UNIVERSITY OF CALIFORNIA SAN DIEGO

Simplifying Data Preparation for Machine Learning on Tabular Data

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Vraj Shah

Committee in charge:

Professor Arun Kumar, Chair
Professor Alin Deutsch
Professor Philip Guo
Professor Lawrence Saul
Professor Jingbo Shang

2022

The dissertation of Vraj Shah is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

To my parents, Paragbhai and Mayuriben

For their love, sacrifices, and support all around.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I owe a great debt of gratitude to my advisor, Professor Arun Kumar for his constant support and guidance throughout my Ph.D. I am extremely thankful to him for mentoring and supporting the different "first of its kind" projects and for his unwavering support throughout, especially during the many paper rejections. Observing Professor Arun Kumar's efforts towards making a positive impact in the community through research contributions and also voicing awareness about inclusion has been truly inspiring for me. I am truly blessed to have him as my advisor and I hope I can give back similarly.

I would also like to thank all my thesis committee members, Professor Alin Deutsch, Professor Philip Guo, Professor Lawrence Saul, and Professor Jingbo Shang, for their valuable insights, comments, and feedback that have helped me improve my conference papers and research talks. I thank Professor Alin Deutsch for his kind words of confidence during the very initial stage of the SortingHat project that encouraged me to push ahead. I thank Professor Philip Guo for his valuable advice on grad school through several web blogs and videos that have helped me navigate this journey. I thank Professor Lawrence Saul for his collaboration on the SpeakQL project. I learned a lot from him through all the insightful discussions. I thank Professor Jingbo Shang for the collaboration on the AIMaker project that helped me learn about developing a large-scale project and how to effectively manage it. I am grateful to all my co-authors for helping me during different projects: Lawrence Saul, Xiaojin Zhu, Side Li, Jonathan Lacanlale, Dharmil Chandarana, Premanand Kumar, Thomas Parashos, and Kevin Yang. I thank Neoklis Polyzotis, Google TFDV, AWS, and OpenML teams for the collaboration opportunity that has helped my work gain more traction and visibility. I am also thankful to all my colleagues in the ADALab and Database Group for the many discussions and their feedback on my work.

Finally, I would have not been able to navigate through this journey without the support of my parents and my brother. I am extremely grateful to my many friends at UCSD for keeping the journey joyful, lively, and for the lovely memories.

The material in this dissertation is based on the following publications.

- Chapter 3 contains material from "Are key-foreign key joins safe to avoid when learning high-capacity classifiers?" by Vraj Shah, Arun Kumar, and Xiaojin Zhu, which appears in Proceedings of the 2018 Very Large Data Bases Conference (VLDB'18). The dissertation author was the primary investigator and author of this paper.

- Chapter 4 contains material from "The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML" by Vraj Shah and Arun Kumar, which appears in Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning (DEEM'19). The dissertation author was the primary investigator and author of this paper.

- Chapter 5 contains material from "Towards Benchmarking Feature Type Inference for AutoML Platforms" by Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar, which appears in Proceedings of the 2021 International Conference on Management of Data (SIGMOD'21). The dissertation author was the primary investigator and author of this paper.

- Chapter 6 contains material from "An Empirical Study on (Non-)Importance of Cleaning Categorical Duplicates before ML" by Vraj Shah, Thomas Parashos, and Arun Kumar, which has been submitted for publication of the material. The dissertation author was the primary investigator and author of this material.

My co-authors (Premanand Kumar, Jonathan Lacanlale, Thomas Parashos, Kevin Yang, and Xiaojin Zhu) have all kindly approved the inclusion of the aforementioned publications in my dissertation.

VITA

2016	B.Tech. in Computer Science, Indian Institute of Technology, Indore, India
2021	M.S. in Computer Science, University of California San Diego
2022	Ph.D. in Computer Science, University of California San Diego

PUBLICATIONS

Vraj Shah, Thomas Parashos, and Arun Kumar, "An Empirical Study on (Non-)Importance of Cleaning Categorical Duplicates before ML," Under Submission.

Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar, "Towards Benchmarking Feature Type Inference for AutoML Platforms," Proceedings of the 2021 International Conference on Management of Data (SIGMOD'21), 1584–1596, 2021.

Vraj Shah, Side Li, Arun Kumar, and Lawrence Saul, "SpeakQL: Towards Speech-driven Multimodal Querying of Structured Data," In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), 2363–2374, 2020.

Vraj Shah, Side Li, Kevin Yang, Arun Kumar, and Lawrence Saul, "Demonstration of SpeakQL: Speech-driven Multimodal Querying of Structured Data," In Proceedings of the 2019 International Conference on Management of Data (SIGMOD'19), 2001–2004, 2019.

Vraj Shah and Arun Kumar, "The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML," In Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning (DEEM'19), 1–4, 2019.

Vraj Shah, "SpeakQL: Towards Speech-driven Multimodal Querying," In Proceedings of the 2019 International Conference on Management of Data (SIGMOD'19), 1847–1849, 2019.

Vraj Shah, Side Li, Kevin Yang, Arun Kumar, and Lawrence Saul, "Demonstration of SpeakQL: Speech-driven Multimodal Querying of Structured Data," In Proceedings of the 2019 International Conference on Management of Data (SIGMOD'19), 2001–2004, 2019.

Vraj Shah, Arun Kumar, and Xiaojin Zhu, "Are key-foreign key joins safe to avoid when learning high-capacity classifiers?," Proceedings of the 2018 Very Large Data Bases Conference (VLDB'18), 366–379, 2018.

Dharmil Chandarana, **Vraj Shah**, Arun Kumar, and Lawrence Saul, "SpeakQL: Towards Speech-driven Multi-modal Querying," In Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics (HILDA'17), 1–6, 2017.

ABSTRACT OF THE DISSERTATION

Simplifying Data Preparation for Machine Learning on Tabular Data

by

Vraj Shah

Doctor of Philosophy in Computer Science

University of California San Diego, 2022

Professor Arun Kumar, Chair

Machine learning (ML) over tabular data has become ubiquitous with applications in many domains. This success has led to the rise of ML platforms, including automated ML (AutoML) platforms to manage the end-to-end ML workflow. The tedious grunt work involved in data preparation (prep) reduces data scientist productivity and slows down the ML development lifecycle, which makes the automation of data prep even more critical. While many works have looked into feature engineering and model selection in the end-to-end ML workflows, little attention has been paid towards understanding data prep and its utility for ML. Also, automating data prep remains challenging due to several reasons such as semantic gaps and lack of ways to objectively measure accuracy. In this dissertation, we take a step towards addressing such

challenges using database schema management and ML techniques to simplify, better automate, and understand the utility of ML data prep. We create new benchmark datasets, methodology for benchmarking and automating ML data prep, and devise novel empirical analyses to characterize the significance of critical data prep steps. Our work presents several critical artifacts that not only provide a systematic approach to reduce grunt work and improve the productivity of ML practitioners but also can help establish the science of building (Auto)ML platforms. Our work opens up several new research directions at the intersection of ML, data management, and ML system design.

Chapter 1

Introduction

Tabular (or relational) data remains the most commonly reported type of data for business critical use cases of machine learning (ML), as per the Kaggle State of Data Science survey [12]. ML over tabular data is ubiquitous in application sectors such as finance, healthcare, insurance, ecommerce, and retail [50, 152]. Applying ML over tabular data involves a complex pipeline where the ML algorithm is only a tiny component in the overall workflow [132]. This pipeline can include several stages such as data preparation (prep), feature engineering, and model and hyper-parameter search, as Figure 1.1 shows. This has led to a growing interest in both research and industry to support, manage, and automate the entire ML development lifecycle over tabular data [170, 45].

From conversations with our collaborators at TensorFlow Extended team at Google [45] and Salesforce Einstein [153], we find that their ML platform is already being used to support ML over tens of thousands of tabular datasets in production settings. Additionally, ML platforms such as Airbnb's Zipline [3], Uber's Michelangelo [20], Facebook's FBLearner Flow [6], and commercial platforms such as H20.AI [11], Salesforce Einstein [153], and DataRobot [5] have drawn a massive investment from the industry. This platformization of ML has also created an opportunity to automate several components of the ML workflow. In particular, the paradigm

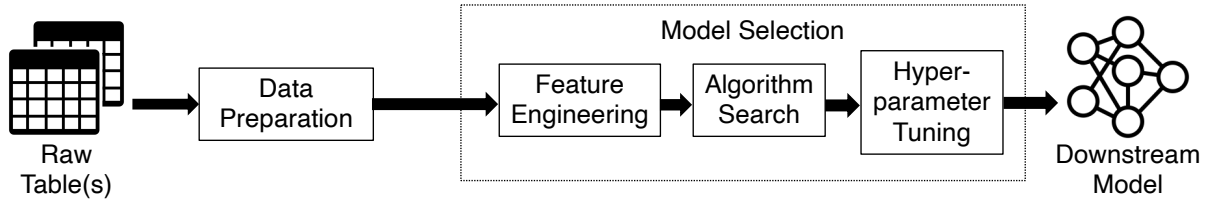


Figure 1.1: A simplified ML Development Lifecycle.

of AutoML aims to automate the entire ML workflow with the opportunity to enable ML for the masses. AutoML platforms such as Einstein AutoML [13], AutoML Tables [9], and AutoGluon [62] build ML models on millions of datasets from thousands of small-and-medium enterprises automatically. The central goal of these platforms is to get an accurate model for the prediction task while achieving the maximum possible automation of the end-to-end ML workflow. Several components of the end-to-end ML workflow such as automated model selection and hyper-parameter search are well-studied in the community [80, 67]. However, little attention has been paid to understanding and automating an equally important component of (Auto)ML platforms: data prep.

Data prep also remains particularly challenging for tabular data for data scientists who perform human-in-the-loop ML analytics. Surveys of such users have repeatedly shown that they spend from 45% up to 80% of their time on data prep [53, 163, 152]. Self-service data prep tools such as Trifacta [18], Tableau Prep [154], and Informatica Enterprise Data Prep [151] aim to reduce human effort and time involved with data prep grunt work. Thus, such human-in-the-loop tools also desire better automation and simplification of data prep to improve the overall productivity of their users.

Despite the importance of data prep for ML, there exists no consensus on what exactly constitutes data prep [34]. Depending upon the application, data prep can involve many diverse tasks such as integrating data from multiple sources, inferring feature types, performing feature transformations based on their types, and organizing feature values. Thus, data prep, in general, is ill-defined, unlike say TPC benchmarks in the database community where the query semantics

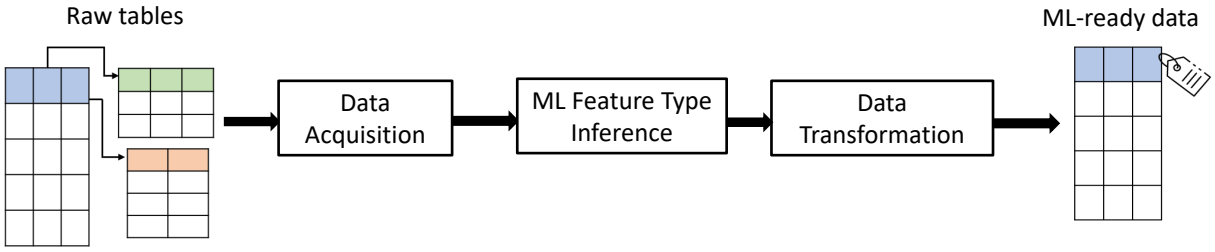


Figure 1.2: A simplified high-level ML Data Prep workflow that supports the ML-readiness properties.

are well-defined [16]. However, in the context of (Auto)ML, the end goal is crystal clear: build an accurate model on the raw table(s). This makes defining data prep more tractable since the specific tasks can be measured by their effect on the end ML accuracy.

