1$, the label tally γ is entirely determined by the top-1 tuple t. The same observation goes for γ' and t'. To denote this, we define a constant Γ_L parameterized by some label L. It represents a tally vector with all values 0 and only the value corresponding to label L being set to 1. We thus need to fix γ to be equal to $\Gamma_{y(t)}$ (and the same for γ'). Finally, as we observed earlier, when computing $\omega_{t,t'}$ for $K = 1$, the provenance polynomial of the tuple t' must equal a_i. With these notions, we can define the counting oracle as

$$\omega_{t,t'}(\alpha, \gamma, \gamma') = \binom{|\{t'' \in \mathcal{D} \ : \ \sigma(t'') < \sigma(t)\}|}{\alpha} \mathbb{1}\{p(t') = a_i\} \mathbb{1}\{\gamma = \Gamma_{y(t)}\} \mathbb{1}\{\gamma' = \Gamma_{y(t')}\}. \tag{4.22}$$

Note that we always assume $\binom{a}{b} = 0$ for all $a < b$. Given this, we can prove the following corollary about *map* pipelines:

Corollary 8. *For the* 1*-NN accuracy utility and a* map *pipeline, which takes as input a dataset of size* N*, the Shapley value can be computed in* $O(N \log N)$ *time.*

We present the proof in Section A.4.5 in the appendix.

FORK PIPELINE. As we noted, both *map* and *fork* pipelines result in polynomials made up of only one variable. The difference is that in *map* pipeline each variable is associated with at most one polynomial, whereas in *fork* pipelines it can be associated with multiple polynomials. However, for 1-NN classifiers, this difference vanishes when it comes to Shapley value computation:

Corollary 9. *For the* 1*-NN accuracy utility and a* fork *pipeline, which takes as input a dataset of size* N*, the Shapley value can be computed in* $O(N \log N)$ *time.*

We present the proof in Section A.4.6 in the appendix.

Dataset	Modality	# Examples	# Features
UCIAdult [125]	tabular	$49K$	14
Folktables [126]	tabular	$1.6M$	10
FashionMNIST [127]	image	$14K$	(Image) 28×28
20NewsGroups [128]	text	$1.9K$	(Text) $20K$ after TF-IDF
Higgs [129]	tabular	$11M$	28

TABLE 4.1: Datasets characteristics

4.6 Experimental Evaluation

We evaluate the performance of DataScope when applied to data debugging and repair. In this section, we present the empirical study we conducted with the goal of evaluating both quality and speed.

4.6.1 Experimental Setup

HARDWARE AND PLATFORM. All experiments were conducted on Amazon AWS c5.metal instances with a 96-core Intel(R) Xeon(R) Platinum 8275CL 3.00GHz CPU and 192GB of RAM. We ran each experiment in single-thread mode.

DATASETS. We assemble a collection of widely used datasets with diverse modalities (i.e. tabular, textual, and image datasets). Table 4.1 summarizes the datasets that we used.

(Tabular Datasets) UCIAdult is a tabular dataset from the US census data [125]. We use the binary classification variant where the goal is to predict whether the income of a person is above or below $50K. One of the features is 'sex,' which we use as a *sensitive attribute* to measure group fairness with respect to male and female subgroups. A very similar dataset is Folktables, which was developed to redesign and extend the original UCIAdult dataset with various aspects interesting to the fairness community [126]. We use the 'income' variant of this dataset, which also has a 'sex' feature and has a binary label corresponding to the $50K income threshold. Another tabular dataset that we use for large-scale experiments is the

Pipeline	Dataset Modality	w/ Reduce	Operators
Identity	tabular	false	$\emptyset$
StandardScaler	tabular	true	`StandardScaler`
LogarithmicScaler	tabular	true	`Log1P` $\circ$ `StandardScaler`
PCA	tabular	true	`PCA`
MissingIndicator + KMeans	tabular	true	`MissingIndicator` $\oplus$ `KMeans`
GaussianBlur	image	false	`GaussBlur`
HistogramofOrientedGradients	image	false	`HogTransform`
TFIDF	text	true	`CountVectorizer` $\circ$ `TfidfTransformer`
Tolower + URLRemove + TFIDF	text	false	`TextToLower` $\circ$ `UrlRemover` $\circ$ `CountVectorizer` $\circ$ `TfidfTransformer`

TABLE 4.2: Feature extraction pipelines used in experiments.

Higgs dataset, which has 28 features that represent physical properties of particles in an accelerator [129]. The goal is to predict whether the observed signal produces Higgs bosons or not.

(Non-tabular Datasets) We used two non-tabular datasets. One is FashionMNIST, which contains 28×28 grayscale images of 10 different categories of fashion items [127]. To construct a binary classification task, we take only images of the classes 'shirt' and 'T-shirt.' We also use TwentyNewsGroups, which is a dataset with text obtained from newsgroup posts categorized into 20 topics [128]. To construct a binary classification task, we take only two newsgroup categories, 'sci.med' and 'comp.graphics.' The task is to predict the correct category for a given piece of text.

FEATURE PROCESSING PIPELINES. We obtained a dataset with about $500K$ machine learning workflow instances from internal Microsoft users [119]. Each workflow consists of a dataset, a feature extraction pipeline, and an ML model. We identified a handful of the most representative pipelines and translated them to `sklearn` pipelines. We list the pipelines used in our experiments in Table 4.2.

As Table 4.2 shows, we used pipelines of varying complexity. The data modality column indicates which types of datasets we applied each pipeline to. Some pipelines are pure map pipelines, while some implicitly require a reduce operation. Table 4.2 shows the operators contained by each pipeline. They are combined either using a composition symbol $\circ$, i.e., operators are applied in sequence; or a concatenation

symbol $\oplus$, i.e., operators are applied in parallel and their output vectors are concatenated. Some operators are taken directly from `sklearn` (`StandardScaler`, `PCA`, `MissingIndicator`, `KMeans`, `CountVectorizer`, and `TfidfTransformer`), while others require customized implementations: (1) `Log1P`, using the `log1p` function from `numpy`; (2) `GaussBlur`, using the `gaussian_filter` function from `scipy`; (3) `HogTransform`, using the `hog` function from `skimage`; (4) `TextToLower`, using the built-in `tolower` Python function; and (5) `UrlRemover`, using a simple regular expression.

Fork Variants: We also create a "fork" version of the above pipelines, by prepending each with a `DataProvider` operator. It simulates distinct data providers that each provides a portion of the data. The original dataset is split into a given number of groups (we set this number to 100 in our experiments). We compute importance for each group, and we conduct data repairs on entire groups all at once.

MODELS. We use three machine learning models as the downstream ML model following the previous feature extraction pipelines: XGBoost, LogisticRegression, and KNearestNeighbor. We use the `LogisticRegression` and `KNeighborsClassifier` provided by the `sklearn` package. We use the default hyper-parameter values except that we set `max_iter` to 5,000 for LogisticRegression and `n_neighbors` to 1 for the KNearestNeighbor.

DATA DEBUGGING METHODS. We apply different data debugging methods and compare them based on their effect on model quality and the computation time that they require:

- Random — We measure importance with a random number and thus apply data repairs in random order.
- TMCShapleyx10 and TMCShapleyx100 — We express importance as Shapley values computed using the Truncated Monte-Carlo (TMC) method [77], with 10 and 100 Monte-Carlo iterations, respectively. We then follow the computed importance in ascending order to repair data examples.
- DataScope — This is our K-nearest-neighbor based method for efficiently computing the Shapley value. We then follow the computed importance in ascending order to repair data examples.

- DataScopeInteractive — While the above methods compute importance scores only once at the beginning of the repair, the speed of DataScope allows us to *recompute* the importance after *each* data repair. We call this strategy DataScopeInteractive.

PROTOCOL. In most of our experiments (unless explicitly stated otherwise), we simulate importance-driven data repair scenarios performed on a given *training dataset*. In each experimental run, we select a dataset, pipeline, model, and a data repair method. We compute the importance using the utility defined over a *validation set*. Training data repairs are conducted one unit at a time until all units are examined. The order of units is determined by the specific repair method. We divide the range between 0% data examined and 100% data examined into 100 checkpoints. At each checkpoint we measure the quality of the given model on a separate *test dataset* using some metric (e.g. accuracy). For importance-based repair methods, we also measure the time spent on computing importance. We repeat each experiment 10 times and report the median as well as the 90-th percentile range (either shaded or with error bars).

4.6.2 Results

Following the protocol of [36], [39], we start by flipping certain amount of labels in the training dataset. We then use a given data debugging method to go through the dataset and repair labels by replacing each label with the correct one. As we progress through the dataset, we measure the model quality on a separate test dataset using a metric such as accuracy or equalized odds difference (a commonly used fairness metric). Our goal is to achieve the best possible quality while at the same time having to examine the least possible amount of data. Depending on whether the pipeline is an original one or its fork variant, we have slightly different approaches to label corruption and repair. For original pipelines, each label can be flipped with some probability (by default this is 50%). Importance is computed for independent data examples, and repairs are performed independently as well. For fork variants, data examples are divided into groups corresponding to their respective data providers. By default, we set the number of data providers to 100. Each label inside a single group is flipped based on a fixed probability. However, this probability differs across data providers (going from 0% to 100%). Importance

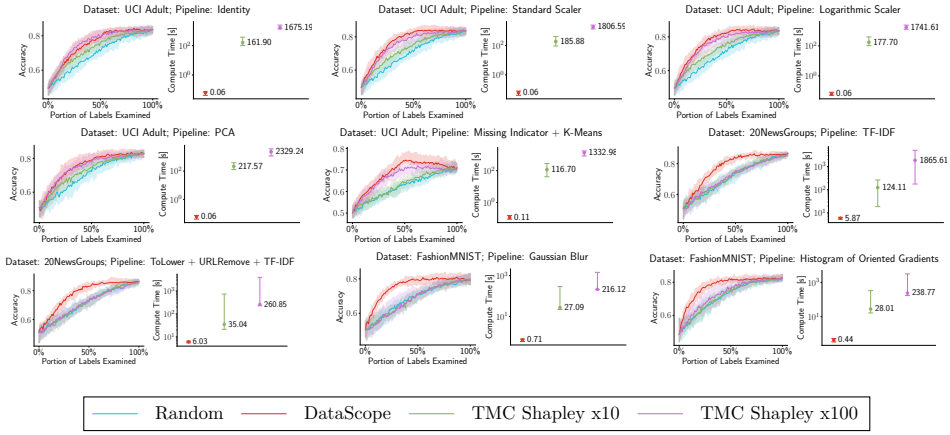

FIGURE 4.7: Label Repair experiment results over various combinations of datasets (1k samples) and map pipelines. We optimize for accuracy. The model is XGBoost.

is computed for individual providers, and when a provider is selected for repair, all its labels get repaired.

IMPROVING ACCURACY WITH LABEL REPAIR. In this set of experiments, we aim to improve the accuracy as much as possible with as least as possible labels examined. We show the case for XGBoost in Figure 4.7

We see that DataScope is significantly faster than TMC-based methods. The speed-up is in the order of 100× to 1,000× for models such as logistic regression. For models requiring slightly longer training time (e.g., XGBoost), the speed-up can be up to 10,000×.

In terms of quality, we see that DataScope is comparable with or better than the TMC-based methods (mostly for the logistic regression model), both outperforming the Random repair method. In certain cases, DataScope, despite its orders of magnitude speed-up, also clearly dominates the TMC-based methods, especially when the pipelines produce features of high-dimensional datasets (such as the text-based pipelines used for the 20NewsGroups dataset and the image-based pipelines used for the FashionMNIST dataset).