Defining ML Data Prep. We first take a scientific approach towards understanding ML data prep (or data prep for ML); we define this as a stage where raw data is transformed to "ML-ready" data before the feature engineering stage of the model selection begins. At a high-level, we characterize this in terms of three ML readiness properties of the data. We show the corresponding ML data prep workflow that supports these ML-readiness properties in Figure 1.2. There exist enormous variety of ML data prep task, but we specifically focus on these steps since they are common in existing ML data prep workflows but have received much less attention in the research community.

1. *Discriminative Attributes.* An ML application needs features that can represent different kinds of signals. For instance, Figure 1.3 shows a normalized dataset where informative signals are distributed across tables. Thus, one would perform "data acquisition," a data prep step to gather additional features. This is typically performed through joins of multiple tables.

2. *Accurate ML Feature Types.* Any ML models can not directly operate over raw attribute types, but needs ML feature types with either continuous or discrete domain. For instance, in Figure 1.3 example, *Zip* needs to be recognized as *Categorical* and *Age* as *Numeric*. Thus, data prep step of "ML Feature Type Inference" is needed to determine the correct feature types for

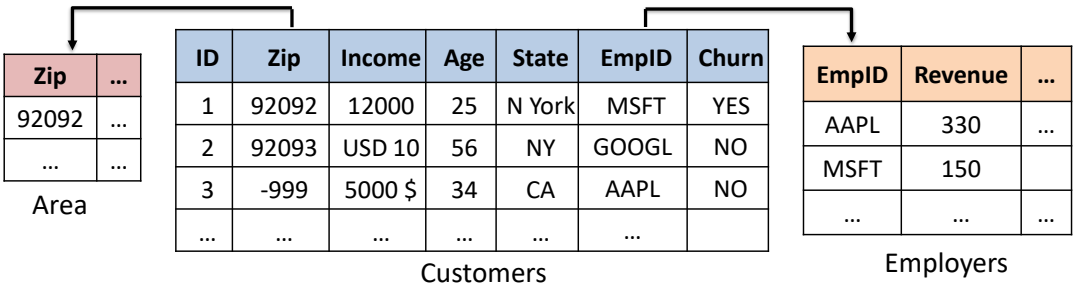


Figure 1.3: Example scenario for ML churn prediction task with multi-tabled data about customers. The schema for the dataset is *Customers* (ID, Zip, Income, Age, State, EmpID, Churn), *Employers* (EmpID, Revenue, State), *Area* (Zip, Density, CrimeRate).

every column of the table.

3. *Consistent Attribute Values.* With the attribute values in a consistent representation, one can obtain high-quality signals to build accurate ML models. For instance, in Figure 1.3, the *State* column have duplicates diluting the domain of the *Categorical* feature. Also, the values of the *Income* attribute are in various currencies. Thus, "data transformation" steps are performed to fix them.

Prior Art. Self-service visual tools such as Trifacta [18], Excel [35], and Tableau Prep [154] make it easier for data scientists to perform many data prep steps with a human-in-the-loop. To automate data prep tasks, there exist open-source AutoML tools such as TransmogrifAI in Einstein [17], Tensorflow Data Validation (TFDV) in TensorFlow Extended [45], and AutoGluon from AWS [62]. However, they rely on ad hoc syntactical and rule-based pipelines and are thus often limited in accuracy. For instance, consider what TransmogrifAI does on the dataset in Figure 1.3 for a common ML task, customer churn prediction. It assumes a single table as input, thus requiring users to perform joins when dealing with normalized datasets. Also, it wrongly calls many *Categorical* features with integer values as *Numeric*, e.g., ZipCode. This can cause the downstream model to produce garbage results. Also, many important data transformation steps are not handled. For instance, *Income* column values are not normalized in a common representation. Thus, the numerical signal from the column can likely be completely ignored

by the model. Moreover, a large number of duplicates in *State* column can potentially cause the feature vector dimensionality to expand which can make the learning difficult.

Challenges. When using existing AutoML tools, data prep issues discussed above can lead to loss of information and can potentially reduce the accuracy of the model, or even cause it to fail in some scenarios. Also, performing joins before ML puts an additional burden on not only the AutoML users but also data scientists when performing human-in-the-loop ML analytics. The effectiveness of data prep automation of existing tools is not known because of the lack of formalized task definitions and objective benchmarks to assess them. In other words, there exist several open questions: Just how good is the automation of data prep in such AutoML tools? How can one do better? How can we improve the productivity of the users involved? What is the significance of data prep for downstream ML model accuracy?

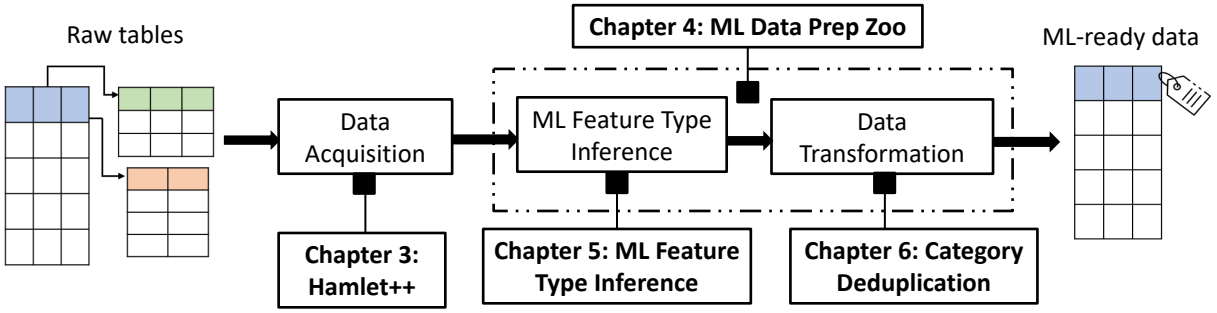
Considering these open questions, the thesis of this dissertation is that:

> *For many critical ML data prep steps, database schema management and ML techniques can be used to simplify, objectively quantify, and automate them to significantly improve the accuracy of automation, improve productivity, and better the understanding of accuracy implications for downstream ML.*

With a mix of contributions such as creating new benchmark labeled datasets, new methodologies, and novel empirical analyses, this dissertation shows how we can simplify, significantly improve the accuracy of automation, and characterize the significance of critical ML data prep steps. We open source our research artifacts to invite community-driven contributions in this direction.

1.1 Technical Contributions

In this dissertation, we choose three ubiquitous data prep tasks, one from each component of the ML data prep workflow: joining tables with key-foreign key dependencies between them,



	Methodological	New Research Artifacts: Benchmark Dataset and ML models	Empirical Comparisons and Insights
Chapter 3: Hamlet++			✓
Chapter 4: ML Data Prep Zoo	✓		
Chapter 5: ML Feature Type Inference	✓	✓	✓
Chapter 6: Category Deduplication	✓	✓	✓

Figure 1.4: The type of contributions our work makes with respect to the ML Data Prep workflow.

ML feature type inference, and category deduplication. We specifically choose these steps because they are commonly practiced in ML workflows but they have received little attention in the research community. We show how database schema management and ML techniques can help achieve several goals such as understanding their utility for ML, objective quantification of different tasks, and also improved accuracy of automation. Figure 1.4 summarizes our contributions. We explain our contributions in three parts as follows.

HAMLET++: Avoiding Joins when Learning High-Capacity Classifiers. In this project, we exploit database schema information with key-foreign key dependencies (KFKD) between the tables to decide when to avoid join without significantly affecting the ML accuracy of high-capacity classifiers; thus, reducing the data acquisition burden. A prior work [100] showed that one can often omit an entire table by exploiting KFKDs in the *schema* (referred to as "avoid the join"), but do so without significantly reducing ML accuracy ("safely") of linear models. The basis for this is that a KFK join creates a *functional dependency* between the *foreign key* and the features brought in by the join. But their work applied only to linear classifiers.

We perform a comprehensive empirical and simulation study and analysis to validate the applicability of the idea of avoiding joins safely to three popular "high-capacity" (i.e., with large or even infinite VC dimensions) classifiers: decision trees, SVMs, and ANNs. Our empirical analysis refutes an intuition from prior work and shows that these classifiers are counter-intuitively more robust to avoiding joins than linear classifiers. Our results raise new research questions at the intersection of data management and ML theory. Moreover, we highlight two new practical bottlenecks caused by large domain sizes of foreign key features. We offer novel methods to improve the interpretability and smoothing mechanism of foreign key features when dealing with decision trees. We verify their effectiveness empirically. A paper on this work appeared at the VLDB conference in 2018 [134]. This project is joint work with Arun Kumar and Xiaojin Zhu and is the subject of Chapter 3.

ML DATA PREP ZOO: Systematically automating and benchmarking ML Data Prep. In this work, we set up our vision for a methodological and a community-driven effort towards automating and benchmarking ML data prep steps such as ML feature type inference and data transformation steps. We find that there exists a *semantic gap* between what an *attribute* is in a database schema and what a *feature* is for ML. This makes tabular data not directly consumable by ML. We use ML to close the semantic gap by abstracting specific ML data prep tasks and cast them as applied ML tasks. For impactful research on this problem, we believe the major impediment is not new algorithms or theory but rather common task definitions and benchmark labeled datasets. We discuss research challenges in scaling up data labeling, defining accuracy metrics, and creating practical tool support. Finally, we announce a public "zoo" of labeled datasets and pre-trained ML models for data prep tasks to act as a community-led repository for further research on this problem. A vision paper on this work appeared at the DEEM workshop as part of the SIGMOD conference in 2019 [133]. This project is joint work with Arun Kumar and is the subject of Chapter 4.

ML FEATURE TYPE INFERENCE: Benchmarking and Automating for ML. In this

project, as a case study of our ML Data Prep Zoo vision, we initiate work on benchmarking and automating the ML feature type inference step by capturing ML semantics of attribute types in the database schema. The semantic gap between attribute types (e.g., strings, numbers) in databases/files and ML feature types (e.g., *Numeric*, *Categorical*) necessitates type inference. We formalize and standardize this task by creating a benchmark labeled dataset, which we use to objectively evaluate existing AutoML tools. Our dataset has 9921 examples and a 9-class label vocabulary. Our labeled data also offers an alternative approach to automate this task than existing rule-based or syntax-based approaches: use ML itself to predict feature types. We collate a benchmark suite of 30 classification and regression tasks to assess the importance of type inference for downstream models.

An empirical comparison on our labeled data shows that an ML-based approach delivers a lift of an average 14% and up to 38% in accuracy for identifying feature types compared to prominent industrial tools. Our downstream benchmark suite reveals that the ML-based approach outperforms existing industrial-strength tools for 47 out of 60 downstream models. We release our labeled dataset, models, and downstream benchmarks as part of the ML Data Prep Zoo public repository with a leaderboard. A paper on this work appeared at the SIGMOD conference in 2020 [135]. This project is joint work with Arun Kumar, Jonathan Lacanlale, Premanand Kumar, and Kevin Yang and is the subject of Chapter 5.