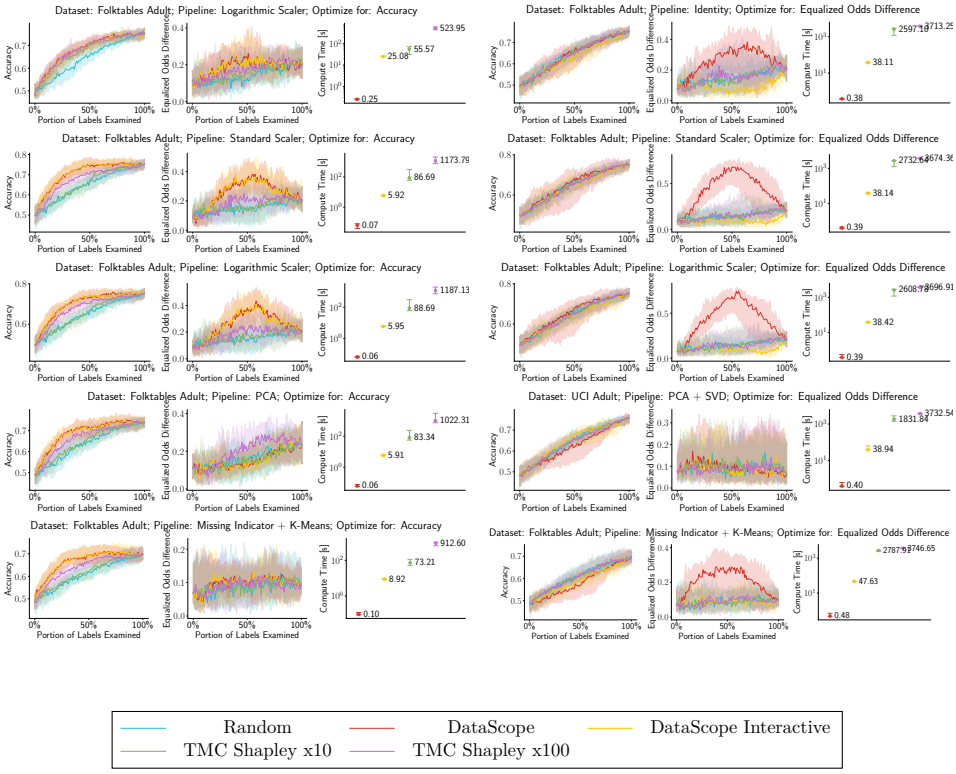

FIGURE 4.8: Label Repair experiment results over various combinations of datasets (1k samples) and map pipelines. We optimize for fairness. The model is XGBoost.

IMPROVING ACCURACY AND FAIRNESS. We then explore the relationship between accuracy and fairness when performing label repairs. Figure 4.8 shows the result for XGBoost over original pipelines. In these experiments we only use the two tabular datasets UCIAdult and Folktables, which have a 'sex' feature that we use to compute group fairness using *equalized odds difference*, one of the commonly used fairness metrics [123]. We use equalized odds difference as the utility function for both DataScope and TMC-based methods.

We first see that being able to debug specifically for fairness is important — the left panel of Figure 4.8 illustrates the behavior of optimizing for accuracy whereas the right panel illustrates the behavior of optimizing for fairness. In this example, the 100% clean dataset is unfair. When optimizing for accuracy, we see that the unfairness of model can also increase. On the other hand, when taking fairness into consideration, DataScopeInteractive is able to maintain fairness while improving accuracy significantly — the unfairness increase of DataScopeInteractive only happens at the very end of the cleaning process, where all other "fair" data examples have already been cleaned. This is likely due to the way that the equalized odds difference utility is approximated in DataScope. When computing the utility, we first make a choice on which G_i and G_j to choose in Equation 4.12, as well as a choice between TPR_Δ and FPR_Δ in Equation 4.11; only then we compute the Shapley value. We assume that these choices are stable over the entire process of label repair. However, if these choices are ought to change, only DataScopeInteractive is able to make the necessary adjustment because the Shapley value is recomputed after every repair.

In terms of speed, DataScope significantly outperforms TMC-based methods — in the order of $100\times$ to $1,000\times$ for models like logistic regression and up to $10,000\times$ for XGBoost. In terms of quality, DataScope is comparable to TMC-based methods, while DataScopeInteractive, in certain cases, dramatically outperforms DataScope and TMC-based methods. DataScopeInteractive achieves much better fairness (measured by equalized odds difference, lower the better) while maintaining similar, if not better, accuracy compared with other methods. When optimizing for fairness we can observe that sometimes non-interactive methods suffer in pipelines that use standard scalers. It might be possible that importance scores do not remain stable over the course of our data repair process. Because equalized odds difference is a non-trivial measure, even though it may work in the beginning of our process, it might mislead us in the wrong direction after some portion of the labels get repaired. As a result, being able to compute data importance frequently, which is

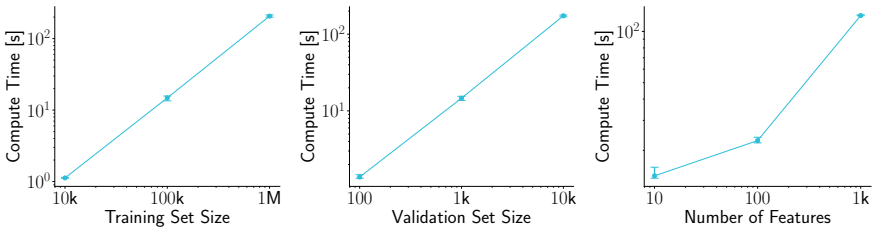

FIGURE 4.9: Scalability of DataScope

enabled by our efficient algorithm, is crucial to effectively navigate and balance accuracy and fairness.

SCALABILITY. We now evaluate the quality and speed of DataScope for larger training datasets. We test the runtime for various sizes of the training set ($10k$-$1M$), the validation set (100-$10k$), and the number of features (100-$1k$). As expected, the impact of training set size and validation set size is roughly linear. Furthermore, we see that even for large datasets, DataScope can compute Shapley scores in minutes.

When integrated into an intearctive data repair workflow, this could have a dramatic impact on the productivity of data scientists. We have clearly observed that Monte Carlo approaches do improve the efficiency of importance-based data debugging. However, given their lengthy runtime, one could argue that many users would likely prefer not to wait and consequently end up opting for the random repair approach. What DataScope offers is a viable alternative to random which is equally attainable while at the same time offering the significant efficiency gains provided by Shapley-based importance.

4.7 Conclusion

In this chapter we have presented ease.ml/datascope, the first system that efficiently computes Shapley values of training examples over an *end-to-end* ML pipeline. Our core contribution is a novel algorithmic framework that computes Shapley value over a specific family of ML pipelines that we call *canonical pipelines*. For many subfamilies of canonical pipelines, computing Shapley value is in PTIME,

contrasting the exponential complexity of computing Shapley value in general. Our experimental evaluation results show that ease.ml/datascope is up to four orders of magnitude faster over state-of-the-art Monte Carlo-based methods, while being comparably, and often even more, effective in data debugging.

In Listing 4.1, we have seen an example of invoking ease.ml/datascope directly from the workflow code. In this case it is used as a stand alone tool for explainability of ML pipelines. On top of that, when integrated into the ease.ml workflow, it is also possible to apply ease.ml/datascope in the same way as ease.ml/cpclean can be applied in the context of manual label repair (as described in Chapter 1).

5

DISCUSSION, CONCLUSION AND FUTURE WORK

The maturity and richness of the field of applied machine learning revealed many novel problems. As we've mentioned in the introduction, the field of "systems for ML" is taking off. However, if we look at the field of regular software development, the development experience and the usability of so many tools that are available for developers, it becomes clear that a lot of work still needs to be done. We are still relatively far off from realizing the dream of having a real IDE for ML with a rich set of Dev/Ops tools, standards and practices.

In this thesis we have attempted to make a tiny step in that direction. Based on our direct and indirect experience with applied machine learning, we have identified an array of commonly occuring pain points. We described these pain points in Chapter 2 and provide a toolkit of proposed system components that are meant to overcome them.

We then zoomed into one particular pain point – labor efficient data debugging guided by some measure of importance. We identified two types of data errors: missing values and wrong values, and propose two measures of importance to tackle them. They were, respectively: information gain (in Chapter 3) and Shapley value (in Chapter 4). We conducted experimental evaluations of those two measures of importance in order to test their effectiveness at their respective data debugging tasks.

Finally, in this chapter we will briefly go over some lessons learned and proposed recipes for designing *usable* systems for managing machine learning workflows. If we measure usability as the absence of time wasted on unproductive work, then we argue that our approach for data debugging guided by measures of importance indeed improves usability of the system.

5.1 Insights for Building Usable ML Workflow Management Systems

In previous chapters of this thesis, we went deeper into our proposed methods for solving one particular technical problem related to one particular pain point – labor-efficient data repair. Now we will zoom back out and go over some high-level insights that we have gained while conducting this research. Our hope is that these insights could serve as recipes for designing more usable machine learning systems in the future.

5.1.1 The Data Model is Probabilistic

One key realization that we have, only after we constructed each of the individual components, is that all user-facing components can be modeled via the same data model — a *probabilistic database* in which uncertainty is induced by dirty data, unlabeled examples, weak supervision, etc. Moreover, all human interactions can be modeled as the process of *mutual information maximization*, i.e., eliminating uncertainties to maximize the "closeness" to an unknown, "ground-truth" world. This unified framework is often expensive to implement, and one often needs to develop non-trivial proxies and approximations for each individual component. Note that this particular view definitely has its own limitations, which we will discuss in the next section. Nevertheless, it does provide a principled, yet practical, framework to model human interactions in an end-to-end ML process.

The logical data model of ease.ml is *a probabilistic database [45] in which each row has independent uncertainty.* This is our key design decision, which provides a unified abstraction for (1) noise in features and labels; (2) weak supervision (e.g., Snorkel) that provides multiple alternatives to labels; (3) availability of multiple state-of-the-art data cleaning tools that provide multiple candidates of the cleaned value; (4) unknown label values for data and label acquisition; and (5) data augmentation. This design decision was entirely unclear to us at the beginning — only after we put all components together, we realized how probabilistic databases would provide a clean, unified abstraction for almost all ease.ml steps.

LOGICAL DATA MODEL Let $\mathcal{X}$ be the feature domain and let $\mathcal{Y}$ be the label domain. Note that, despite its name, the label domain does not need to be categorical as in supervised classification. For example, both domains can be images for an image translation task. A *dataset* contains pairs of *random variables*, taking values in the feature domain and the label domain, respectively:

$$\mathbf{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : \mathbf{x}_i \in \mathcal{X}, \mathbf{y}_i \in \mathcal{Y}\}, \tag{5.1}$$

and it induces two probability distributions

$$\Pr_{\mathbf{D}}[\mathbf{x}_i = X], \quad X \in \mathcal{X}; \qquad \Pr_{\mathbf{D}}[\mathbf{y}_i = Y], \quad Y \in \mathcal{Y}. \tag{5.2}$$

As a concrete example, $\mathbf{x}_i$ can be the output of multiple different state-of-the-art automatic data cleaning tools, each of which forms one candidate value that $\mathbf{x}_i$ can take for the i^{th} data example and $\Pr_{\mathbf{D}}[\mathbf{x}_i = X]$ is our prior on their relative accuracy. $\mathbf{y}_i$ can be the output of multiple labeling functions, each of which forms one candidate label that $\mathbf{y}_i$ can take for the i^{th} data example and $\Pr_{\mathbf{D}}[\mathbf{y}_i = Y]$ is our prior confidence on each of the candidate labels. Note that, in our setting, an *unlabeled dataset* is simply one in which

$$\Pr_{\mathbf{D}}[\mathbf{y}_i = Y] \tag{5.3}$$

is a uniform distribution. One particular direction we take in ease.ml is that there exists a single, *unknown ground truth configuration* for each uncertainty example $(\mathbf{x}_i, \mathbf{y}_i)$, and the *ground truth dataset* $G^*_{\mathbf{D}}$ is

$$G^*_{\mathbf{D}} = \{(x_i^*, y_i^*) : (\mathbf{x}_i, \mathbf{y}_i) \in \mathbf{D}\} \tag{5.4}$$

where (x_i^*, y_i^*) is the ground-truth value of $(\mathbf{x}_i, \mathbf{y}_i)$. In practice, the ground-truth dataset corresponds to the clean dataset if the uncertainty is induced by dirty data, or true labels if the uncertainty is induced by unlabeled examples and weak supervision.

INPUT TO EASE.ML The input to ease.ml is four datasets, all with the *same* feature domain and label domain: (1) **Labeled Training Set $\mathbf{D}_{tr}$**, dataset to train an ML model – both features and labels have uncertainty; (2) **Pool of Unlabeled Data Examples $\mathbf{D}_{unl}$**, pool of data examples that are unlabeled and can be acquired to be included into the training set – features have uncertainty and labels have uniform prior (unlabeled); (3) **Labeled Validation Set $\mathbf{D}_{val}$**, validation dataset that can be used during development – features and labels *do*

not have uncertainty, i.e., we have access to $G^*_{\mathbf{D}_{val}}$; (4) **Unlabeled Production Test Set $\mathbf{D}_{prod}$**, test set that comes from production – only labels have uncertainty (unlabeled).