CATEGORY DEDUPLICATION: Assessing Importance for ML and Making Automation Accurate. In this project, we take the first step towards empirically characterizing the impact of Category Deduplication, an important data transformation step on ML with a three-pronged approach. We first study how duplicates exhibit themselves by creating a labeled dataset of 1248 *Categorical* columns, where true entities in each column are annotated with their duplicates. We then curate a downstream benchmark suite of 14 real-world datasets to make observations on the effect of duplicates on three popular downstream classifiers: Logistic Regression, Random Forest, and Multi-layer Perceptron (MLP). We finally use simulation studies to validate our observations

and disentangle multiple confounders that impact ML. We find that duplicates can cause Random Forest accuracy to drop by a median of 1.6% and up to 11.5% compared to the truth. Moreover, Logistic Regression and *Similarity* encoding for *Categoricals* are more robust to duplicates than *One-hot* encoding on the two high-capacity classifiers, Random Forest and MLP.

We also pursue a complementary research direction of accurately automating category deduplication task using our labeled dataset. We cast this task as a binary classification problem and build rule-based and learning-based approaches to evaluate them on the task. Our feasibility study shows that our labeled data is useful to deliver ML-based solutions that are more accurate than rule-based baselines. We provide intuitive explanations and takeaways that can potentially help both AutoML developers to build better platforms and ML practitioners to reduce grunt work. Our work presents novel data artifacts and benchmarks, as well as novel empirical analyses that can help advance the science of data prep on AutoML platforms. This project is joint work with Arun Kumar and Thomas Parashos and is the subject of Chapter 6.

1.2 Practical Impact

Throughout this dissertation work, we interacted with AutoML platform developers, researchers, and data scientists at various companies and open-source communities to understand their challenges and make it easier for them to adopt our contributions. At the time of writing this document, the contributions discussed in this dissertation have the following practical impacts.

- Google collaborated with us to adopt the best performing ML models of ML FEATURE TYPE INFERENCE into Tensorflow Data Validation (TFDV). TFDV is a part of a larger ML platform called TensorFlow Extended [45] and its role is to perform several data validation steps before the raw data is fed to ML pipelines. One such step is ML schema inference, where the ML feature types are identified and directly consumed by the downstream pipelines. We added two modules to their open source code: a scalable way to do featur-

ization to extract the features for our models and an inference module with our Random Forest to populate their schema with our feature types. We improved their inference of *Categorical* feature type with integer domain [137]. Google engineers are now reviewing it on internal benchmarks for adoption.

- A library that implements ML-based feature type inference is open-sourced as Python PIP package [150]. The library provides easy-to-use APIs to collect ML feature types given a raw table as Pandas DataFrame.

- We are collaborating with OpenML [155], which is an open-source platform for ML practitioners and researchers to share their models, datasets, and workflows for reuse. In the present state, users upload their tabular datasets to the platform with manual annotations of feature types. The Python library based on our ML FEATURE TYPE INFERENCE work also provides APIs to interface with OpenML's Attribute-Relation File Format (ARFF) feature types [150]. Currently, the OpenML team is doing an internal review to make this library automatically fill-in feature types for the users. Thus, our work can potentially save manual grunt work for $\sim$ 14000 OpenML users.

- AIMaker is an ongoing multi-disciplinary project at UCSD to enable users from different domains to find and share AI models and datasets [85]. Our ML FEATURE TYPE INFERENCE work is a key component of this project to help automatically match AI Models and Datasets.

- Integration of our ML feature type inference models with the AutoGluon Tabular project at AWS is currently under development [149]. A preliminary analysis has shown that AutoGluon using ML FEATURE TYPE INFERENCE gives more accurate downstream models than their existing setup.

- Several teams at AWS such as Glue [4], SageMaker [148], and Redshift ML [147] have

expressed interest in adopting our data and models from ML FEATURE TYPE INFERENCE for production use.

1.3 Summary

Reducing manual grunt work involved with ML data prep over tabular data is a major focus area for both researchers and industry building (Auto)ML platforms. Although the major focus of the end-to-end ML research effort has been on studying automated model selection and hyper-parameter search in the end-to-end ML pipeline, in this dissertation, we bring the community's attention to the importance of understanding and formalizing ML data prep steps. We take three specific commonly performed tasks in the ML workflow. We then use database schema management and ML techniques to simplify, objectively measure, and better automate data prep, thereby also improving the productivity of the users involved. We make a variety of contributions such as a novel methodological approaches, new benchmark labeled dataset, pre-trained ML models, and empirical analyses and insights towards our goals.

Our work in this space can help objectively validate and improve existing ML platforms and can substantially advance the science of building such platforms and automated data prep tools. Also, our work lays out an important foundation to address several open research questions at the intersection of ML theory, data management, ML system design, and human-computer interaction, e.g., complementary open research directions such as devising new labeling data prep interfaces, novel ML algorithms and architectures to accurately automate data prep, designing ML systems to guide them with appropriate data prep, and even ML theoretical quantification of the utility of data prep with bias-variance tradeoff analysis. More importantly, we believe that the research artifacts delivered by this dissertation are valuable to drive community-driven contributions. This can substantially accelerate the advancement in the science of further simplifying ML data prep over tabular data.

Chapter 2

Preliminaries

We intuitively explain the relevant ML terms, concepts, models, and theory and refer the interested reader to [75, 109, 139] for a deeper background.

Basics. We focus on classification and regression models, which need a *training dataset* with *labeled* examples to learn the *parameters* of the model. Examples include logistic regression, support vector machines (SVMs), decision trees, and artificial neural networks (ANNs). Most ML models assume that the examples are independently and identically distributed (IID) samples from an underlying (hidden) data distribution. Complex models known as statistical relational models avoid the IID assumption and handle correlated examples [66]. Such models are beyond the scope of this work. A trained model's prediction *error* (or *accuracy*) is measured using a *test dataset* not used for training. Popular testing methodologies include *holdout* validation and (nested) *k-fold cross-validation*. In the former, the labeled dataset is split three-ways: one for training, one for validation (say, to tune *hyper-parameters*), and one for final testing. In the latter, the labeled dataset is partitioned into k folds, with $k-1$ folds used for training (and validation) and the last fold used for testing; k error estimates are obtained by cycling through each fold for testing and averaged.

Models. Logistic regression and linear SVM classify examples using a hyperplane; thus,

they are called *linear* classifiers. Naive Bayes models the probability of the label by estimating the conditional probability distribution of each feature and multiplying them all; it can be viewed as a linear classifier [109]. 1-NN simply picks the training example nearest to a given example for prediction. Kernel SVM implicitly transforms feature vectors to a different representation and obtains the so-called "support vectors," which are examples that help separate classes. An ANN applies multiple layers of complex non-linear transformations to feature vectors to separate the classes. Finally, a decision tree learns a disjunction of conjunctive predicates to predict classes.

Theory. The set of prediction functions learnable by a model is called its *hypothesis space*. The test error has three components: *bias* (*approximation error*), *variance* (*estimation error*), and noise [139]. Informally, bias quantifies the error of a hypothetical "best" function in the hypothesis space and is related to the *capacity* of a model's hypothesis space (how many prediction functions it can represent), while variance quantifies the error of the actual prediction function obtained after training relative to the aforementioned hypothetical best function. Typically, a more complex model (say, with more parameters) has a lower bias but higher variance; this is the *bias-variance trade-off*. A classifier's capacity can be quantified using the Vapnik-Chervonenkis (VC) Dimension [139]. Intuitively, the VC dimension is the largest number of training examples the model can classify perfectly regardless of the training label distribution–a capability called "shattering." For example, logistic regression in 2 dimensions (features) can shatter at most 3 examples due to the "XOR problem" [156]. In general, given d features, its VC dimension is $d+1$. Decision trees, RBF-SVMs, and ANNs typically have large (even infinite) VC dimensions [139]; we call such models *high-capacity classifiers*. High-capacity classifiers often tend to have higher variance than simpler *linear models* (with VC dimensions linear in d), an issue colloquially called *overfitting*.

Feature Selection. Feature selection methods are meta-algorithms that are almost always used with an ML algorithm to improve accuracy and/or interpretability. There are three main types: (1) *Wrappers*: Also called subset selection, these methods use the ML algorithm as a

black-box to search through different feature subsets and pick the one with the lowest error. Since optimal subset selection is NP-Hard, various heuristic wrappers are popular in practice, including sequential greedy search [72]. (2) *Filters*: These methods assign a score to each feature (e.g., correlation co-efficient) and the top k features are selected. (3) *Embedded*: These methods "wire" feature selection into the ML algorithm, e.g., L1 or L2 regularization for logistic regression. Typically, feature selection alters the bias-variance balance by tolerating a small increase in bias for a larger decrease in variance and thus, reducing errors overall.

Chapter 3

Hamlet++: Avoiding Joins when Learning High-Capacity Classifiers

3.1 Introduction

In this chapter, we dive deeper into how we tackle the pains of acquiring the data for ML tasks. Applications often have many tables connected by database dependencies such as *key-foreign key dependencies* (KFKDs) [120]. Thus, given an ML task, data scientists almost always *join* multiple tables to obtain more *features* [99]. But conversations with data scientists at many enterprise and Web companies revealed that even this simple process of procuring tables is often painful in practice, since different tables are often "owned" by different teams with different access restrictions. This slows down the ML analytics lifecycle [90]. Recent reports of Google's production ML systems also show that features that yield marginal benefits incur high "technical debt" that decreases code mangeability and increases costs [131, 117].

In this context, [100] showed that one can often omit an entire table by exploiting KFKDs in the *schema* ("avoid the join"), but do so without significantly reducing ML accuracy ("safely"). The basis for this dramatic capability is that a KFK join creates a *functional dependency* (FD)

between the *foreign key* and the features brought in by the join, which we call "foreign features."[1]

Example (based on [100]). Consider a common classification task: predicting *customer churn*. The data scientist starts with the main table for training (simplified for exposition): `Customers` (`CustomerID`, `Churn`, `Gender`, `Age`, `Employer`). `Churn` is the *target*, while `Gender`, `Age`, and `Employer` are features. So far, this is a standard classification task. She then notices the table `Employers` (`Employer`, `State`, `Revenue`) in her database with extra features about customers' employers. `Customers.Employer` is thus a foreign key feature connecting these tables. She joins the tables to bring in the foreign features (about employers) because she has a hunch that customers employed by rich companies in coastal states might be less likely to churn. She then tries various classifiers, e.g., logistic regression or decision trees.

Using learning theory, [100] revealed a dichotomy in how safe it is to avoid a KFK join, which we summarize next. Essentially, ML error has two main components, *bias* and *variance*; informally, bias quantifies how complex the ML model is, while variance quantifies how tied the trained model is to the given training dataset [139]. Intuitively, more complex models have lower bias and higher variance; this is known as the *bias-variance trade-off*. Cases with high-variance are colloquially called *overfitting* [109]. Avoiding a KFK join is unlikely to raise the bias but likely to raise the variance, since foreign keys typically have larger domains than foreign features. *In simple terms, avoiding joins might cause extra overfitting*. But this extra overfitting subsides with more training examples, a behavior that was formally quantified using the powerful ML notion of *VC dimension*, which indicates the complexity of an ML model. Using this notion, [100] defined a new quantity, the *tuple ratio*, which is the ratio of the numbers of tuples of the tables being joined (customers and employers in our example). As the tuple ratio goes up, it becomes safer to avoid the join. Users can then configure a VC dimension-specific threshold based on their error tolerance. For simple classifiers with VC dimensions *linear* in the number of features (e.g., logistic regression and Naive Bayes), this threshold is as low as 20. This idea was empirically

[1] While KFKDs are not the same as FDs [140], assuming features have "closed" domains, they behave essentially as FDs in the output of the join [100].

validated with multiple real-world datasets.

While initially controversial, the idea of avoiding joins safely has been adopted by many data scientists, including at Facebook, LogicBlox, and MakeMyTrip [1]. Since the tuple ratio only needs the foreign table's cardinality rather than the table itself, data scientists can easily decide if they want to avoid the join or procure the extra table. However, the results in [100] had a major caveat–they applied only to linear classifiers. In fact, their VC dimension-based analysis suggested that the tuple ratio thresholds might be too high for high-capacity non-linear classifiers, potentially rendering this idea inapplicable to such classifiers in practice.

In this work, we perform a comprehensive empirical and simulation study and analysis to verify (or refute) the applicability of the idea of avoiding joins safely to three popular "high-capacity" (i.e., with large or even infinite VC dimensions) classifiers: decision trees, SVMs, and ANNs.

Such complex classifiers are known to be prone to overfitting [109]. Thus, the natural expectation is that avoiding a KFK join might cause more overfitting and raise the tuple ratio threshold compared to linear models (i.e., $\gg 20$). Surprisingly, our results show the *exact opposite*! We start by rerunning the experiments from [100] for such models; we also generalize the problem slightly to allow non-categorical features. Irrespective of which model is used, the same set of joins usually turn out to be safe to avoid. Furthermore, on the datasets that had joins that were not safe to avoid, the decrease in accuracy caused by avoiding said joins (unsafely) was lower for the high-capacity classifiers. In other words, *our work refutes an intuition from the VC dimension-based analysis of [100] and shows that these popular high-capacity classifiers are counter-intuitively comparably or more robust to avoiding KFK joins than linear classifiers, not less.*

To understand the above surprising behavior in depth, we conduct a Monte Carlo-style simulation study to stress test how safe it is to avoid a join. We use decision trees, since they were the most robust to avoiding joins. We generate data for a two-table KFK join and embed various

"true" distributions for the target. This includes a known "worst-case" scenario for avoiding joins for linear classifiers (i.e., errors blow up) [100]. We vary different properties of the data and the true distribution: numbers of features and training examples, noise, foreign key domain size, and skew. In very few cases does avoiding the join cause the error to rise beyond 1%. Indeed, the only scenario with much higher overfitting was when the tuple ratio was less than 3; this scenario arose in only 1 of the 7 real datasets. These results are in stark contrast to the results for linear classifiers.

Our counter-intuitive results raise new research questions at the intersection of data management and ML theory. There is a need to formalize the effects of KFKDs/FDs on the behavior of decision trees, SVMs, and ANNs. As a first step, we analyze and intuitively explain the behavior of decision trees and SVMs. Other open questions include the implications of more general database dependencies on the behavior of such models and the implications of all database dependencies for other ML tasks such as regression and clustering. We believe that solving these fundamental questions could lead to new ML analytics systems functionalities that make it easier to use ML for data analytics.

Finally, we observed two new practical bottlenecks caused by foreign key features, especially for decision trees. First, the sheer size of their domains makes it hard to interpret and visualize the trees. Second, some foreign key values may not have any training examples even if they are known to be in the domain. We adapt standard techniques to resolve these bottlenecks and verify their effectiveness empirically. Overall, the contributions of this work are as follows:

- To the best of our knowledge, this is the first work to analyze the effects of avoiding KFK joins on three popular high-capacity classifiers: decision trees, SVMs, and ANNs. We present a comprehensive empirical study that refutes an intuition from prior work and shows that these classifiers are counter-intuitively more robust to avoiding joins than linear classifiers.

- We conduct an in-depth simulation study with a decision tree to assess the effects of various data properties on how safe it is to avoid a KFK join.
- We present an intuitive analysis to explain the behavior of decision trees and SVMs when joins are avoided. We identify open questions for research at the intersection of data management and ML theory.
- We resolve two new practical bottlenecks with foreign key features by adapting standard techniques.

3.2 Background

3.2.1 Notation

We focus on the standard star schema KFK join setting, which is ubiquitous in many applications, including retail, insurance, Web security, and recommendation systems [120, 100, 99]. The fact table, which has the target variable, is denoted $\mathbf{S}$. It has the schema $\mathbf{S}(\underline{SID}, Y, \mathbf{X}_S, FK_1, \ldots, FK_q)$. A dimension table is denoted $\mathbf{R}_i$ ($i = 1$ to q) and it has the schema $\mathbf{R}_i(\underline{RID_i}, \mathbf{X}_{R_i})$. Y is the target variable (class label), $\mathbf{X}_S$ and $\mathbf{X}_{R_i}$ are vectors (sequences) of features, RID_i is the primary key of $\mathbf{R}_i$, while FK_i is a foreign key feature that refers to $\mathbf{R}_i$. We call $\mathbf{X}_S$ *home* features and $\mathbf{X}_{R_i}$ *foreign* features. Let $\mathbf{T}$ be the output of the star join that constructs the full training dataset by *concatenating* the features from all base tables. In general, its schema is $\mathbf{T}(\underline{SID}, Y, \mathbf{X}_S, FK_1, \ldots, FK_q, \mathbf{X}_{R_1}, \ldots, \mathbf{X}_{R_q})$. In contrast to our setting, traditional ML formulations do not distinguish between home features, foreign keys, and foreign features. The number of tuples in $\mathbf{S}$ (resp. $\mathbf{R}_i$) is denoted n_S (resp. n_{R_i}); the number of features in $\mathbf{X}_S$ (resp. $\mathbf{X}_{R_i}$) is denoted d_S (resp. d_{R_i}). Without loss of generality, we assume that the join is not selective, which means n_S is also the number of tuples in $\mathbf{T}$. $\mathcal{D}_{FK_i}$ denotes the domain of FK_i and by definition, $|\mathcal{D}_{FK_i}| = n_{R_i}$. We call $\frac{n_S}{n_{R_i}}$ the *tuple ratio* for $\mathbf{R}_i$. If $q = 1$ (only one foreign table), we drop the subscript in the

notation and use $\mathbf{R}$, FK, and n_R; for simplicity of exposition, we will assume $q = 1$ and use this notation.

3.2.2 Background: Avoiding KFK Joins Safely

It was shown in [100] that the FD $FK \rightarrow X_R$ has an interesting and surprising implication for the bias-variance trade-off: avoiding the KFK join (i.e., omitting X_R) is unlikely to increase the bias because the hypothesis space of almost any classifier does not shrink when X_R is avoided, but in the context feature selection, avoiding the join could result in much higher variance. The latter is because $|\mathcal{D}_{FK}|$ is usually much larger than the domains of the features in X_R. For instance, `State` in our example only has 50 values but `Employer` could have millions. This dichotomy led to the idea of *avoiding joins safely*: avoid it only if the variance is unlikely to increase much. To enable this, [100] introduced a simple *decision rule* with a user-settable threshold based on their error tolerance. The decision rule adapts a standard bound on variance from the ML literature that grows with the VC dimension and shrinks with n_S and it was simplified for linear models to enable thresholding directly on the tuple ratio ($n_S/|\mathcal{D}_{FK}|$). Thus, as the tuple ratio goes up, there will be less overfitting, since there are more training examples relative to the model's capacity. For linear classifiers, a threshold of 20 ensured the extra overfitting was marginal. But since high-capacity classifiers are usually more prone to overfitting, this approach suggests that the tuple ratio threshold might have to be higher for such classifiers.

3.2.3 Assumptions and Scope

For the sake of tractability, we adopt some assumptions from [100], but also drop some others to generalize the problem. In particular, we drop the assumption that all features are categorical (finite discrete set); we allow numeric features. We focus on classification and retain the assumption that FK is not a (primary) key in $\mathbf{S}$; otherwise, it will *not* be "generalizable," i.e., all future examples will have values never seen before. In our example, `CustomerID` is not

Table 3.1: Dataset statistics. q is the number of dimension tables. n_S is the total number of labeled examples; since we use nested cross-validation, the tuple ratio listed is $0.6 \times n_S/n_R$. N/A means that dimension table can never be discarded because its corresponding foreign key has an "open domain" and is not generalizable.

Dataset	(n_S, d_S)	q	(n_R, d_R)	Tuple Ratio
Expedia	942142, 1	2	11939, 8	47.4
			37021, 14	N/A
Movies	1000209, 0	2	6040, 4	99.4
			3706, 21	162
Yelp	215879, 0	2	11535, 32	11.2
			43873, 6	3
Walmart	421570, 1	2	2340, 9	108.1
			45, 2	5620.9
LastFM	343747, 0	2	4099, 7	50
			50000, 4	4.1
Books	253120, 0	2	27876, 2	5.5
			49972, 4	3
Flights	66548, 20	3	540, 5	74
			3167, 6	12.6
			3170, 6	12.6

generalizable but the foreign key `Employer` is. We also retain the assumption that all feature domains are fully known during training; this is a standard assumptions in ML [109, 100]. Handling unseen feature values is called the "cold start" issue in ML [128]. In practice, cold start is often resolved by temporarily mapping new values to a known "Others" placeholder. As ML models are periodically retrained, feature domains are expanded with such new information. In particular, we assume $\mathcal{D}_{FK}$ is the same as the set of $\mathbf{R}.RID$ values (new FK_i values are mapped to "Others"). Our goal is *not* to create new ML or feature selection algorithms, nor is to ascertain which algorithm yields the best accuracy or runtime. We aim to expose and analyze how KFKDs/FDs enable us to dramatically discard foreign features a priori when learning some popular high-capacity classifiers.

3.3 Empirical Study with Real Data

We present results for 10 classifiers, including 7 high-capacity ones (CART decision tree with gini, information gain, and gain ratio; SVM with RBF and quadratic kernels; multi-

layer perceptron ANN; 1-nearest neighbor), and 3 linear classifiers (Naive Bayes with backward selection, logistic regression with L1 regularization, and linear SVM). We also tried a few other feature selection techniques for the linear classifiers: Naive Bayes with forward selection and filter methods and logistic regression L2 regularization. Since these additional linear classifiers did not provide any new insights, we omit them due to space constraints.

3.3.1 Datasets

We take the seven real datasets from [100]; these are originally from Kaggle, GroupLens, `openflights.org`, `mtg.upf.edu/node/1671`, and `last.fm`. Two datasets have binary targets (Flights and Expedia); the others have multi-class ordinal targets. However, to generalize the scope of the problem studied, we retain numeric features rather than discretize them as in [100]. The dataset statistics are provided in Table 3.1. We briefly describe the task for each dataset and explain what the foreign features are. More details about their schemas, including the list of all features are already in the public domain (listed in [100]). *All of our datasets, scripts, and code are available for download on our project webpage[2] to make reproducibility easier.*

Walmart: predict if department-wise sales will be high using past sales (fact table) joined with stores and weather/economic indicators.