5.1.2 ML Models are Data Arifacts

A core feature of our design is that ML models are treated as *data artifacts*. This means that the system can manage them as any other tuple in a relation, except that they are also executable equivalently to a UDF. A similar approach is taken for any other ML pipeline operator. They can be assembled together into a single workflow, stored, loaded, executed, etc.

We apply ML training only over a deterministic dataset, which is the assumption of most ML models. A deterministic dataset is one without uncertainty: $D \subseteq \mathcal{X} \times \mathcal{Y}$. Given a deterministic dataset D, the training process of ML returns an ML model that is a mapping $\mathcal{A}_D$: $\mathcal{X} \mapsto \mathcal{Y}$. When we apply the training process over an uncertain dataset $\mathbf{D}$, it returns a random variable, $\mathcal{A}_{\mathbf{D}}$, taking values over all possible classifiers.

The inference process of a given model $\mathcal{A}_D$ is to apply such a mapping to another deterministic dataset D': $\mathcal{A}_D(D')$. We use similar notation for datasets $\mathbf{D}$ and $\mathbf{D}'$ with uncertainties, in which $\mathcal{A}_{\mathbf{D}}(\mathbf{D}')$ returns a random variable, depending on the randomness in $\mathbf{D}$ and $\mathbf{D}'$.

5.1.3 Workflows are Data Artifacts

Each manipulation is tracked and stored in a lineage graph that is used to represent the user's ML workflow. This graph can then be optimized, materialized, or stored for later use. A lineage graph that is saved can also be executed at a later point, e.g. through a command line shell or as part of some CI/CD workflow. Furthermore, the user can ask the system to give suggestions for next steps given the current state of the lineage graph.

As mentioned earlier, the logical data model we assume for all ML datasets is the probabilistic relational model where each row is associated with a single data

example (i.e. a sample from a data distribution). Each data example is furthermore associated with one or more possible candidate rows from which only one can be selected at any given time. Each column has a type that can be either a simple scalar value (e.g. numeric, Boolean, categorical, string, etc.) or a more complex type (e.g. document, tensor, etc.) This data model is implemented with the `Relation` class that wraps around the `DataFrame` class taken from the popular Pandas framework which we extend with the mentioned capabilities.

To manipulate the data stored in our data model, the user would run various available operators. These operators work in a staged manner whereby invoking an operator merely appends it to the lineage graph. The return value of the operator invocation is the updated lineage graph. At any point in the interactive ML process the user can choose to materialize the graph by calling the `eval()` function. This process involves performing various optimizations after which the resulting data view is shown in the output cell of the Jupyter notebook while at the same time being cached for future use. This approach allows us to treat ML workflows as first-class citizens that can be managed like any other data artifact.

5.1.4 Provenance is a First-class Citizen

As we have seen in Chapter 4, we absolutely need provenance information in order to run our algorithms. Therefore, information needs to be available as an integral part of any data object. At the very least this needs to be true on the logical level. In practice, due to the potential overhead associated with provenance tracking, we could probably opt for lazily evaluating this information. Operators such as the one that implements ease.ml/datascope could "declare" that they need provenance information. As mentioned earlier, our workflow graphs are lazily evaluated, which makes it easier to decide if we are going to need provenance information by simply analyzing the workflow and checking if any operator requires it.

5.1.5 The Development Environment Revolves around Jupyter Notebooks

One key design decision is to base our system with Jupyter notebooks as it not only represents a familiar and powerful platform to many data scientists but also provides naturally all interactions that ease.ml has with its users.

The user interaction in Jupyter notebooks is conducted through "code cells" where the user types in usually short code snippets and then executes them, after which the output is presented immediately below the cell. We use the same approach and we extend it with slightly enriched output cells that can contain various UI items such as buttons, progress bars and interactive graphs. The user loads the data after which she can execute any sequence of data manipulation operations (e.g. input preprocessing, cleaning, filtering) intertwined with various ease.ml operations (e.g. ease.ml/snoopy or ease.ml/automl).

5.1.6 Guide the User through the Entire ML Workflow

An additional feature that we envision is intended to solve the problem of "What should I do next?" This is a question that the user is likely ask themselves many times during their ML development process. For this we intend a whatnext command that could be invoked at any point in time and, given the context if the current workflow, provide recommendations to the user on what step they should perform next.

5.2 Future Work

As the field of machine learning continues its path into virtually all areas of software development, the problems described in this thesis will only get exarcerbated. Of course, the work we present in this thesis merely scratches the surface. In this section we list some ideas for future research directions and group them based on the topic.

THE EASE.ML SYSTEM. Although we already have previous work that covers many of the system components described in Chapter 2, there are still some elements of the system that are not fully clear. The first one is ease.ml/datamagic because, although we already have some work done in the space of document ingestion, we still cannot provide automated ingestion of arbitrary datasets. Another open problem is that there are no non-trivial methods for implementing the whatnext command. This could merit a real user study on the particular pain points and decisions that users make which are likely to lead them to positive outcomes.

MISSING DATA REPAIR. If we look at ease.ml/cpclean and we compare it with ease.ml/datascope, one limitation becomes clear immediately – ease.ml/cpclean does not assume the existence of an end-to-end ML pipeline that precedes ML model training. This could easily serve as one potential research direction. Another, slightly more open-ended direction, would be to extend the notion of incomplete information to other applications, beyond data cleaning. One that we already explored has to do with certifying model robustness. But beyond that, the notion of uncertainty pops up in various places. Essentially, every decision that we make along the ML workflow is a source of uncertainty. So, one idea would be to have e.g. a "checking query" that checks if the choice between multiple pipeline operators makes any difference in the quality of the model. The goal would then be to integrate the notion of uncertainty into the model optimization process itself.

WRONG DATA REPAIR. When it comes to ease.ml/datascope, even though we have demonstrated that it can support a variety of utility functions, the question remains if we can push that even further. Sometimes pipelines have post-processing operations which could also resemble some positive relational algebra queries, such as the ones studied by the Rain system [9]. It would be interesting to test the theoretical limits of ease.ml/datascope in terms of post processing pipelines.

BIBLIOGRAPHY

[1] S. Karayev, J. Tobin, and P. Abbeel, *Full sack deep learning*, `https://fullstackdeeplearning.com/spring2021/lecture-6/`, 2021.

[2] L. A. deeplearning.ai, "Data-centric ai competition", 2021.

[3] *Labelbox*, `https://labelbox.com/`, Accessed: 2022-05-17.

[4] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference", *Proceedings of the VLDB Endowment*, vol. 10, no. 11, 1190, 2017.

[5] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision", in *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, NIH Public Access, vol. 11, 2017, 269.

[6] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, "Zeroer: Entity resolution using zero labeled examples", in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, 1149.

[7] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform", in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, 1387.

[8] E. Breck, N. Polyzotis, S. Roy, S. Whang, and M. Zinkevich, "Data validation for machine learning", *Proceedings of Machine Learning and Systems*, vol. 1, 334, 2019.

[9] W. Wu, L. Flokas, E. Wu, and J. Wang, "Complaint-driven training data debugging for query 2.0", in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, 1317.

[10] S. Nakandala, A. Kumar, and Y. Papakonstantinou, "Incremental and approximate inference for faster occlusion-based deep cnn explanations", in *Proceedings of the 2019 International Conference on Management of Data*, 2019, 1589.

[11] S. Nakandala, Y. Zhang, and A. Kumar, "Cerebro: A data system for optimized deep learning model selection", *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 2159, 2020.

[12] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "Modeldb: A system for machine learning model management", in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 2016, 1.

[13] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, *et al.*, "Accelerating the machine learning lifecycle with mlflow.", 2018.

[14] C. Zhang, C. Ré, M. Cafarella, C. De Sa, A. Ratner, J. Shin, F. Wang, and S. Wu, "Deepdive: Declarative knowledge base construction", *Communications of the ACM*, vol. 60, no. 5, 93, 2017.

[15] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning for statistical modeling", *Proceedings of the VLDB Endowment*, vol. 9, no. 12, 948, 2016.

[16] T. Kraska, "Northstar: An interactive data science system", *Proceedings of the VLDB Endowment*, vol. 11, no. 12, 2150, 2018.

[17] K. Schawinski, C. Zhang, H. Zhang, L. Fowler, and G. K. Santhanam, "Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit", *Monthly Notices of the Royal Astronomical Society: Letters*, vol. 467, no. 1, L110, 2017.

[18] M. Su, H. Zhang, K. Schawinski, C. Zhang, and M. A. Cianfrocco, "Generative adversarial networks as a tool to recover structural information from cryo-electron microscopy data", *BioRxiv*, 256792, 2018.

[19] L. F. Sartori, K. Schawinski, B. Trakhtenbrot, N. Caplar, E. Treister, M. J. Koss, C. Megan Urry, and C. Zhang, "A model for agn variability on multiple time-scales", *Monthly Notices of the Royal Astronomical Society: Letters*, vol. 476, no. 1, L34, 2018.

[20] D. Stark, B. Launet, K. Schawinski, C. Zhang, M. Koss, M. D. Turp, L. F. Sartori, H. Zhang, Y. Chen, and A. K. Weigel, "Psfgan: A generative adversarial network system for separating quasar point sources and host galaxy light", *Monthly Notices of the Royal Astronomical Society*, vol. 477, no. 2, 2513, 2018.

[21] S. Ackermann, K. Schawinski, C. Zhang, A. K. Weigel, and M. D. Turp, "Using transfer learning to detect galaxy mergers", *Monthly Notices of the Royal Astronomical Society*, vol. 479, no. 1, 415, 2018.

[22] I. Girardi, P. Ji, A.-p. Nguyen, N. Hollenstein, A. Ivankay, L. Kuhn, C. Marchiori, and C. Zhang, "Patient risk assessment and warning symptom detection using deep attention-based neural networks", in *The Ninth International Workshop on Health Text Mining and Information Analysis*, 2018.

[23] N. Glaser, O. I. Wong, K. Schawinski, and C. Zhang, "Radiogan–translations between different radio surveys with generative adversarial networks", *Monthly Notices of the Royal Astronomical Society*, vol. 487, no. 3, 4190, 2019.

[24] B. Karlaš, P. Li, R. Wu, N. M. Gürel, X. Chu, W. Wu, and C. Zhang, "Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions", *Proc. VLDB Endow.*, vol. 14, no. 3, 255, 2020.

[25] C. Renggli, B. Karlaš, B. Ding, F. Liu, K. Schawinski, W. Wu, and C. Zhang, "Continuous integration of machine learning models with ease.ml/ci: A rigorous yet practical treatment", in *Proceedings of Machine Learning and Systems*, 2019.

[26] B. Karlaš, M. Interlandi, C. Renggli, W. Wu, C. Zhang, D. M. I. Babu, J. Edwards, C. Lauren, A. Xu, and M. Weimer, "Building continuous integration services for machine learning", in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

[27] C. Renggli, F. A. Hubis, B. Karlaš, K. Schawinski, W. Wu, and C. Zhang, "Ease.ml/ci and ease.ml/meter in action: Towards data management for statistical generalization", *Proceedings of the VLDB Endowment*, 2019.

[28] C. Yu, B. Karlaš, J. Zhong, C. Zhang, and J. Liu, "Automl from service providers perspective: Multi-device, multi-tenant model selection with gp-ei", in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, 2829.

[29] B. Karlaš, J. Liu, W. Wu, and C. Zhang, "Ease.ml in action: Towards multi-tenant declarative learning services", *Proceedings of the VLDB Endowment*, 2018.

[30] M. R. Karimi, N. M. Gürel, B. Karlaš, J. Rausch, C. Zhang, and A. Krause, "Online active model selection for pre-trained classifiers", in *International Conference on Artificial Intelligence and Statistics*, 2021.

[31] G. Russo, N. Hollenstein, C. C. Musat, and C. Zhang, "Control, generate, augment: A scalable framework for multi-attribute text generation", in *EMNLP (Findings)*, 2020.

[32] J. Rausch, O. Martinez, F. Bissig, C. Zhang, and S. Feuerriegel, "Docparser: Hierarchical document structure parsing from renderings", in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, vol. 35, 2021, 4328.

[33] C. Renggli, L. Rimanic, L. Kolar, N. Hollenstein, W. Wu, and C. Zhang, "Ease.ml/snoopy: Towards automatic feasibility study for machine learning applications", *arXiv preprint arXiv:2010.08410*, 2020.

[34] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. G"urel, B. Li, C. Zhang, D. Song, and C. J. Spanos, "Towards efficient data valuation based on the shapley value", in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, 1167.

[35] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gurel, B. L. C. Zhang, and C. S. D. Song, "Efficient task-specific data valuation for nearest neighbor algorithms", *Proceedings of the VLDB Endowment*, vol. 12, no. 11, 2019.

[36] R. Jia, F. Wu, X. Sun, J. Xu, D. Dao, B. Kailkhura, C. Zhang, B. Li, and D. Song, "Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification?", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, 8239.

[37] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang, "Ease.ml: Towards multi-tenant resource sharing for machine learning workloads", *Proceedings of the VLDB Endowment*, vol. 11, no. 5, 607, 2018.

[38] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu, "Boostclean: Automated error detection and repair for machine learning", *arXiv preprint arXiv:1711.01299*, 2017.

[39] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "Cleanml: A benchmark for joint data cleaning and machine learning [experiments and analysis]", *arXiv preprint arXiv:1904.09483*, 2019.

[40] Y. Chen, H. Hassani, A. Karbasi, and A. Krause, "Sequential information maximization: When is greedy near-optimal?", in *Conference on Learning Theory*, 2015.

[41] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases: The Logical Level*, 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[42] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases", in *Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, 1999, 68.

[43] A. Lopatenko and L. E. Bertossi, "Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics", in *Proc. 11th Int. Conf. on Database Theory*, 2007, 179.

[44] L. E. Bertossi, *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.

[45] D. Suciu, D. Olteanu, C. Ré, and C. Koch, "Probabilistic databases", *Synthesis lectures on data management*, vol. 3, no. 2, 1, 2011.

[46] P. J. García-Laencina, J.-L. Sancho-Gómez, and A. R. Figueiras-Vidal, "Pattern classification with missing data: A review", *Neural Computing and Applications*, vol. 19, no. 2, 263, 2010.

[47] D. B. Rubin, "Multiple imputation after 18+ years", *Journal of the American Statistical Association*, vol. 91, no. 434, 473, 1996.

[48] P. Khosravi, Y. Liang, Y. Choi, and G. V. den Broeck, "What to expect of classifiers? reasoning about logistic regression with missing features", *CoRR*, vol. abs/1903.01620, 2019.

[49] I. F. Ilyas and X. Chu, *Data Cleaning*. ACM, 2019.

[50] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo, "A sample-and-clean framework for fast and accurate query processing on dirty data", in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2014, 469.

[51] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing", in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2015, 1247.

[52] P. K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang, "Nearest-neighbor searching under uncertainty", in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '12, Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, 225.

[53] P. K. Agarwal, B. Aronov, S. Har-Peled, J. M. Phillips, K. Yi, and W. Zhang, "Nearest-neighbor searching under uncertainty ii", *ACM Trans. Algorithms*, vol. 13, no. 1, 2016.

[54] H.-P. Kriegel, P. Kunath, and M. Renz, "Probabilistic nearest-neighbor query on uncertain objects", in *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, ser. DASFAA'07, Bangkok, Thailand: Springer-Verlag, 2007, 337.

[55] A. Krause and D. Golovin, "Submodular function maximization", in *Tractability*, 2014.

[56] K. Fujii and S. Sakaue, "Beyond adaptive submodularity: Approximation guarantees of greedy policy with adaptive submodularity ratio", in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, Long Beach, California, USA: PMLR, 2019, 2042.

[57] H. Esfandiari, A. Karbasi, and V. Mirrokni, "Adaptivity in adaptive submodularity", in *Conference on Learning Theory*, PMLR, 2021, 1823.

[58] Y. Chen and A. Krause, "Near-optimal batch mode active learning and adaptive submodular optimization", in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, ser. JMLR Workshop and Conference Proceedings, vol. 28, JMLR.org, 2013, 160.

[59] S. Dasgupta, "Analysis of a greedy active learning strategy", in *Neural Information Processing Systems*, 2004.

[60] A. X. Zheng, I. Rish, and A. Beygelzimer, "Efficient test selection in active diagnosis via entropy approximation", in *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, ser. UAI'05, Edinburgh, Scotland: AUAI Press, 2005, 675.

[61] C. Wa Ko, J. Lee, and M. Queyranne, "An exact algorithm for maximum entropy sampling", *Oper. Res.*, vol. 43, no. 4, 684, 1995.

[62] S. Das, A. Doan, P. S. G. C., C. Gokhale, P. Konda, Y. Govind, and D. Paulsen, *The magellan data repository*, `https://sites.google.com/site/anhaidgroup/projects/data`.

[63] J. S. Simonoff, *Analyzing categorical data*. Springer Science & Business Media, 2013.

[64] C. E. Rasmussen, R. M. Neal, G. E. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, "The delve manual", *URL http://www. cs. toronto. edu/~ delve*, 1996.

[65] D. B. Rubin, "Inference and missing data", *Biometrika*, vol. 63, no. 3, 581, 1976.

[66] A. Ratner, D. Alistarh, G. Alonso, D. G. Andersen, P. Bailis, S. Bird, N. Carlini, B. Catanzaro, E. Chung, B. Dally, *et al.*, "SysML: The new frontier of machine learning systems", 2019.

[67] J. Liu, C. Zhang, *et al.*, "Distributed learning systems with First-Order methods", *Foundations and Trends® in Databases*, vol. 9, no. 1, 1, 2020.

[68] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "PyTorch distributed: Experiences on accelerating data parallel training", *Proceedings VLDB Endowment*, vol. 13, no. 12, 3005, 2020.

[69] S. Gan, J. Jiang, B. Yuan, C. Zhang, X. Lian, R. Wang, J. Chang, C. Liu, H. Shi, S. Zhang, X. Li, T. Sun, S. Yang, and J. Liu, "Bagua: Scaling up distributed learning with system relaxations", *Proceedings VLDB Endowment*, vol. 15, no. 4, 804, 2021.

[70] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow", 2018.

[71] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters", in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, 463.

[72] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures", Proceedings of Machine Learning Research, vol. 28, no. 1, S. Dasgupta and D. McAllester, Eds., 115, 2013.

[73] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning", 2016.

[74] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning", in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, 2755.

[75] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions", *International conference on machine learning*, Proceedings of Machine Learning Research, vol. 70, D. Precup and Y. W. Teh, Eds., 1885, 2017.

[76] P. W. W. Koh, K.-S. Ang, H. Teo, and P. S. Liang, "On the accuracy of influence functions for measuring group effects", *Advances in neural information processing systems*, vol. 32, 2019.

[77] A. Ghorbani and J. Zou, "Data shapley: Equitable valuation of data for machine learning", in *International Conference on Machine Learning*, PMLR, 2019, 2242.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python", *the Journal of machine Learning research*, vol. 12, 2825, 2011.

[79] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, *et al.*, "Mllib: Machine learning in apache spark", *The Journal of Machine Learning Research*, vol. 17, no. 1, 1235, 2016.

[80] A. Adadi and M. Berrada, "Peeking inside the Black-Box: A survey on explainable artificial intelligence (XAI)", *IEEE Access*, vol. 6, 52138, 2018.

[81] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models", *ACM Comput. Surv.*, vol. 51, no. 5, 1, 2018.

[82] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning", in *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, IEEE, 2018, 80.

[83] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" explaining the predictions of any classifier", in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, 1135.

[84] S. Krishnan and E. Wu, "Palm: Machine learning explanations for iterative debugging", in *Proceedings of the 2Nd workshop on human-in-the-loop data analytics*, 2017, 1.

[85] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *European conference on computer vision*, Springer, 2014, 818.

[86] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences", in *International conference on machine learning*, PMLR, 2017, 3145.

[87] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks", in *International conference on machine learning*, PMLR, 2017, 3319.

[88] I. Covert, S. Lundberg, and S.-I. Lee, "Explaining by removing: A unified framework for model explanation", *Journal of Machine Learning Research*, vol. 22, no. 209, 1, 2021.

[89] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions", *Advances in neural information processing systems*, vol. 30, 2017.

[90] J. Pei, "Data pricing–from economics to data science", in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, 3553.

[91] S. Basu, X. You, and S. Feizi, "On Second-Order group influence functions for Black-Box predictions", Proceedings of Machine Learning Research, vol. 119, H. D. Iii and A. Singh, Eds., 715, 2020.

[92] B. Sharchilev, Y. Ustinovskiy, P. Serdyukov, and M. Rijke, "Finding influential training samples for gradient boosted decision trees", in *International Conference on Machine Learning*, PMLR, 2018, 4577.

[93] A. Meliou, S. Roy, and D. Suciu, "Causality and explanations in databases", *Proc. VLDB Endow.*, vol. 7, no. 13, 1715, 2014.

[94] A. Meliou, W. Gatterbauer, and D. Suciu, "Reverse data management", *Proc. VLDB Endow.*, vol. 4, no. 12, 1490, 2011.

[95] B. Kanagal, J. Li, and A. Deshpande, "Sensitivity analysis and explanations for robust query evaluation in probabilistic databases", in *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*, Athens, Greece: ACM Press, 2011.

[96] S. Roy, L. Orr, and D. Suciu, "Explaining query answers with explanation-ready databases", *Proc. VLDB Endow.*, vol. 9, no. 4, 348, 2015.

[97] S. Roy and D. Suciu, "A formal approach to finding explanations for database queries", in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14, Snowbird, Utah, USA: Association for Computing Machinery, 2014, 1579.

[98] A. Meliou and D. Suciu, "Tiresias: The database oracle for how-to queries", in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12, Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, 337.

[99] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance", in *International conference on database theory*, Springer, 2001, 316.

[100] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings", in *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2007, 31.

[101] Y. Amsterdamer, D. Deutch, and V. Tannen, "Provenance for aggregate queries", in *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '11, Athens, Greece: Association for Computing Machinery, 2011, 153.

[102] ——, "On the limitations of provenance for queries with difference", in *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP 11)*, 2011.

[103] S. Grafberger, P. Groth, J. Stoyanovich, and S. Schelter, "Data distribution debugging in machine learning pipelines", *The VLDB Journal*, 1, 2022.

[104] S. Grafberger, S. Guha, J. Stoyanovich, and S. Schelter, "Mlinspect: A data distribution debugger for machine learning pipelines", in *Proceedings of the 2021 International Conference on Management of Data*, 2021, 2736.

[105] S. Schelter, S. Grafberger, S. Guha, O. Sprangers, B. Karlaš, and C. Zhang, "Screening native ml pipelines with "arguseyes"", *CIDR*, 2022.

[106] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning", in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, Chicago, Illinois, USA: Association for Computing Machinery, 2017, 1723.

[107] X. Wang, X. L. Dong, and A. Meliou, "Data x-ray: A diagnostic tool for data errors", in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15, Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, 1231.

[108] S. Nakandala, K. Saur, G.-I. Yu, K. Karanasos, C. Curino, M. Weimer, and M. Interlandi, "A tensor compiler for unified machine learning prediction serving", in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, 899.

[109] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Gowdal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, *et al.*, "Cloudy with high chance of dbms: A 10-year prediction for enterprise-grade ml", 2020.

[110] L. G. Valiant, "The complexity of computing the permanent", *Theor. Comput. Sci.*, vol. 8, no. 2, 189, 1979.

[111] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[112] M. Cadoli and F. M. Donini, "A survey on knowledge compilation", *AI Communications*, vol. 10, no. 3, 4, 137, 1997.

[113] A. Jha and D. Suciu, "Knowledge compilation meets database theory: Compiling queries to decision diagrams", in *ACM International Conference Proceeding Series*, 2011, 162.

[114] C. Y. Lee, "Representation of switching circuits by binary-decision programs", *The Bell System Technical Journal*, vol. 38, no. 4, 985, 1959.

[115] R. E. Bryant, "Graph-based algorithms for boolean function manipulation", *Computers, IEEE Transactions on*, vol. 100, no. 8, 677, 1986.

[116] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebric decision diagrams and their applications", *Formal methods in system design*, vol. 10, no. 2, 171, 1997.

[117] S. Sanner and D. McAllester, "Affine algebraic decision diagrams (aadds) and their application to structured probabilistic inference", in *IJCAI*, vol. 2005, 2005, 1384.

[118] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, "Formal verification using edge-valued binary decision diagrams", *IEEE Transactions on Computers*, vol. 45, no. 2, 247, 1996.

[119] F. Psallidas, Y. Zhu, B. Karlaš, M. Interlandi, A. Floratou, K. Karanasos, W. Wu, C. Zhang, S. Krishnan, C. Curino, *et al.*, "Data science through the looking glass and what we found there", *arXiv preprint arXiv:1912.09536*, 2019.