Flights: predict if a route is codeshared by using other routes (fact table) joined with airlines, source, and destination airports.

Yelp: predict if a business will be rated highly using past ratings (fact table) joined with users and businesses.

MovieLens: predict if a movie will be rated highly using past ratings (fact table) joined with users and movies.

Expedia: predict if a hotel will be ranked highly using past search listings (fact table) joined with hotels and search events; one foreign key, viz., the search ID, has an "open" domain,

[2] `https://adalabucsd.github.io/hamlet.html`

i.e., past values will not be seen in the future, which makes it unusable as a feature.

LastFM: predict if a song will be played often using past play information (fact table) joined with users and artists.

Books: predict if a book will be rated highly using past ratings (fact table) joined with readers and books.

3.3.2 Methodology

We perform 10-fold nested cross-validation, with a random third of the examples in the training folds being used for validation during feature selection and/or hyper-parameter tuning). We compare two approaches for each classifier: *JoinAll*, which uses all features from all base tables (the current widespread practice), and *NoJoin*, which avoids all foreign features a priori (the approach we study). For additional insights, we also include a third approach for decision trees: *NoFK*, which uses all features except the foreign keys. We used the popular R packages "rpart" for the decision trees (but for gain ratio, we used "CORElearn") and "e1071" for the SVMs. For the ANNs, we used the popular Python library Keras on TensorFlow. For Naive Bayes, we used the code from [100], while for logistic regression with L1 regularization, we used the popular R package "glmnet". We used a standard grid search for hyper-parameter tuning, with the grids described in detail below.

Decision Trees: There are two hyper-parameters to tune: *minsplit* and *cp*. *minsplit* is the number of observations that must exist in a node for a split to be attempted. Any split that does not improve the fit by a factor of *cp* is pruned off. The grid is set as follows: *minsplit* $\in \{1, 10, 100, 10^3\}$ and *cp* $\in \{10^{-4}, 10^{-3}, 0.01, 0.1, 0\}$

RBF-SVM: There are two hyper-parameters: C and γ. C controls the cost of misclassification. $\gamma > 0$ controls the bandwidth in the Gaussian kernel; given points x_i and x_j, $k(x_i, x_j) = \exp(-\gamma \cdot \|x_i - x_j\|^2)$. The grid is set as follows: $C \in \{10^{-1}, 1, 10, 100, 10^3\}$ and $\gamma \in \{10^{-4}, 10^{-3}, 0.01, 0.1, 1, 10\}$. On *Movies* and *Expedia*, we perform an extra fine tuning step

with $\gamma \in \{2^{-7}, 2^{-6}, \ldots, 2^{3}\}$ to improve accuracy.

Quadratic-SVM: We tune the same hyper-parameters C and γ for the polynomial kernel of degree 2: $k(x_i, x_j) = (-\gamma\, x_i^T \cdot x_j)^{degree}$. We use the same grid as RBF-SVM.

Linear-SVM: We tune the C hyper-parameter for the linear kernel: $k(x_i, x_j) = x_i^T \cdot x_j$, $C \in \{10^{-1}, 1, 10, 100, 10^3\}$.

ANN: The multi-layer perceptron architecture comprises of 2 hidden units with 256 and 64 neurons respectively. Rectified linear unit (ReLU) is used as the activation function. In order to allow penalties on layer parameters, we do L_2 regularization, with the regularization parameter tuned using the following grid axis: $\{10^{-4}, 10^{-3}, 10^{-2}\}$. We choose the popular Adam stochastic gradient optimization algorithm [91] with the learning rate tuned using the following grid axis: $\{10^{-3}, 10^{-2}, 10^{-1}\}$. The other hyper-parameters of the Adam algorithm used the default values.

Logistic Regression: The glmnet package performs automatic hyper-parameter tuning for the L1 regularizer, as well as the optimization algorithm. However, it has three parameters to specify a desired convergence threshold and a limit on the execution time: *nlambda*, which we set to 100, *maxit*, which we set to 10000, and *thresh*, which we set to 0.001.

Tables 3.2 present the 10-fold cross-validation errors of all models on all datasets.

3.3.3 Results

Accuracy. Our first and most important observation is that for almost all the datasets (*Yelp* being the exception) and for all three split criteria, the error of the decision tree is comparable (a gap of within 0.01) between *NoJoin* and *JoinAll*. The trend is virtually the same for the RBF-SVM and ANN as well. We also observe that the trend is almost the same for the linear models, albeit less robustly so. Thus, regardless of whether our classifier is linear or higher capacity, the relative behavior of *NoJoin* vis-a-vis *JoinAll* is virtually the same. These results represent our key counter-intuitive finding: joins are no less safe to avoid with the high-capacity classifiers than with the linear classifiers. The absolute errors of the high-capacity classifiers is

Table 3.2: 10-fold CV errors for all ML models. We compare the accuracy of *JoinAll* and *NoJoin* within each model. For *Expedia* and *Flights*, we use the zero-one error; for the other datasets, we use the RMSE. The bold font marks the cases where the error of *NoJoin* is at least 0.01 higher than *JoinAll*.

(A) Dataset	Decision Tree									1NN	
	Gini			Information			Gain Ratio				
	JoinAll	*NoJoin*	*NoFK*	*JoinAll*	*NoJoin*	*NoFK*	*JoinAll*	*NoJoin*	*NoFK*	*JoinAll*	*NoJoin*
Expedia	0.2456	0.2524	0.2653	0.2457	0.2500	0.2643	0.2462	0.2464	0.2751	0.2743	0.2685
Movies	1.0263	1.0250	1.0544	1.0320	1.0327	1.0548	1.0345	1.0361	1.0627	1.0958	1.0658
Yelp	**1.2929**	**1.3096**	1.2242	**1.3062**	**1.3259**	1.2257	**1.2452**	**1.2634**	1.2252	1.3567	1.2951
Walmart	0.7829	0.7867	0.8358	0.7839	0.7877	0.8354	0.7893	0.7899	0.8312	0.7938	0.7645
LastFM	0.9431	0.9463	1.1299	0.9445	0.9447	1.1320	0.9547	0.9552	1.1341	1.0463	1.0451
Books	0.9771	0.9797	1.0293	0.9763	0.9845	1.0312	1.0091	1.0093	1.0661	1.0936	1.0857
Flights	0.1388	0.1442	0.2030	0.1440	0.1474	0.1977	0.1419	0.1385	0.1707	0.1128	0.1087

(B) Dataset	SVM						ANN		Naïve Bayes		Logistic Regression	
	Linear		Polynomial		RBF				BFS		L1	
	JoinAll	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*
Expedia	0.2131	0.2155	0.2075	0.2129	0.2049	0.2105	0.1896	0.1912	0.2423	0.2450	0.2134	0.2176
Movies	1.0337	1.0342	1.0147	1.0149	0.9855	0.9856	0.9754	0.9755	1.0678	1.0742	1.0350	1.0413
Yelp	**1.1950**	**1.2060**	**1.1553**	**1.1662**	**1.1263**	**1.1456**	1.1965	1.2052	**1.1145**	**1.1842**	**1.1250**	**1.1502**
Walmart	0.8448	0.8460	0.7942	0.7948	0.7651	0.7656	0.7354	0.7355	0.8821	0.8851	0.8240	0.8243
LastFM	**1.0953**	**1.1149**	1.0038	1.0081	0.9643	0.9691	**1.0102**	**1.0255**	**0.9645**	**0.9758**	0.9838	0.9869
Books	1.1239	1.1262	1.0234	1.0245	0.9839	0.9856	0.9588	0.9585	1.0667	1.0674	**0.9719**	**0.9831**
Flights	0.1229	0.1274	0.1013	0.1065	0.0752	0.0802	0.0651	0.0688	0.1313	0.1353	0.1205	0.1230

mostly lower than the linear models; this is expected but orthogonal to our focus. Interestingly, on *Yelp*, in which both joins are known to be *not* safe to avoid for linear models [100], *NoJoin* correctly sees a large rise in error against *JoinAll*–almost 0.07 for Naive Bayes. But the rise is smaller for some high-capacity classifiers, e.g., RBF-SVM, Gini decision tree, and ANN all see a rise less than 0.03. Thus, these high-capacity classifiers are counter-intuitively *more* robust than linear classifiers to avoiding joins.

We also see that *NoFK* often has much higher errors than *JoinAll* and *NoJoin*. Thus, foreign key features are useful even for high-capacity classifiers; it is known for linear classifiers that dropping foreign keys causes bias to shoot up [100]. Interestingly, on *Yelp*, which has very low tuple ratios, *NoFK* has much lower errors than *JoinAll* and *NoJoin*.

Table 3.3: Robustness results for discarding individual dimension tables with a Gini decision tree.

Dataset	Expedia	Movies	Yelp	Walmart	LastFM	Books
NoR1	0.2456	1.0261	1.2918	0.7846	0.9434	0.9772
NoR2	X	1.0252	1.3077	0.7844	0.9465	0.9869
JoinAll	0.2452	1.0263	1.2929	0.7829	0.9431	0.9771
NoJoins	0.2524	1.0250	1.3096	0.7867	0.9463	0.9797

Flights : $NoR1 : 0.1387 \quad NoR2 : 0.1392 \quad NoR3 : 0.1404$
$NoR1,R2 : 0.1377 \quad NoR1,R3 : 0.1379 \quad NoR2,R3 : 0.1426$

To understand the above results more deeply, we conduct a "robustness" experiment by discarding dimension tables one at a time: Table 3.3 presents these results for the Gini decision tree. We see that the errors with dropping dimension tables one (or even two) at a time are all still within 0.01 of *NoJoin* in all cases, except for *Yelp*. Even on *Yelp*, the error increases significantly only when $\mathbf{R}_2$ (users table) is dropped, not $\mathbf{R}_1$. As Table 3.1 shows, the tuple ratio for $\mathbf{R}_2$ is only 3, while that for $\mathbf{R}_1$ is 11.2. Interestingly, the tuple ratio is similarly low (3) for $\mathbf{R}_2$ in *Books* but *NoJoin* error is not much higher. Thus, the tuple ratio is only a *conservative* indicator: it can tell if an error is likely to rise but the error may not actually rise in some cases. Almost every other dimension table can safely be discarded. The results were similar for ANN on *Yelp* and for the RBF-SVM on *Yelp*, *LastFM*, and *Books*; we skip these for brevity.

Overall, out of 14 dimension tables across the 7 datasets, we are able to safely avoid (with a tolerance of 0.01) 13 for decision trees and ANN with a tuple ratio threshold of about 3. For RBF-SVM, we were able to safely avoid 11 dimension tables with a tuple ratio threshold being of about 6. These are in stark contrast to the more modest results reported with the linear classifiers in [100]: only 7 of the dimension tables could be safely avoided, that too with a tuple ratio threshold of 20. *Thus, we see that the decision trees and ANN need six times fewer training examples and the RBF-SVM needs three times fewer training examples than linear classifiers to*

Table 3.4: Results of the hypothesis tests.

Dataset	eps = 0		eps = 0.01	
	α=0.05	α=0.005	α=0.05	α=0.005
Expedia	3	5	10	10
Movies	8	8	10	10
Yelp	1	1	4	5
Walmart	7	8	10	10
LastFM	4	6	9	9
Books	6	7	10	10
Flights	2	4	10	10

avoid significant extra overfitting when avoiding KFK joins. These results are counter-intuitive because such complex classifiers are known to need more (not less) training examples to avoid extra overfitting.