[120] Y. Zhou, U. Porwal, C. Zhang, H. Q. Ngo, X. Nguyen, C. Ré, and V. Govindaraju, "Parallel feature selection inspired by group testing", in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, 3554.

[121] J. Cheney, L. Chiticariu, and W.-C. Tan, *Provenance in databases: Why, how, and where*. Now Publishers Inc, 2009.

[122] D. Petersohn, S. Macke, D. Xin, W. Ma, D. Lee, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, and A. Parameswaran, "Towards scalable dataframe systems", *Proceedings of the VLDB Endowment*, vol. 13, no. 11,

[123] M. Hardt, E. Price, E. Price, and N. Srebro, "Equality of opportunity in supervised learning", in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016.

[124] S. Barocas, M. Hardt, and A. Narayanan, *Fairness and Machine Learning*. fairmlbook.org, 2019, `http://www.fairmlbook.org`.

[125] R. Kohavi *et al.*, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.", in *Kdd*, vol. 96, 1996, 202.

[126] F. Ding, M. Hardt, J. Miller, and L. Schmidt, "Retiring adult: New datasets for fair machine learning", *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[127] H. Xiao, K. Rasul, and R. Vollgraf. "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms". (2017).

[128] T. Joachims, "A probabilistic analysis of the rocchio algorithm with tfidf for text categorization.", Carnegie-mellon univ pittsburgh pa dept of computer science, Tech. Rep., 1996.

[129] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning", *Nature communications*, vol. 5, no. 1, 1, 2014.

[130] P. Ross, "Generalized hockey stick identities and n-dimensional blockwalking", *The College Mathematics Journal*, vol. 28, no. 4, 325, 1997.

A

APPENDIX

A.1 Correctness Proofs for Certain Prediction Algorithms

A.1.1 Correctness Proof of the Min-Max (MM) Algorithm

Lemma 10. *Let $D^{(1)}, D^{(2)} \in \mathcal{I}_{\mathcal{D}}$ be two possible worlds generated from an incomplete dataset $\mathcal{D}$. Given a test example $t \in \mathcal{X}$ and label $l \in \mathcal{Y}$ where $|\mathcal{Y}| = 2$, let $R_{t,l}$ be a partial ordering relation defined as such:*

$$R_{t,l}\left(D^{(1)}, D^{(2)}\right) := \bigwedge_{i=1}^{N} \left(y_i = l \ \wedge \ \kappa(x_i^{(1)}, t) \leq \kappa(x_i^{(2)}, t)\right) \vee \left(y_i \neq l \ \wedge \ \kappa(x_i^{(1)}, t) \geq \kappa(x_i^{(2)}, t)\right) \tag{A.1}$$

Then, the following relationship holds:

$$R_{t,l}\left(D^{(1)}, D^{(2)}\right) \implies \left(\left(\mathcal{A}_{D^{(1)}}(t) = l\right) \implies \left(\mathcal{A}_{D^{(2)}}(t) = l\right)\right)$$

Proof. We will prove this by contradiction. Consider the case when $\mathcal{A}_{D^{(1)}}(t) = l$ and $\mathcal{A}_{D^{(2)}}(t) \neq l$, that is, possible world $D^{(1)}$ predicts label l but possible world $D^{(2)}$ predicts some other label $l' \neq l$. That means that in the top-K induced by $D^{(2)}$ has to be at least one more data point with label l' than in the top-K induced by $D^{(1)}$. Is it possible for the premise to be true?

The similarities $\kappa(x_i^{(1)}, t)$ and $\kappa(x_i^{(2)}, t)$ cannot all be equal because that would represent equal possible worlds and that would trivially contradict with the premise since the labels predicted by equal possible worlds cannot differ. Therefore, at least one of the $i = 1, ..., N$ inequalities has to be strict. There, we distinguish three possible cases with respect to the class label y_i of the i-th example:

1. $y_i = l$: This means that the similarity of a data point coming from $D^{(2)}$ is higher than the one coming from $D^{(1)}$. However, this could only elevate that data point in the similarity-based ordering and could only increase the number of data points with label l in the top-K. Since $\mathcal{A}_{D^{(1)}}(t) = l$, the prediction of $D^{(2)}$ could not be different, which is **contradicts** the premise.

2. $y_i = l' \neq l$ and $|\mathcal{Y}| = 2$: We have a data point with a label different from l with lowered similarity, which means it can only drop in the ordering. This can not cause an increase of data points with label l' in the top-K, which again **contradicts** the premise.

3. $y_i = l' \neq l$ and $|\mathcal{Y}| \geq 2$: Here we again have a data point with label l' with lowered similarity. However, if that data point were to drop out of the top-K, it would have been possible for a data point with a third label $l'' \notin \{l, l'\}$ to enter the top-K and potentially tip the voting balance in factor of this third label l'' (assuming there are enough instances of that label in the top-K already). This would **not contradict** the premise. However, since the lemma is defined for $|\mathcal{Y}| = 2$, this third case can actually never occur.

Finally, for $|\mathcal{Y}| = 2$, we can conclude that our proof by contradiction is complete.

□

Lemma 11. *Let $E_{l,\mathcal{D}}$ be the l-extreme world defined in Equation 3.4. Then, the K-NN algorithm trained over $E_{l,\mathcal{D}}$ will predict label l if and only if there exists a possible world $D \in \mathcal{I}_{\mathcal{D}}$ that will predict label l.*

Proof. We consider the following two cases:

1. $\mathcal{A}_{E_{l,\mathcal{D}}}(t) = l$: Since $E_{l,\mathcal{D}} \in \mathcal{I}_{\mathcal{D}}$, the successful prediction of label l represents a trivial proof of the existence of a possible world that predicts l.

2. $\mathcal{A}_{E_{l,\mathcal{D}}}(t) \neq l$: We can see that $E_{l,\mathcal{D}}$ is unique because it is constructed by taking from each candidate set $\mathcal{C}_i$ the minimal/maximal element, which itself is always unique (resulting from the problem setup laid out in Section 3.7.1). Consequently, the relation $R_{t,l}(D, E_{l,\mathcal{D}})$ holds for every $D \in \mathcal{I}_{\mathcal{D}}$. Given Lemma 14, we can say that if there exists any $D \in \mathcal{I}_{\mathcal{D}}$ that will predict l, then it is impossible for $E_{l,\mathcal{D}}$ to not predict l. Conversely, we can conclude that if $E_{l,\mathcal{D}}$ does not predict l, then no other possible world can predict l either.

□

Theorem 12. *The MM algorithm correctly answers* $Q1(\mathcal{D}, t, l)$.

Proof. The MM algorithm simply constructs the l'-extreme world $E_{l',\mathcal{D}}$ for each label $l' \in \mathcal{Y}$ and runs K-NN over it to check if it will predict l'. Given Lemma 15, we can conclude that this test is sufficient to check if there exists a possible world that can predict label l'. Then, the algorithm simply checks if l is the only label that can be predicted. We can trivially accept that this always gives the correct answer given that it is an exhaustive approach.

□

A.1.2 Correctness Proof of the Sort-Scan (SS) Algorithm

Theorem 13. *The SS algorithm correctly answers* $Q2(\mathcal{D}, t, l)$.

Proof. The SS algorithm aims to solve a counting problem by using a technique of partitioning a set and then computing the sizes of the relevant partitions. To prove its correctness we need to: (1) argue that the partitioning procedure is valid and produces disjoint subsets of the original set; and (2) argue that the size of the subset is computed correctly.

To prove the validity of the partitioning method, we start off by reviewing how a brute-force approach would answer the same query:

$$Q2(\mathcal{D}, t, l) = \sum_{D \in \mathcal{D}} \mathbb{I}[\mathcal{A}_D(t) = l]$$

When we partition the sum over all possible worlds into boundary sets $BSet(i, j; K)$ for each $i \in 1...N$ and $j \in 1...M$, we obtain the following expression:

$$Q2(\mathcal{D}, t, l) = \sum_{i \in [N]} \sum_{j \in [M]} \sum_{D \in BSet(i,j;K)} \mathbb{I}[\mathcal{A}_D(t) = l]$$

As we mentioned, a boundary set is the set of the possible worlds where $x_i = x_{i,j}$ and x_i is the K-th most similar data example to t. Since every possible world selects just one candidate per candidate set, for every $i \in \{1...N\}$, the possible world where $x_i = x_{i,j}$ is always different from the possible world where $x_i = x_{i,j'}$, for every

$j, j' \in \{1...M\}$ such that $j \neq j'$. Furthermore, every possible world induces a fixed ordering of data examples based on their similarity to t. Therefore, any possible worlds where x_i occupies the K-th position in that ordering is different from the possible world where it occupies any other position. Thus, we can conclude that all boundary sets $BSet(i, j, K)$ are distinct for all distinct i and j.

Given that we are dealing with the K-NN algorithm, since each possible world D induces a fixed set of top-K examples, consequently it induces a fixed top-K label tally γ^D. Since only one label tally of all the possible ones will be correct one, we can rewrite the inner sum as:

$$Q2(\mathcal{D}, t, l) = \sum_{i \in [N]} \sum_{j \in [M]} \sum_{D \in BSet(i,j;K)} \sum_{\gamma \in \Gamma} \mathbb{I}[\gamma^D = \gamma] \mathbb{I}[l = \arg\max \gamma]$$

Since in the above expression, the γ is independent from D, we can reorganize the sums as such:

$$Q2(\mathcal{D}, t, l) = \sum_{i \in [N]} \sum_{j \in [M]} \sum_{\gamma \in \Gamma} \mathbb{I}[l = \arg\max \gamma] \sum_{D \in BSet(i,j;K)} \mathbb{I}[\gamma^D = \gamma]$$

We can notice that the innermost sum is equivalent to the definition of a label tally support, which means we can replace it as such:

$$Q2(\mathcal{D}, t, l) = \sum_{i \in [N]} \sum_{j \in [M]} \sum_{\gamma \in \Gamma} \mathbb{I}[l = \arg\max(\gamma)] \cdot Support(i, j, \gamma)$$

Assuming that the label tally support $Support(i, j, \gamma)$ is computed correctly, as shown in Section 3.7.1, we can conclude that both the partitioning and the partition size computation problems are solved correctly, hence proving our original claim.

□

Lemma 14. *Let $D^{(1)}, D^{(2)} \in \mathcal{I}_{\mathcal{D}}$ be two possible worlds generated from an incomplete dataset $\mathcal{D}$. Given a test example $t \in \mathcal{X}$ and label $l \in \mathcal{Y}$ where $|\mathcal{Y}| = 2$, let $R_{t,l}$ be a partial ordering relation defined as such:*

$$R_{t,l}\left(D^{(1)}, D^{(2)}\right) := \bigwedge_{i=1}^{N} \left(y_i = l \ \wedge \ \kappa(x_i^{(1)}, t) \leq \kappa(x_i^{(2)}, t)\right) \vee \left(y_i \neq l \ \wedge \ \kappa(x_i^{(1)}, t) \geq \kappa(x_i^{(2)}, t)\right) \tag{A.2}$$

Then, the following relationship holds:

$$R_{t,l}\left(D^{(1)}, D^{(2)}\right) \implies \left(\left(\mathcal{A}_{D^{(1)}}(t) = l\right) \implies \left(\mathcal{A}_{D^{(2)}}(t) = l\right)\right)$$

Proof. We will prove this by contradiction. Consider the case when $\mathcal{A}_{D^{(1)}}(t) = l$ and $\mathcal{A}_{D^{(2)}}(t) \neq l$, that is, possible world $D^{(1)}$ predicts label l but possible world $D^{(2)}$ predicts some other label $l' \neq l$. That means that in the top-K induced by $D^{(2)}$ has to be at least one more data point with label l' than in the top-K induced by $D^{(1)}$. Is it possible for the premise to be true?

The similarities $\kappa(x_i^{(1)}, t)$ and $\kappa(x_i^{(2)}, t)$ cannot all be equal because that would represent equal possible worlds and that would trivially contradict with the premise since the labels predicted by equal possible worlds cannot differ. Therefore, at least one of the $i = 1, ..., N$ inequalities has to be strict. There, we distinguish three possible cases with respect to the class label y_i of the i-th example:

1. $y_i = l$: This means that the similarity of a data point coming from $D^{(2)}$ is higher than the one coming from $D^{(1)}$. However, this could only elevate that data point in the similarity-based ordering and could only increase the number of data points with label l in the top-K. Since $\mathcal{A}_{D^{(1)}}(t) = l$, the prediction of $D^{(2)}$ could not be different, which is **contradicts** the premise.