For an interesting comparison that we use later in Section 3.5, we also show the results for 1-NN (from "RWeka" in R) in Table 3.2. Surprisingly, this "braindead" classifier has significantly *lower* errors with *NoJoin* than *JoinAll* for most datasets! We discuss this behavior further in Section 3.5.

Hypothesis Tests. The cross-validation errors suggest that *NoJoin* is not significantly worse than *JoinAll* for most datasets, especially those with high tuple ratios. We now validate if the error differences are indeed statistically significant for a given error tolerance. We perform a one-tailed t-test with the ten folds' asymmetric error differences between *NoJoin* and *JoinAll* for each model on each dataset. We set the tolerance (ε) to both 0 and 0.01. The null hypothesis is that the error difference is not significantly higher than ε. Table 3.4 lists the number of models for which the null hypothesis was *not rejected* for the standard $\alpha = 0.05$ confidence level and the recently recommended stricter level of $\alpha = 0.005$ [2]. We see that except on *Yelp*, which has very low tuple ratios, *NoJoin* is *not* statistically significantly worse than *JoinAll* for most models (both linear and higher capacity), especially for $\varepsilon = 0.01$ but also for $\varepsilon = 0$ in many cases. For example, logistic regression on *Movies* and *Yelp* has p-values of 0.97 and 0.000026 respectively for $\varepsilon = 0.01$. Since the p-value for *Movies* is greater than the α levels, the null hypothesis is

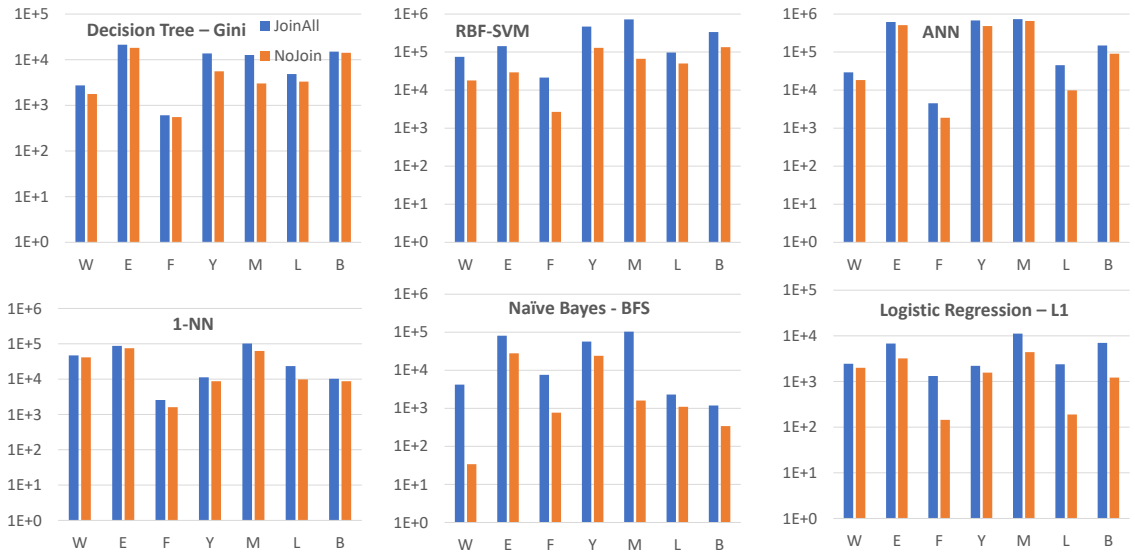


Figure 3.1: End-to-end runtimes on the real-world datasets: Walmart (W), Expedia (E), Flights (F), Yelp (Y), Movies (M), LastFM (L) and Books (B).

retained. But for *Yelp*, the null hypothesis is rejected as the p-value is far below the α levels. Due to space constraints, we skip the other p-values here but have released all the detailed results on our project webpage.

Runtimes. A key benefit of avoiding joins safely is that ML runtimes (including feature selection) could be significantly lowered for linear models [100]. We now check if this holds for high-capacity classifiers as well by comparing the end-to-end execution times (training, validation with grid search, and testing). Due to space constraints, we only report Gini metric for decision trees and RBF kernel for SVMs; these were also the most robust to avoiding joins. All experiments (except for ANN) were run on CloudLab [126]; we use a custom OpenStack profile running Ubuntu 14.10 with 40 Intel Xeon cores and 160GB of RAM. The ANN experiments were run on a commodity laptop with Nvidia GeForce GTX 1050 GPU, 16GB RAM and Windows 10. We used R version 3.2.2 and TensorFlow version 1.1.0. Figure 3.1 presents the results.

For the high-capacity classifiers, we saw an average speedup of about 2x for *NoJoin* over *JoinAll*. The highest speedup was on the *Movies*: 3.6x for the decision tree and 6.2x for the RBF-SVM. As for the ANN, *LastFM* reported the largest speedup of 2.5x. The speedup for the

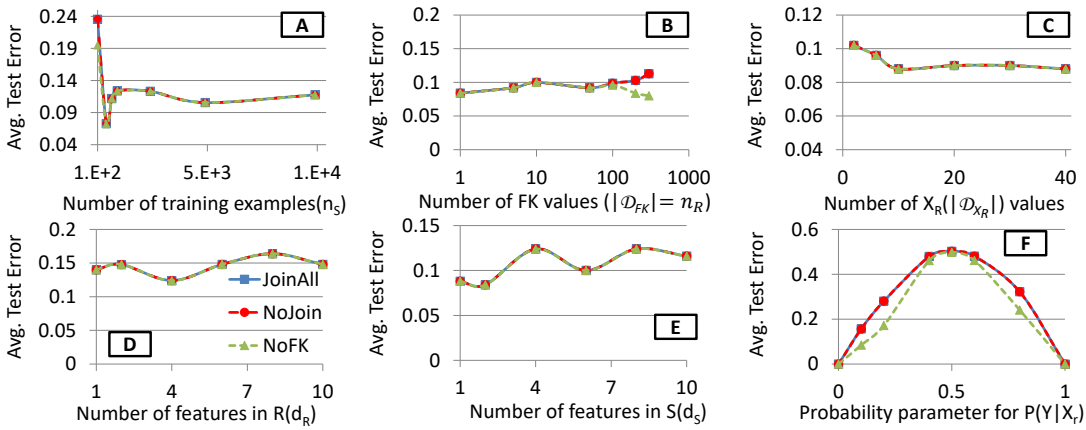


Figure 3.2: Simulation results for Scenario `OneXr`. For all plots except (E), we fix $p = 0.1$. Note that $n_R \equiv |\mathcal{D}_{FK}|$. (A) Vary n_S, while fixing $(n_R, d_S, d_R) = (40, 4, 4)$. (B) Vary n_R, while fixing $(n_S, d_S, d_R) = (1000, 4, 4)$. (C) Vary d_S, while fixing $(n_S, n_R, d_R) = (1000, 40, 4)$. (D) Vary d_R, while fixing $(n_R, d_S, d_R) = (1000, 40, 4)$. (E) Vary p, while fixing $(n_S, n_R, d_S, d_R) = (1000, 40, 4, 4)$. (F) Vary $|\mathcal{D}_{X_r}|$, while fixing $(n_S, n_R, d_S, d_R) = (1000, 40, 4, 4)$; all other features in $\mathbf{X}_R$ and $\mathbf{X}_S$ are binary.

linear classifiers were more significant, e.g., over 80x for Naive Bayes on on *Movies* and and about 20x for logistic regression on *LastFM*. These results corroborate the orders of magnitude speedup reported in [100].

3.4 In-depth Simulation Study

We now dive deeper into the behavior of the decision trees using a simulation study in which we vary the underlying "true" data distribution and sampling datasets of different dimensions. We focus on a two-table join for simplicity. We use the decision tree, since it exhibited the maximum robustness to avoiding KFK joins on the real data. Our study comprehensively "stress tests" this robustness. Note that our methodology is generic enough to be applicable to any other classifier too, since we only use generic notions of error and *net variance* as defined in [100].

Setup and Data Synthesis There is one dimension table $\mathbf{R}$ ($q = 1$), and all of $\mathbf{X}_S$, $\mathbf{X}_R$, and Y are boolean (domain size 2). We control the "true" distribution $P(Y, \mathbf{X})$ and sample labeled

examples in an IID manner from it. We study two different scenarios for what features are used to (probabilistically) determine Y: `OneXr` and `XSXR`. These scenarios represent opposite extremes for how likely the (test) error is likely to shoot up when $\mathbf{X}_R$ is discarded and FK is used as a representative [100]. In `OneXr`, a lone feature $X_r \in \mathbf{X}_R$ determines Y; the rest of $\mathbf{X}_R$ and $\mathbf{X}_S$ are random noise (but note that FK will not be noise because it functionally determines X_r). In `XSXR`, all features in $\mathbf{X}_S$ and $\mathbf{X}_R$ determine Y. Intuitively, `OneXr` is the worst-case scenario for discarding $\mathbf{X}_R$ because X_r is typically far more succinct than FK, which we expect to translate to less possibility of overfitting with *NoJoin*. Note that if we use FK directly in P, $\mathbf{X}_R$ can be more easily discarded because FK conveys more information anyway; so, we skip such a scenario.

The following data parameters are varied one at a time: number of training examples (n_S), size of foreign key domain ($|\mathcal{D}_{FK}| = n_R$), number of features in $\mathbf{X}_R$ (d_R), and number of features in $\mathbf{X}_S$ (d_S). We also sample $\frac{n_S}{4}$ examples each for the validation set (for hyper-parameter tuning) and the holdout test set (final indicator of error). We generate 100 different training datasets and measure the average test error and average net variance (as defined in [59]) based on the different models obtained from these 100 runs.

3.4.1 Scenario OneXr

The "true" distribution is set as follows: $P(Y = 0|X_r = 0) = P(Y = 1|X_r = 1) = p$, where p is called the probability skew parameter that controls the noise (also called Bayes error [75]). The exact procedure for sampling examples is as follows: (1) Construct tuples of $\mathbf{R}$ by sampling $\mathbf{X}_R$ values randomly (each feature value is an independent coin toss). (2) Construct the tuples of $\mathbf{S}$ by sampling $\mathbf{X}_S$ values randomly (independent coin tosses). (3) Assign FK values to $\mathbf{S}$ tuples uniformly randomly from $\mathcal{D}_{FK}$. (4) Assign Y values to $\mathbf{S}$ tuples by looking up into their respective X_r value (implicit join on $FK = RID$) and sampling from the above conditional distribution.

We compare *JoinAll*, *NoJoin*, and *NoFK*; we include *NoFK* for a lower bound on errors,

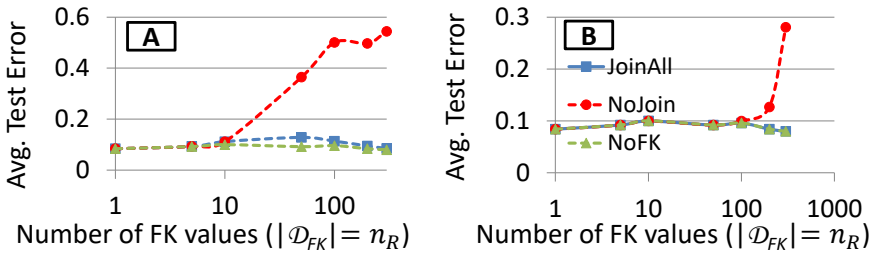


Figure 3.3: Scenario `OneXr` simulations with the same setup as Figure 3.2(B), except for (A) 1-NN and (B) RBF-SVM.

since we know FK does not directly determine Y (although indirectly it does).[3] Figure 3.2 presents the results for the test errors for varying each relevant data and distribution parameter, one at a time.