2. $y_i = l' \neq l$ and $|\mathcal{Y}| = 2$: We have a data point with a label different from l with lowered similarity, which means it can only drop in the ordering. This can not cause an increase of data points with label l' in the top-K, which again **contradicts** the premise.

3. $y_i = l' \neq l$ and $|\mathcal{Y}| \geq 2$: Here we again have a data point with label l' with lowered similarity. However, if that data point were to drop out of the top-K, it would have been possible for a data point with a third label $l'' \notin \{l, l'\}$

to enter the top-K and potentially tip the voting balance in factor of this third label l'' (assuming there are enough instances of that label in the top-K already). This would **not contradict** the premise. However, since the lemma is defined for $|\mathcal{Y}| = 2$, this third case can actually never occur.

Finally, for $|\mathcal{Y}| = 2$, we can conclude that our proof by contradiction is complete.

□

Lemma 15. *Let $E_{l,\mathcal{D}}$ be the l-extreme world defined in Equation 3.4. Then, the K-NN algorithm trained over $E_{l,\mathcal{D}}$ will predict label l if and only if there exists a possible world $D \in \mathcal{I}_{\mathcal{D}}$ that will predict label l.*

Proof. We consider the following two cases:

1. $\mathcal{A}_{E_{l,\mathcal{D}}}(t) = l$: Since $E_{l,\mathcal{D}} \in \mathcal{I}_{\mathcal{D}}$, the successful prediction of label l represents a trivial proof of the existence of a possible world that predicts l.

2. $\mathcal{A}_{E_{l,\mathcal{D}}}(t) \neq l$: We can see that $E_{l,\mathcal{D}}$ is unique because it is constructed by taking from each candidate set $\mathcal{C}_i$ the minimal/maximal element, which itself is always unique (resulting from the problem setup laid out in Section 3.7.1). Consequently, the relation $R_{t,l}(D, E_{l,\mathcal{D}})$ holds for every $D \in \mathcal{I}_{\mathcal{D}}$. Given Lemma 14, we can say that if there exists any $D \in \mathcal{I}_{\mathcal{D}}$ that will predict l, then it is impossible for $E_{l,\mathcal{D}}$ to not predict l. Conversely, we can conclude that if $E_{l,\mathcal{D}}$ does not predict l, then no other possible world can predict l either.

□

Theorem 16. *The MM algorithm correctly answers $Q1(\mathcal{D}, t, l)$.*

Proof. The MM algorithm simply constructs the l'-extreme world $E_{l',\mathcal{D}}$ for each label $l' \in \mathcal{Y}$ and runs K-NN over it to check if it will predict l'. Given Lemma 15, we can conclude that this test is sufficient to check if there exists a possible world that can predict label l'. Then, the algorithm simply checks if l is the only label that can be predicted. We can trivially accept that this always gives the correct answer given that it is an exhaustive approach.

□

A.2 Theoretical Guarantee for the CPClean Algorithm

We begin with the following supplementary results.

Lemma 17. *Let D_{Opt} be the optimal set of size t described in Corollary 2. Denote the set of cleaned training data instances of size T by D_π. For $\mathbf{c}_{Opt_i} \in D_{Opt}, i = \{1, 2, \ldots, t\}$, we have*

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{val}); \mathbf{c}_{Opt_j} | \mathbf{c}_{Opt_{j-1}}, \ldots, \mathbf{c}_{Opt_1}, D_\pi) \\ &\leq \theta \min\{\log|\mathcal{Y}|, \log m\} I(\mathcal{A}_{\mathbf{D}}(D_{val}); \mathbf{c}_{\pi_{t+1}} | D_\pi). \end{aligned} \tag{A.3}$$

where $\theta = \frac{1}{\max_{\mathbf{v} \in D_{train}} I(\mathcal{A}_{\mathbf{D}}(D_{val}); \mathbf{v})}$.

Proof. We first start with the following inequality:

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | \mathbf{c}_{\text{Opt}_{j-1}}, ..., \mathbf{c}_{\text{Opt}_1}, D_\pi) \\ &= \frac{I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | \mathbf{c}_{\text{Opt}_{j-1}}, ..., \mathbf{c}_{\text{Opt}_1}, D_\pi)}{\max_{\mathbf{v}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v} | D_\pi)} \max_{\mathbf{v}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v} | D_\pi) \\ &\leq \frac{\min\{\log|\mathcal{Y}|, \log m\}}{\max_{\mathbf{v}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v} | D_\pi)} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | D_\pi) \end{aligned} \tag{A.4}$$

where the last inequality follows from

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | \mathbf{c}_{\text{Opt}_{j-1}}, ..., \mathbf{c}_{\text{Opt}_1}, D_\pi) \\ &\quad \leq \min\{\mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}})), \mathcal{H}(\mathbf{c}_{\text{Opt}_j})\} \\ &\quad \leq \min\{\log|\mathcal{Y}|, \log m\}. \end{aligned}$$

Let $\mathbf{v}^* = \arg\max_{\mathbf{v} \in D_{\text{train}}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v})$. We have

$$\begin{aligned} \max_{\mathbf{c}_{\text{Opt}_j}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | D_\pi) &\geq I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}^* | D_\pi) \\ &\geq I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}^*) \end{aligned} \tag{A.5}$$

where the last inequality follows from the independence of $\mathbf{v}_i$'s in D_{train}. That is,

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}), D_\pi; \mathbf{v}^*) \\ &\quad = I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}^*) + I(D_\pi^{[l]}; \mathbf{v}^* | \mathcal{A}_{\mathbf{D}}(D_{\text{val}})) \\ &\quad = I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}^* | D_\pi) + I(D_\pi^{[l]}; \mathbf{v}^*). \end{aligned}$$

The independence of $\mathbf{v}_i$'s implies that $I(D_\pi; \mathbf{v}^*) = 0$. Hence (A.5) follows.

In the next step, we let $\theta = \frac{1}{I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}});\mathbf{v}^*)}$ where

$$\mathbf{v}^* = \arg\max_{\mathbf{v}\in D_{\text{train}}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}). \tag{A.6}$$

Combining (A.4) and (A.5), we further have

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | \mathbf{c}_{\text{Opt}_{j-1}}, ..., \mathbf{c}_{\text{Opt}_1}, D_\pi) \\ &\quad \leq \theta \min\{\log|\mathcal{Y}|, \log m\} \max_{\mathbf{v}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}|D_\pi). \end{aligned} \tag{A.7}$$

We remind that our update rule is simply

$$\mathbf{c}_{\pi_{T+1}} \coloneqq \{\arg\max_{\mathbf{v}} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{v}|D_\pi)\}.$$

Inserting this into (A.7) proves the Lemma. □

We now move to the proof of Corollary 2.

Proof. We start by noting that

$$\mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}})|D_{\text{Opt}}) \geq \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}})|D_{\text{Opt}}, D_\pi).$$

We also note

$$\begin{aligned} \mathcal{I}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}) &= \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}})) - \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}})|D_{\text{Opt}}) \\ \mathcal{I}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}, D_\pi) &= \mathcal{H}(\mathcal{A}_{\mathbf{D}}(D_{\text{val}})) - \mathcal{H}(h\mathcal{A}_{\mathbf{D}}(D_{\text{val}})|D_{\text{Opt}}, D_\pi) \end{aligned}$$

Hence

$$I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}) \leq I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}, D_\pi). \tag{A.8}$$

We further proceed with (A.8) as follows.

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}) \\ &\quad \leq I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}, D_\pi) \\ &\quad = I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}, D_\pi) - I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_\pi) \\ &\qquad + I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_\pi) \\ &\quad = I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}}|D_\pi) + I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_\pi) \\ &\quad = \sum_{j=1}^{t} I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); \mathbf{c}_{\text{Opt}_j} | \mathbf{c}_{\text{Opt}_{j-1}}, \ldots, \mathbf{c}_{\text{Opt}_1}, D_\pi) \\ &\qquad + I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_\pi) \end{aligned} \tag{A.9}$$

where the last equality follows from the telescopic sum with $\mathbf{c}_{\mathrm{Opt}_j} \in D_{\mathrm{Opt}}$.

Using Lemma 17, (A.9) can be followed by

$$\begin{aligned}
&I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); D_{\mathrm{Opt}}) - I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); D_{\pi}) \\
&\quad \leq \sum_j I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); \mathbf{c}_{\mathrm{Opt}_j} | \mathbf{c}_{\mathrm{Opt}_{j-1}}, \ldots, \mathbf{c}_{\mathrm{Opt}_1}, D_{\pi}) \\
&\quad \leq \sum_j \theta \min\{\log|\mathcal{Y}|, \log m\} \max_{\mathbf{c}_{\mathrm{Opt}_j}} I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); \mathbf{c}_{\mathrm{Opt}_j} | D_{\pi}) \\
&\quad \leq \theta \min\{\log|\mathcal{Y}|, \log m\} \sum_j I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); \mathbf{c}_{\pi_{T+1}} | D_{\pi}) \\
&\quad \leq t\theta \min\{\log|\mathcal{Y}|, \log m\} I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); \mathbf{c}_{\pi_{T+1}} | D_{\pi}) \\
&\quad \leq t\theta \min\{\log|\mathcal{Y}|, \log m\} \big(I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); \mathbf{c}_{\pi_{T+1}} \cup D_{\pi}) \\
&\qquad - I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); \mathbf{c}_{\pi_{T+1}} \cup D_{\pi}) \big).
\end{aligned} \tag{A.10}$$

We further let $\Delta_T = I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); D_{\mathrm{Opt}}) - I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); D_{\pi})$. (A.10) becomes:

$$\Delta_T \leq t\theta \min\{\log|\mathcal{Y}|, \log m\}(\Delta_T - \Delta_{T+1}). \tag{A.11}$$

Arranging the terms of (A.11), we have

$$t\theta \min\{\log|\mathcal{Y}|, \log m\}\Delta_{T+1} \leq (t\theta \min\{\log|\mathcal{Y}|, \log m\} - 1)\Delta_T$$

and hence

$$\begin{aligned}
\Delta_{T+1} &\leq \frac{t\theta \min\{\log|\mathcal{Y}|, \log m\} - 1}{t\theta \min\{\log|\mathcal{Y}|, \log m\}} \Delta_T \\
&\leq \cdots \\
&\leq \left(\frac{t \min\{\log|\mathcal{Y}|, \log m\}\theta - 1}{t\theta \min\{\log|\mathcal{Y}|, \log m\}} \right)^T \Delta_0.
\end{aligned} \tag{A.12}$$

Noting

$$\Big(\frac{t\theta \min\{\log|\mathcal{Y}|, \log m\} - 1}{t\theta \min\{\log|\mathcal{Y}|, \log m\}} \Big)^l \leq \exp(-l/t\theta \min\{\log|\mathcal{Y}|, \log m\})$$

we have

$$\begin{aligned}
\Delta_{T+1} &\leq \exp(-T/t\theta \min\{\log|\mathcal{Y}|, \log m\})\Delta_0 \\
&= \exp(-T/t\theta \min\{\log|\mathcal{Y}|, \log m\}\gamma) I(\mathcal{A}_{\mathbf{D}}(D_{\mathrm{val}}); D_{\mathrm{Opt}}).
\end{aligned}$$

By the definition of Δ_T, we therefore have

$$\begin{aligned} &I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_\pi) \\ &\quad \geq I(\mathcal{A}_{\mathbf{D}}(D_{\text{val}}); D_{\text{Opt}})(1 - e^{-\frac{T}{t\theta \min\{\log|\mathcal{Y}|, \log m\}}}). \end{aligned} \tag{A.13}$$

which proves the Corollary 2. □

A.3 Correctness Proofs for Data Provenance and ADD's

A.3.1 Correctness Proof of ADD Model Counting (Lemma Lemma 18)

Proof. We will prove this by induction on the structure of the recursion.

(Base case.) Based on Equation 4.2, when $n = \boxdot$ we get $\text{eval}_v(n) = 0$ for all v. Furthermore, when $n = \boxdot$, the set $\mathcal{V}_{\mathcal{X}}[x_{>\pi(x(n))} = 0]$ contains only one value assignment with all variables set to zero. Hence, the model count will equal to 1 only for $e = 0$ and it will be 0 otherwise, which is reflected in the base cases of Equation 4.4.