Interestingly, regardless of the parameter varied, in almost all cases, *NoJoin* and *JoinAll* have almost identical errors (close to the Bayes error)! From inspecting the actual decision trees learned in these two settings, we found that in almost all cases, FK was used repeatedly for partitioning; seldom was a feature from $\mathbf{X}_R$, including X_r, used. This suggests that FK can indeed act as a good representative of $\mathbf{X}_R$ even in this extreme case. In contrast to these results, [100] found that for linear models, the errors of *NoJoin* shot up compared to *JoinAll* (a gap of nearly 0.05) as the tuple ratio starts falling below 20. In stark contrast, as Figure 3.2(B) shows, even for a tuple ratio of just 3, *NoJoin* and *JoinAll* have similar errors with the decision tree. This corroborates the results seen for the decision tree on the real datasets (Table 3.2). When n_S becomes very low or when $|\mathcal{D}_{FK}|$ becomes very high, the absolute errors of *JoinAll* and *NoJoin* increase compared to *NoFK*. This suggests that when the tuple ratio is very low, *NoFK* is perhaps worth trying too. This is similar to the behavior seen on *Yelp*. Overall, *NoJoin* exhibits similar behavior as *JoinAll* in most cases.

We also ran this scenario for the RBF-SVM (and 1-NN); the trends were similar, except for the value of the tuple ratio at which *NoJoin* deviates from *JoinAll*. Figure 3.3 presents the results for the experiment in which we increase $|\mathcal{D}_{FK}| = n_R$, while fixing everything else, similar

[3] In general though, *NoFK* could have much higher errors if FK is part of the true distribution; indeed, *NoFK* had much higher errors on many real datasets (Table 3.2).

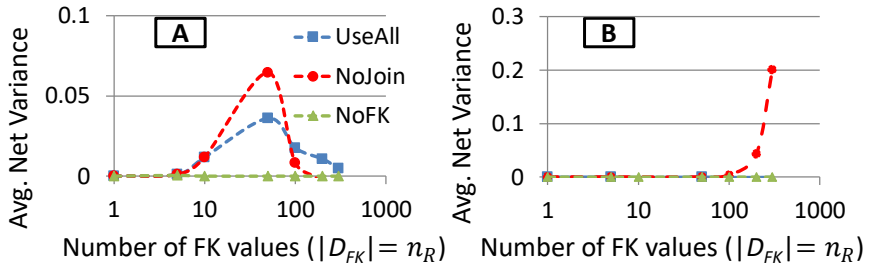


Figure 3.4: Average net variance in the scenario `OneXr` for (A) 1-NN and (B) RBF-SVM.

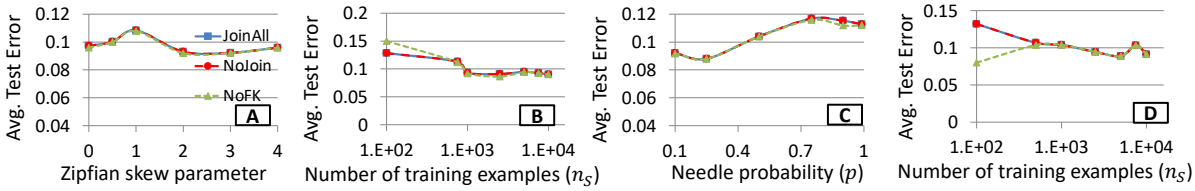


Figure 3.5: Scenario `OneXr` simulations with skew in $P(FK)$. (A-B) Zipfian skew. (C-D) Needle-and-thread skew. For (A) and (C), we vary the respective skew parameter (Zipfian skew parameter and needle probability), while fixing $(n_S, n_R, d_S, d_R) = (1000, 40, 4, 4)$. For (B) and (D), we vary n_S, while fixing $(n_R, d_S, d_R) = (40, 4, 4)$, the Zipfian skew parameter to 2 for (B), and the needle probability to 0.5 for (D).

to Figure 3.2(B) for the decision tree. We see that for the RBF-SVM, the error deviates when the tuple ratio falls below roughly 6. This corroborates its behavior on the real datasets (Table 3.2). The 1-NN, as expected, is far less stable and the deviation starts even at a tuple ratio of 100. As Figure 3.4 confirms, the deviation in error for the RBF-SVM is due to the net variance, which helps quantify the extra overfitting. This is akin to the extra overfitting reported in [100] using the plots of the net variance. Intriguingly, the 1-NN sees its net variance exhibit non-monotonic behavior; this is likely an artifact of its unstable behavior, since fewer and fewer training examples will match on FK as n_R keeps rising.

Finally, we also ran this scenario with a skew in $P(FK)$, which makes it less safe to avoid the join for linear classifiers [100]. But our simulations with a decision tree show that it is robust even to foreign key skew in terms of how safe it is to avoid the join. The regular `OneXr` scenario samples FK uniformly randomly from $\mathcal{D}_{FK}$ (step 3 in the procedure). We now ask if a *skew* in the distribution of FK values could widen the gap between *JoinAll* and *NoJoin*. To study this scenario, we modify the data generation procedure slightly: in step 3, we sample FK values with a

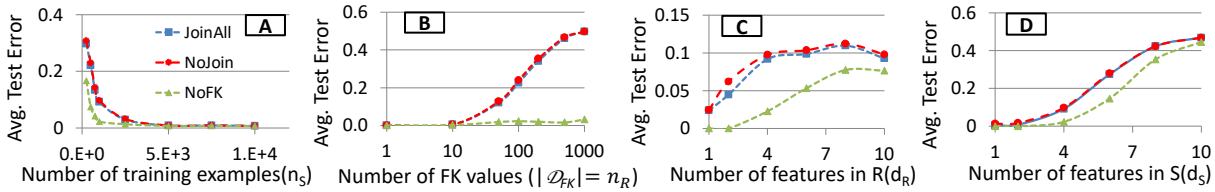


Figure 3.6: Simulation results for Scenario `XSXR`. The parameter values varied/fixed are the same as in Figure 3.2 (A)-(D).

Zipfian skew or a needle-and-thread skew. The Zipfian skew simply uses a Zipfian distribution for $P(FK)$ controlled by the Zipfian skew parameter. The needle-and-thread skew allocates a large probability mass (parameter p) to a single FK value (the "needle") and uniformly distributes the rest of the probability mass to all other FK values (the "thread"). For the linear model case, [100] reported that as the skew parameters increased, the gap widened. Figure 3.5 presents the results for the decision tree.

Surprisingly, the gap between *NoJoin* and *JoinAll* does not widen significantly no matter how much skew introduced in either the Zipfian or the needle-and-thread case! This result further affirms the remarkable robustness of the decision tree when discarding foreign features. As expected, *NoFK* is better when n_S is low, while overall, *NoJoin* is quite similar to *JoinAll*.

3.4.2 Scenario XSXR

Unlike `OneXr`, we now create a true distribution that maps $\mathbf{X} \equiv [\mathbf{X}_S, \mathbf{X}_R]$ to Y without any noise (Bayes error). The exact procedure for sampling examples is as follows: (1) Construct a true probability table (TPT) with entries for all possible values of $[\mathbf{X}_S, \mathbf{X}_R]$ and assign a random probability to each entry such that the total probability is 1. (2) For each entry in the TPT, pick a Y value randomly and append the TPT entry; this ensures $H(Y|\mathbf{X}) = 0$. (3) Marginalize the TPT to obtain $P(\mathbf{X}_R)$ and from it, sample $n_R = \mathcal{D}_{FK}$ tuples for $\mathbf{R}$ along with an associated sequential *RID* value. (4) In the original TPT, push the probability of each entry to 0 if its $\mathbf{X}_R$ values did not get picked for $\mathbf{R}$ in step 3. (5) Renormalize the TPT so that the total probability is 1 and sample n_S examples (Y values do not change) and construct $\mathbf{S}$. (6) For each tuple in $\mathbf{S}$, pick its FK value

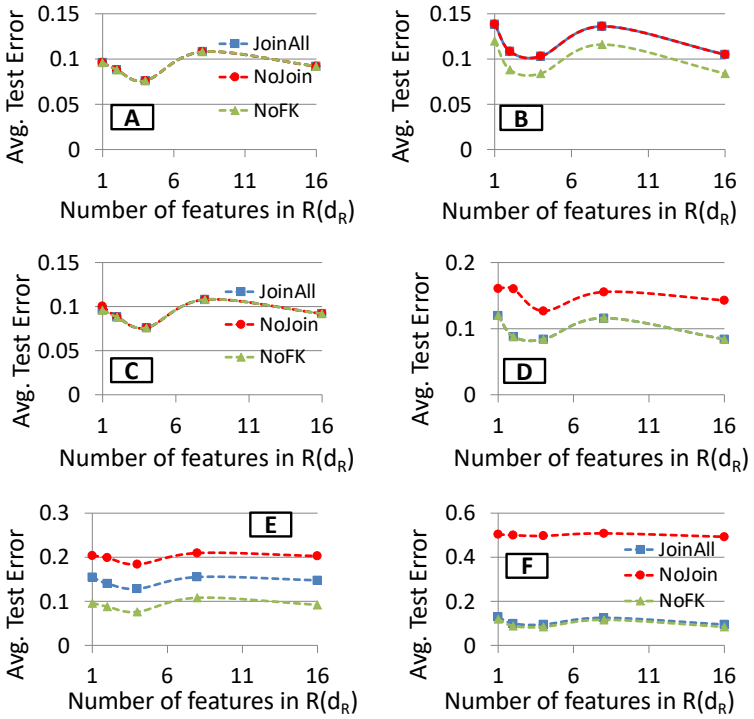


Figure 3.7: Scenario `RepOneXr` simulations for decision tree with (A-B), for RBF-SVM with (C-D), and for 1-NN with (E-F). (A,C,E) Vary d_R while fixing $(n_S, n_R, d_S) = (1000, 40, 4)$. (B,D,F) Vary d_R while fixing $(n_S, n_R, d_S) = (1000, 200, 4)$.

uniformly randomly from the subset of *RID* values that map to its $\mathbf{X}_R$ value in $\mathbf{R}$ (an implicit join). We again compare *JoinAll*, *NoJoin*, and *NoFK*. Figure 3.6 presents the results.

Once again, we see that *NoJoin* and *JoinAll* exhibit similar errors in almost all cases, with the largest gap being 0.017 in Figure 3.6(C). Interestingly, even when the tuple ratio is close to 1, the gap between *NoJoin* and *JoinAll* does not widen much. Figure 3.6(B)) shows that as $|\mathcal{D}_{FK}|$ increases, *NoFK* remains at low overall errors, unlike both *JoinAll* and *NoJoin*. But as we increase d_R or d_S, the gap between *JoinAll*/*NoJoin* and *NoFK* narrows because even *NoFK* does not have enough training examples. Of course, all gaps virtually disappear as the number of training examples increases, as shown by Figure 3.6(A). Overall, *NoJoin* again exhibits similar behavior as *JoinAll*.