(Inductive step.) Because our ADD is ordered and full, both $c_L(n)$ and $c_H(n)$ are associated with the same variable, which is the predecessor of $x(n)$ in the permutation π. Based on this and the induction hypothesis, we can assume that

$$\begin{aligned} \text{count}_{e-a_L(n)}(c_L(n)) &= \left|\left\{v \in \mathcal{V}_{\mathcal{X}[\leq x(c_L(n))]} \mid \text{eval}_v(c_L(n)) = e - a_L(n)\right\}\right| \\ \text{count}_{e-a_H(n)}(c_H(n)) &= \left|\left\{v \in \mathcal{V}_{\mathcal{X}[\leq x(c_H(n))]} \mid \text{eval}_v(c_H(n)) = e - a_H(n)\right\}\right| \end{aligned} \tag{A.14}$$

We would like to compute $\text{count}_e(n)$ as defined in Equation 4.3. It computes the size of a set defined over possible value assignments to variables in $\mathcal{X}[\leq x(n)]$. The set of value assignments can be partitioned into two distinct sets: one where $x(n) \leftarrow 0$ and one where $x(n) \leftarrow 1$. We thus obtain the following expression:

$$\begin{aligned} \text{count}_e(n) := &\left|\left\{v \in \mathcal{V}_{\mathcal{X}[\leq x(n)]}[x(n) \leftarrow 0] \mid \text{eval}_v(n) = e\right\}\right| \\ &+ \left|\left\{v \in \mathcal{V}_{\mathcal{X}[\leq x(n)]}[x(n) \leftarrow 1] \mid \text{eval}_v(n) = e\right\}\right| \end{aligned} \tag{A.15}$$

Based on Equation 4.2, we can transform the $\text{eval}_v(n)$ expressions as such:

$$\begin{aligned}\text{count}_e(n) := &\left|\left\{v \in \mathcal{V}_{\mathcal{X}[\leq x(c_L(n))]} \mid a_L(n) + \text{eval}_v(c_L(n)) = e\right\}\right| \\ &+ \left|\left\{v \in \mathcal{V}_{\mathcal{X}[\leq x(c_L(n))]} \mid a_H(n) + \text{eval}_v(c_H(n)) = e\right\}\right|\end{aligned} \tag{A.16}$$

Finally, we can notice that the set size expressions are equivalent to those in Equation A.18. Therefore, we can obtain the following expression:

$$\text{count}_e(n) := \text{count}_{e-a_L(n)}(c_L(n)) + \text{count}_{e-a_H(n)}(c_H(n)) \tag{A.17}$$

which is exactly the recursive step in Equation 4.4. This concludes our inductive proof and we move onto proving the complexity bound.

(Complexity.) This is trivially proven by observing that since count has two arguments, we can maintain a table of results obtained for each $n \in \mathcal{N}$ and $e \in \mathcal{E}$. Therefore, we know that we will never need to perform more than $O(|\mathcal{N}| \cdot |\mathcal{E}|)$ invocations of $\text{count}_e(n)$.

□

A.4 Correctness Proofs for Shapley Computation Algorithms

A.4.1 Proof of Theorem 4

MODEL COUNTING FOR ADD'S. We start off by proving that Equation 4.4 correctly performs model counting.

Lemma 18. *For a given node $n \in \mathcal{N}$ of an ADD and a given value $e \in \mathcal{E}$, Equation 4.4 correctly computes $\text{count}_e(n)$ which returns the number of assignments $v \in \mathcal{V}_A$ such that $\text{eval}_v(n) = e$. Furthermore, when computing $\text{count}_e(n)$ for any $n \in \mathcal{N}$, the number of computational steps is bounded by $O(|\mathcal{N}| \cdot |\mathcal{E}|)$.*

Proof. We will prove this by induction on the structure of the recursion.

(Base case.) Based on Equation 4.2, when $n = \boxdot$ we get $\text{eval}_v(n) = 0$ for all v. Furthermore, when $n = \boxdot$, the set $\mathcal{V}_A[a_{>\pi(a(n))} = 0]$ contains only one value assignment with all variables set to zero. Hence, the model count will equal to 1

only for $e = 0$ and it will be 0 otherwise, which is reflected in the base cases of Equation 4.4.

(Inductive step.) Because our ADD is ordered and full, both $c_L(n)$ and $c_H(n)$ are associated with the same variable, which is the predecessor of $a(n)$ in the permutation π. Based on this and the induction hypothesis, we can assume that

$$\begin{aligned}\text{count}_{e-w_L(n)}(c_L(n)) &= \left|\left\{v \in \mathcal{V}_{A[\leq a(c_L(n))]} \mid \text{eval}_v(c_L(n)) = e - w_L(n)\right\}\right| \\ \text{count}_{e-w_H(n)}(c_H(n)) &= \left|\left\{v \in \mathcal{V}_{A[\leq a(c_H(n))]} \mid \text{eval}_v(c_H(n)) = e - w_H(n)\right\}\right|\end{aligned} \tag{A.18}$$

We would like to compute $\text{count}_e(n)$ as defined in Equation 4.3. It computes the size of a set defined over possible value assignments to variables in $A[\leq a(n)]$. The set of value assignments can be partitioned into two distinct sets: one where $a(n) \leftarrow 0$ and one where $a(n) \leftarrow 1$. We thus obtain the following expression:

$$\begin{aligned}\text{count}_e(n) := &\left|\left\{v \in \mathcal{V}_{A[\leq a(n)]}\left[a(n) \leftarrow 0\right] \mid \text{eval}_v(n) = e\right\}\right| \\ &+ \left|\left\{v \in \mathcal{V}_{A[\leq a(n)]}\left[a(n) \leftarrow 1\right] \mid \text{eval}_v(n) = e\right\}\right|\end{aligned} \tag{A.19}$$

Based on Equation 4.2, we can transform the $\text{eval}_v(n)$ expressions as such:

$$\begin{aligned}\text{count}_e(n) := &\left|\left\{v \in \mathcal{V}_{A[\leq a(c_L(n))]} \mid w_L(n) + \text{eval}_v(c_L(n)) = e\right\}\right| \\ &+ \left|\left\{v \in \mathcal{V}_{A[\leq a(c_L(n))]} \mid w_H(n) + \text{eval}_v(c_H(n)) = e\right\}\right|\end{aligned} \tag{A.20}$$

Finally, we can notice that the set size expressions are equivalent to those in Equation A.18. Therefore, we can obtain the following expression:

$$\text{count}_e(n) := \text{count}_{e-w_L(n)}(c_L(n)) + \text{count}_{e-w_H(n)}(c_H(n)) \tag{A.21}$$

which is exactly the recursive step in Equation 4.4. This concludes our inductive proof and we move onto proving the complexity bound.

(Complexity.) This is trivially proven by observing that since count has two arguments, we can maintain a table of results obtained for each $n \in \mathcal{N}$ and $e \in \mathcal{E}$. Therefore, we know that we will never need to perform more than $O(|\mathcal{N}| \cdot |\mathcal{E}|)$ invocations of $\text{count}_e(n)$.

□

ADD CONSTRUCTION. Next, we prove that the size of an ADD resulting from *diagram summation* as defined in Section 4.2.2 is linear in the number of variables.

The size of the diagram resulting from a sum of two diagrams with node sets $\mathcal{N}_1$ and $\mathcal{N}_2$ can be loosely bounded by $O(|\mathcal{N}_1| \cdot |\mathcal{N}_2|)$ assuming that its nodes come from a combination of every possible pair of operand nodes. However, given the much more narrow assumptions we made in the definition of the node sum operator, we can make this bound considerably tighter. For this we define the *diameter* of an ADD as the maximum number of nodes associated with any single variable. Formally we can write:

$$\operatorname{diam}(\mathcal{N}) := \max_{a_i \in A} \big|\{n \in \mathcal{N} : a(n) = a_i\}\big| \tag{A.22}$$

We can immediately notice that the size of any ADD with set of nodes $\mathcal{N}$ and variables A is bounded by $O(|A| \cdot \operatorname{diam}(\mathcal{N}))$. We can use this fact to prove a tighter bound on the size of an ADD resulting from a sum operation:

Lemma 19. *Given two full ordered ADD's with nodes $\mathcal{N}_1$ and $\mathcal{N}_2$, noth defined over the set of variables A, the number of nodes in $\mathcal{N}_1 + \mathcal{N}_2$ is bounded by $O(|A| \cdot \operatorname{diam}(\mathcal{N}_1) \cdot \operatorname{diam}(\mathcal{N}_2))$.*

Proof. It is sufficient to show that $\operatorname{diam}(\mathcal{N}_1 + \mathcal{N}_2) = O(\operatorname{diam}(\mathcal{N}_1) \cdot \operatorname{diam}(\mathcal{N}_2))$. This is a direct consequence of the fact that for full ordered ADD's the node sum operator is defined only for nodes associated with the same variable. Since the only way to produce new nodes is by merging one node in $\mathcal{N}_1$ with one node in $\mathcal{N}_2$, and given that we can merge nodes associated with the same variable, the number of nodes associated with the same variable in the resulting ADD equals the product of the corresponding number of nodes in the constituent ADD's. Since the diameter is simply the upper bound of the number of nodes associated with any single variable, the same upper bound in the resulting ADD cannot be larger than the product of the upper bounds of constituent nodes. □

COMPUTING THE ORACLE USING ADD'S. Finally, we prove the correctness of Theorem 4.

Lemma 20. *Given an Additive Decision diagram with root node $n_{t,t'}$ that computes the Boolean function $\phi_{t,t'}(v)$ as defined in Equation 4.18, the counting oracle $\omega_{t,t'}(\alpha, \gamma, \gamma')$ defined in Equation 4.16 can be computed as:*

$$\omega_{t,t'}(\alpha, \gamma, \gamma') := \text{count}_{(\alpha,\gamma,\gamma')}(n_{t,t'}) \tag{A.23}$$

Proof. Let us define $\mathcal{D}[\geq_\sigma t] \subseteq \mathcal{D}$ as a set of tuples with similarity higher or equal than that of t, formally $\mathcal{D}[\geq_\sigma t] := \{t' \in \mathcal{D} : \sigma(t') \geq \sigma(t)\}$. Similarly to $\mathcal{D}$, the semantics of $\mathcal{D}[\geq_\sigma t]$ is also that of a set of possible candidate sets. Given a value assignment v, we can obtain $\mathcal{D}[\geq_\sigma t][v]$ from $\mathcal{D}[v]$. For convenience, we also define $\mathcal{D}[\geq_\sigma t][y]$ as a subset of $\mathcal{D}[\geq_\sigma t]$ with only tuples that have label y. Given these definitions, we can define several equivalences. First, for top_K we have:

$$\Big(t = \text{top}_K \mathcal{D}[v]\Big) \iff \Big(t \in \mathcal{D}[v] \wedge \big|\mathcal{D}[\geq_\sigma t][v]\big| = K\Big) \tag{A.24}$$

In other words, for t to be the tuple with the K-th highest similarity in $\mathcal{D}[v]$, it needs to be a member of $\mathcal{D}[v]$ and the number of tuples with similarity greater or equal to t has to be exactly K. Similarly, we can define the equivalence for tally_t:

$$\Big(\gamma = \text{tally}_t \mathcal{D}[v]\Big) \iff \Big(\forall y \in \mathcal{Y}, \gamma_y = \big|\mathcal{D}[\geq_\sigma t][y][v]\big|\Big) \tag{A.25}$$

This is simply an expression that partitions the set $\mathcal{D}[\geq_\sigma t][v]$ based on y and tallies them up. The next step is to define an equivalence for $(t = \text{top}_K \mathcal{D}[v]) \wedge (\gamma = \text{tally}_t \mathcal{D}[v])$. We can notice that since $|\gamma| = K$, if we have $(\forall y \in \mathcal{Y}, \gamma_y = |\mathcal{D}[\geq_\sigma t][y][v]|)$ then we can conclude that $(|\mathcal{D}[\geq_\sigma t][v]| = K)$ is redundant. Hence, we can obtain:

$$\begin{aligned} &\Big(t = \text{top}_K \mathcal{D}[v]\Big) \wedge \Big(\gamma = \text{tally}_t \mathcal{D}[v]\Big) \\ \iff &\Big(t \in \mathcal{D}[v]\Big) \wedge \Big(\forall y \in \mathcal{Y}, \gamma_y = \big|\mathcal{D}[\geq_\sigma t][y][v]\big|\Big) \end{aligned} \tag{A.26}$$

According to Equation A.25, we can reformulate the right-hand side of the above equivalence as:

$$\Big(t = \text{top}_K \mathcal{D}[v]\Big) \wedge \Big(\gamma = \text{tally}_t \mathcal{D}[v]\Big) \iff \Big(t \in \mathcal{D}[v]\Big) \wedge \Big(\gamma = \text{tally}_t \mathcal{D}[v]\Big) \tag{A.27}$$

We can construct a similar expression for t' and $v[a_i = 1]$ so we cover four out of five predicates in Equation 4.16. The remaining one is simply the support of the

value assignment v which we will leave intact. This leads us with the following equation for the counting oracle:

$$\begin{aligned}\omega_{t,t'}(\alpha,\gamma,\gamma') := \sum_{v\in\mathcal{V}_A[a_i\leftarrow 0]} & \mathbb{1}\{\alpha = |\text{supp}(v)|\} \\ & \mathbb{1}\{t \in f(\mathcal{D}[v])\}\mathbb{1}\{t' \in f(\mathcal{D}[v[a_i \leftarrow 1]])\} \\ & \mathbb{1}\{\gamma = \text{tally}_t\mathcal{D}[v]\}\mathbb{1}\{\gamma = \text{tally}_t\mathcal{D}[v[a_i \leftarrow 1]]\}\end{aligned} \tag{A.28}$$

We can use the Boolean function $\phi_{t,t'}(v)$ in Equation 4.18 to simplify the above equation. Notice that the conditions $t \in f(\mathcal{D}[v])$ and $t' \in f(\mathcal{D}[v[a_i \leftarrow 1]])$ are embedded in the definition of $\phi_{t,t'}(v)$ which will return ∞ if those conditions are not met. When the conditions are met, $\phi_{t,t'}(v)$ returns exactly the same triple (α,γ,γ'). Therefore it is safe to replace the five indicator functions in the above formula with a single one as such:

$$\omega_{t,t'}(\alpha,\gamma,\gamma') := \sum_{v\in\mathcal{V}_A[a_i\leftarrow 0]} \mathbb{1}\{(\alpha,\gamma,\gamma') = \phi_{t,t'}(v)\} \tag{A.29}$$

Given our assumption that $\phi_{t,t'}(v)$ can be represented by an ADD with a root node $n_{t,t'}$, the above formula is exactly the model counting operation:

$$\omega_{t,t'}(\alpha,\gamma,\gamma') := \text{count}_{(\alpha,\gamma,\gamma')}(n_{t,t'}) \tag{A.30}$$

□

Theorem 21. *If we can represent the Boolean function $\phi_{t,t'}(v)$ defined in Equation 4.18 with an Additive Decision Diagram of size polynomial in $|\mathcal{D}|$ and $|f(\mathcal{D})|$, then we can compute the counting oracle $\omega_{t,t'}$ in time polynomial in $|\mathcal{D}|$ and $|f(\mathcal{D})|$.*

Proof. This theorem follows from the two previously proved lemmas: Lemma 18 and Lemma 20. Namely, as a result of Lemma 20 we claim that model counting of the Boolean function $\phi_{t,t'}(v)$ is equivalent to computing the oracle result. On top of that, as a result of Lemma 18 we know that we can perform model counting in time linear in the size of the decision diagram. Hence, if our function $\phi_{t,t'}(v)$ can be represented with a decision diagram of size polynomial in the size of data, then we can conclude that computing the oracle result can be done in time polynomial in the size of data. □

A.4.2 Proof of Corollary 5

Proof. This follows from the observation that in Algorithm 7, each connected component A_C will be made up from one variable corresponding to the dimension table and one or more variables corresponding to the fact table. Since the fact table variables will be categorized as "leaf variables", the expression $A_C \setminus A_L$ in Line 13 will contain only a single element – the dimension table variable. Consequently, the ADD tree in $\mathcal{N}'$ will contain a single node. On the other side, the $A_C \cap A_L$ expression will contain all fact table variables associated with that single dimension table variable. That chain will be added to the ADD tree two times for two outgoing branches of the single tree node. Hence, the ADD segment will be made up of two fact table variable chains stemming from a single dimension table variable node. There will be $O(|\mathcal{D}_D|)$ partitions in total. Given that the fact table variables are partitioned, the cumulative size of their chains will be $O(|\mathcal{D}_F|)$. Therefore, the total size of the ADD with all partitions joined together is bounded by $O(|\mathcal{D}_D| + |\mathcal{D}_F|) = O(N)$.

Given fact and combining it with Theorem 4 we know that the counting oracle can be computed in time $O(N)$ time. Finally, given Theorem 3 and the structure of Equation 4.17 we can observe that the counting oracle is invoked $O(N^3)$ times. As a result, we can conclude that the total complexity of computing the Shapley value is $O(N^4)$. □

A.4.3 Proof of Corollary 6

Proof. The key observation here is that, since all provenance polynomials contain only a single variable, there is no interdependency between them, which means that the connected components returned in Line 12 of Algorithm 7 will each contain a single variable. Therefore, the size of the resulting ADD will be $O(N)$. Consequently, similar to the proof of the previous corollary, the counting oracle can be computed in time $O(N)$ time. In this case, the size of the output dataset is $O(M)$ which means that Equation 4.17 willinvoke the oracle $O(M^2N)$ times. Therefore, the total time complexity of computing the Shapley value will be $O(M^2N^2)$. □

A.4.4 Proof of Corollary 7

Proof. There are two arguments we need to make which will result in the reduction of complexity compared to fork pipelines. The first argument is that, given that each variable can appear in the provenance polynomial of at most one tuple, having its value set to 1 can result in either zero or one tuple contributing to the top-K tally. It will be one if that tuple is more similar than the boundary tuple t and it will be zero if it is less similar. Consequently, our ADD will have a chain structure with high-child increments being either 0 or 1. If we partition the ADD into two chains, one with all increments 1 and another with all increments 0, then we end up with two uniform ADD's. As shown in Equation 4.5, model counting of uniform ADD's can be achieved in constant time. The only difference here is that, since we have to account for the support size each model, computing the oracle $\omega_{t,t'}(\alpha, \gamma, \gamma')$ for a given α will require us to account for different possible ways to split α across the two ADD's. However, since the tuple t needs to be the boundary tuple, which means it is the K-th most similar, there need to be exactly $K-1$ variables from the ADD with increments 1 that can be set to 1. This gives us a single possible distribution of α across two ADD's. Hence, the oracle can be computed in constant time.

As for the second argument, we need to make a simple observation. For map pipelines, given a boundary tuple t and a tally vector γ corresponding to the variable a_i being assigned the value 0, we know that setting this variable to 1 can introduce at most one tuple to the top-K. That could only be the single tuple associated with a_i. If this tuple has a lower similarity score than t, there will be no change in the top-K. On the other side, if it has a higher similarity, then it will become part of the top-K and it will evict exactly t from it. Hence, there is a unique tally vector γ' resulting from a_i being assigned the value 1. This means that instead of computing the counting oracle $\omega_{t,t'}(\alpha, \gamma, \gamma')$, we can compute the oracle $\omega_t(\alpha, \gamma)$. This means that, in Equation 4.17 we can eliminate the iteration over t' which saves us an order of $O(N)$ in complexity.

As a result, Equation 4.17 will make $O(N^2)$ invocations to the oracle which can be computed in constant time. Hence, the final complexity of computing the Shapley value will be $O(N^2)$. □

A.4.5 Proof of Corollary 8

Proof. We start off by plugging in the oracle definition from Equation 4.22 into the Shapley value computation Equation 4.17:

$$\varphi_i = \frac{1}{N} \sum_{t,t' \in f(\mathcal{D})} \sum_{\alpha=1}^{N} \binom{N-1}{\alpha}^{-1} \sum_{\gamma,\gamma' \in \Gamma} u_\Delta(\gamma,\gamma') \binom{|\{t'' \in \mathcal{D} \ : \ \sigma(t'') < \sigma(t)\}|}{\alpha} \mathbb{1}\{p(t') = a_i\} \mathbb{1}\{\gamma = \Gamma_{y(t)}\} \mathbb{1}\{\gamma' = \Gamma_{y(t')}\} \tag{A.31}$$

As we can see, the oracle imposes hard constraints on the tuple t' and tally vectors γ and γ'. We will replace the tally vectors with their respective constants and the tuple t' we will denote as t_i because it is the only tuple associated with a_i. Because of this, we can remove the sums that iterate over them:

$$\varphi_i = \frac{1}{N} \sum_{t \in f(\mathcal{D})} \sum_{\alpha=1}^{N} \binom{N-1}{\alpha}^{-1} u_\Delta(\Gamma_{y(t)}, \Gamma_{y(t_i)}) \binom{|\{t'' \in \mathcal{D} \ : \ \sigma(t'') < \sigma(t)\}|}{\alpha} \tag{A.32}$$

We could significantly simplify this equation by assuming the tuples in $f(\mathcal{D})$ are sorted by decreasing similarity. We then obtain:

$$\varphi_i = \frac{1}{N} \sum_{j=1}^{N} \sum_{\alpha=1}^{N} \binom{N-1}{\alpha}^{-1} u_\Delta(\Gamma_{y(t)}, \Gamma_{y(t_i)}) \binom{N-j}{\alpha} \tag{A.33}$$

We shuffle the sums a little by multiplying $\frac{1}{N}$ with $\binom{N-1}{\alpha}^{-1}$ and expanding the u_Δ function according to its definition. We also alter the limit of the innermost sum because $\alpha \leq N - j$. Thus, we obtain:

$$\varphi_i = \sum_{j=1}^{N} \Big(\mathbb{1}\{y(t_i) = y(t_v)\} - \mathbb{1}\{y(t_j) = y(t_v)\} \Big) \sum_{\alpha=1}^{N-j} \binom{N}{\alpha}^{-1} \binom{N-j}{\alpha} \tag{A.34}$$

The inntermost sum in the above equation can be simplified by applying the so-called Hockey-stick identity [130]. Specifically, $\binom{N}{\alpha}^{-1}\binom{N-j}{\alpha}$ becomes $\binom{N}{j}^{-1}\binom{N-\alpha}{j}$. Then, $\sum_{\alpha=1}^{N-j} \binom{N}{j}^{-1}\binom{N-\alpha}{j}$ becomes $\binom{N}{j}^{-1}\binom{N}{j+1}$. Finally, we obtain the following formula:

$$\varphi_i = \sum_{j=1}^{N} \Big(\mathbb{1}\{y(t_i) = y(t_v)\} - \mathbb{1}\{y(t_j) = y(t_v)\} \Big) \binom{N-j}{j+1} \tag{A.35}$$

As we can see, the above formula can be computed in $O(N)$ iterations. Therefore, given that we still need to sort the dataset beforehand, the ovarall complexity of the entire Shapley value amounts to $O(N \log N)$. □

A.4.6 Proof of Corollary 9

Proof. We will prove this by reducing the problem of Shapley value computation in fork pipelines to the one of computing it for map pipelines. Let us have two tuples $t_{j,1}, t_{j,2} \in f(\mathcal{D})$, both associated with some variable $a_j \in A$. That means that $p(t_{j,1}) = p(t_{j,2})$. If we examine Equation 4.20, we notice that it will surely evaluate to false if either $\sigma(t_{j,1}) > \sigma(t)$ or $\sigma(t_{j,2}) > \sigma(t)$. The same observation holds for Equation 4.21.

Without loss of generality, assume $\sigma(t_{j,1}) > \sigma(t_{j,2})$. Then, $\sigma(t_{j,1}) > \sigma(t)$ implies $\sigma(t_{j,2}) > \sigma(t)$. As a result, we only ever need to check the former condition without paying attention to the latter. The outcome of this is that for all sets of tuples associated with the same variable, it is safe to ignore all of them except the one with the highest similarity score, and we will nevertheless obtain the same oracle result. Since we transformed the problem to the one where for each variable we have to consider only a single associated tuple, then we have effectively reduced the problem to the one of computing Shapley value for map pipelines. Consequently, we can apply the same algorithm and will end up with the same time complexity. □

CURRICULUM VITAE

Education

2018 – 2022	PhD candidate; focus on data management for machine learning *ETH Zurich*, Switzerland
2014 – 2017	Master of science in Computer Science; specialization in signals, images and interfaces *EPFL Lausanne*, Switzerland
2008 – 2012	Bachelor of science in Software Engineering *ETF Belgrade*, Serbia

Employment

2019	Research intern; Gray Systems Lab *Microsoft*, Redmond, USA
2018	Research intern; Oracle Labs *Oracle*, San Francisco Bay Area, USA
2016 - 2017	Research and Development Intern *Logitech*, Lausanne, Switzerland
2012 - 2014	Software Design Engineer; Microsoft Development Center Serbia *Microsoft*, Belgrade, Serbia