Table 3.5: (A) Training errors for the same experiments as Table 3.2. Bold font marks the cases where the error of *NoJoin* is at least 0.01 higher than *JoinAll*.

(A) Dataset	Decision Tree									1NN	
	Gini			Information			Gain Ratio				
	JoinAll	*NoJoin*	*NoFK*	*JoinAll*	*NoJoin*	*NoFK*	*JoinAll*	*NoJoin*	*NoFK*	*JoinAll*	*NoJoin*
Expedia	0.1425	0.1478	0.1660	0.1442	0.1465	0.1641	0.1455	0.1454	0.1738	0	0
Movies	1.0038	1.0038	1.0476	1.0106	1.0119	1.0545	1.0145	1.0174	1.0551	1.0546	1.0420
Yelp	1.0425	1.0458	1.2149	1.0537	1.0571	1.2290	**1.0650**	**1.0844**	1.2358	1.3050	1.1958
Walmart	0.7339	0.7362	0.8212	0.7362	0.7346	0.8298	0.7373	0.7362	0.8176	0.7045	0.7058
LastFM	0.9153	0.9172	1.0947	0.9153	0.9155	1.1059	0.9247	0.9252	1.1092	1.0140	1.0137
Books	0.8439	0.8440	0.8909	0.8750	0.8750	0.8926	0.8957	0.8953	0.9317	1.0701	1.0540
Flights	0.0004	0.0005	0.0121	0.0008	0.0007	0.0079	0.1247	0.1253	0.1261	0	0

(B) Dataset	SVM						ANN		Naïve Bayes		Logistic Regression	
	Linear		Polynomial		RBF				BFS		L1	
	JoinAll	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*	*JoinAll*	*NoJoin*
Expedia	0.2038	0.2063	0.1934	0.1955	**0.1913**	**0.2017**	0.1770	0.1774	0.2350	0.2359	0.2057	0.2058
Movies	1.0149	1.0161	0.9851	0.9836	0.9685	0.9718	0.9558	0.9531	1.0347	1.0360	1.0152	1.0172
Yelp	1.1258	1.1239	**1.0838**	**1.0979**	**1.0743**	**1.0927**	1.1759	1.1749	**1.0572**	**1.1081**	**1.1041**	**1.1183**
Walmart	0.8254	0.8261	0.7752	0.7755	0.7456	0.7462	0.6956	0.6951	0.8541	0.8553	0.8028	0.8077
LastFM	**1.0461**	**1.0578**	0.9836	0.9830	0.9451	0.9466	0.9848	0.9852	0.9242	0.9253	0.9447	0.9462
Books	1.0737	1.0761	**1.0053**	**1.0059**	0.9534	0.9557	0.9338	0.9345	1.0235	1.0245	0.9436	0.9469
Flights	0.0874	0.0914	0.0189	0.0190	0	0	0.0451	0.0500	0.1246	0.1285	0.0971	0.1028

3.4.3 Scenario RepOneXr

We now present results for a new simulation scenario that is a slight twist on `OneXr`: the tuples of $\mathbf{R}$ are constructed by replicating the value of X_r sampled for a tuple to create all the other features in X_R. That is, X_R of an example is just the same value repeated d_R times. Note that the FD $FK \rightarrow X_R$ implies there are at least as many unique FK values as $\mathbf{X}_R$ values. Thus, by increasing $|\mathcal{D}_{FK}|$ relative to d_R, we hope to increase the chance of the model getting "confused" with *NoJoin*. Our goal is to see if this widens the gap between *JoinAll* and *NoJoin*. Figure 3.7 presents the results for the two experiments on decision trees where (A) has a high tuple ratio of 25 and (B) has a low tuple ratio of 5. Once again, *JoinAll* and *NoJoin* exhibit similar errors in both the cases. For the RBF-SVM, *NoJoin* has higher errors at the tuple ratio of 5 but not 25, while for the 1-NN, *NoJoin* has higher errors in both cases.

3.5 Analysis and Open Questions

3.5.1 Explaining the Results

We now intuitively explain the surprising behavior of decision trees and RBF-SVMs with *NoJoin* vis-a-vis *JoinAll*. We first ask: Does *NoJoin* compromise the "generalization error"? The generalization error is the difference of the test and train errors. Table 3.5 lists the train errors (averaged across the 10 folds). *JoinAll* and *NoJoin* are remarkably close for the decision trees (except for *Yelp*, of course). The absolute generalization errors are often high, e.g., train error is almost 0 on *Flights* with RBF-SVMs but test errors are about 0.08, but this is orthogonal to our focus–we only note that *NoJoin* does not increase this generalization error significantly. The same is true for all the decision trees. *Thus, avoiding the KFK joins safely did not significantly affect the generalization errors the high-capacity classifiers.*

Returning to 1-NN, Table 3.2 showed that it has similar errors as RBF-SVM on some datasets. We now explain why this comparison is useful: RBF-SVM behaves similar to 1-NN in some cases when FK is used (both *JoinAll* and *NoJoin*). But this does not necessarily hurt its test accuracy. Note that FK is represented using the standard one-hot encoding for RBF-SVM and 1-NN. So, FK can contribute to a maximum distance of 2 in a (squared) Euclidean distance between two examples x_i and x_j. But since $\mathbf{X}_R$ is functionally dependent on FK, if $x_i.FK = x_j.FK$, then $x_i.\mathbf{X}_R = x_j.\mathbf{X}_R$. So, if $x_i.FK = x_j.FK$, the only contributor to the distance is $\mathbf{X}_S$. But in many of the datasets, since $\mathbf{X}_S$ is empty ($d_S = 0$), FK becomes the sole determiner of the distances for *NoJoin*. This is akin to sheer *memorization* of a feature's large domain. Since we operate on features with finite domains, test examples will also have FK from that domain. Thus, memorizing FK does not hurt generalization. While this seems similar to how deep neural networks excel at sheer memorization but still offer good test accuracy [171], the models in our setting are not necessarily memorizing all features but rather only FK. A similar explanation holds for the decision tree. If $\mathbf{X}_S$ is not empty, then it will likely play a major role in the distance

computations and our setting becomes more similar to the traditional single-table learning setting (no FDs).

We now explain why *NoJoin* deviates from *JoinAll* when the tuple ratio is very low for RBF-SVM. Even if $x_i.FK \neq x_j.FK$, it is possible that $x_i.\mathbf{X}_R = x_j.\mathbf{X}_R$. Suppose the "true" distribution is captured by $\mathbf{X}_R$ (as in `OneXr`). If the tuple ratio is very low, there might be many FK values but the number of distinct $\mathbf{X}_R$ values might still be small. In this case, given x_i, RBF-SVM (and 1-NN) is more likely to pick an x_j that minimizes the distances on $\mathbf{X}_R$, thus, potentially yielding lower errors. But since *NoJoin* does not have access to $\mathbf{X}_R$, it can only use $\mathbf{X}_S$ and FK. So, if $\mathbf{X}_S$ is mostly noise, the possibility of the model getting "confused" increases. To see why, if there are few other examples that share $x_i.FK$, matching on $\mathbf{X}_S$ becomes more important. Thus, a non-match on FK becomes more likely, which means a non-match on the implicit $\mathbf{X}_R$ becomes more likely, which in turns makes higher errors more likely. But if there are more examples that share $x_i.FK$, then a match on FK is more likely. Thus, as the tuple ratio increases, the gap between *NoJoin* and *JoinAll* decreases, as Figure 3.3 showed. Internally, RBF-SVM seems more robust to such chance mismatches, since it learns a higher-level relationship between all features compared to 1-NN. Thus, RBF-SVM is more robust to avoiding joins at lower tuples ratios compared to 1-NN.

Finally, the decision tree's internal feature selection and partitioning seems to make it robust to noise from many features. Suppose again the "true" distribution is similar to `OneXr`. Since FK already encodes all information that $\mathbf{X}_R$ provides, the tree almost always uses FK in its partitioning, often multiple times. This is not necessarily "bad" for test accuracy because test examples share $\mathcal{D}_{FK}$. But when the tuple ratio is extremely low, the chance of $\mathbf{X}_S$ "confusing" the tree against the information FK provides goes up, potentially leading to higher errors with *NoJoin*. *JoinAll* escapes such a confusion due to $\mathbf{X}_R$. If $\mathbf{X}_S$ is empty, then FK will almost surely be used for partitioning. But with very few training examples per FK value, the chance of sending it to a wrong partition goes up, leading to higher errors. It turns out that even with just 3 or 4

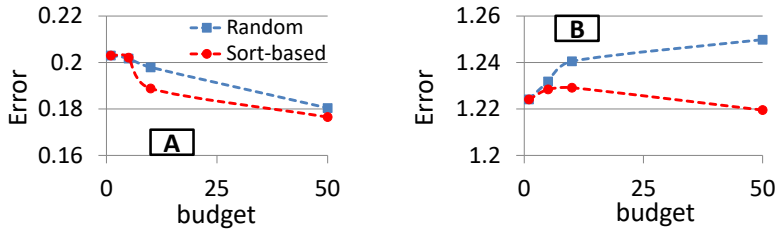


Figure 3.8: Domain compression. (A) *Flights*. (B) *Yelp*.

training examples per FK value, such issues get mitigated. Thus, decision trees seem even more robust to avoiding joins.

3.6 Making FK Features Practical

We now discuss two key practical issues caused by a large $|\mathcal{D}_{FK}|$ and study how standard techniques can be adapted to resolve them. Unlike prior work on handling large-domain regular features [56], foreign key features are distinct, since they have coarser-grained side information available in foreign features, which can be exploited.

3.6.1 Foreign Key Domain Compression

While foreign key features are clearly often useful for *accuracy*, they could make *interpretability* difficult. For example, it is hard to visualize a decision tree that uses a foreign key feature with 1000s of values. Thus, we consider a simple technique from the ML literature to mitigate this issue: *lossy compression*. Essentially, FK with domain $\mathcal{D}_{FK}$ is recoded as $[m]$ (where $m = |\mathcal{D}_{FK}|$). Given a user-specified positive integer "budget" $l \ll m$, we want a mapping $f : [m] \rightarrow [l]$.

A standard unsupervised method to construct f is the *random hashing* trick [161], i.e., randomly map from $[m]$ to $[l]$. We also try a simple supervised method based on filter-based feature selection that we call the *Sort-based* method. It preserves more of the information contained in FK about Y. It is a greedy approach in which we sort $\mathcal{D}_{FK}$ based on $H(Y|FK = z)$, $z \in \mathcal{D}_{FK}$,

compute the differences among adjacent pairs of values, and pick the boundaries corresponding to the top $l-1$ differences (ties broken randomly). This gives us an l-partition of $\mathcal{D}_{FK}$. The intuition is that by grouping FK values that have comparable conditional entropy, $H(Y|f(FK))$ is unlikely to be much higher than $H(Y|FK)$. Note that the lower $H(Y|FK)$ is, the more informative FK is to predict Y.

We empirically compare the above two heuristics using two real datasets for the Gini decision tree with *NoJoin*. Our methodology is as follows. We use the training partition to construct f and then compress FK for the whole dataset. We then use the validation partition and obtain cross-validation errors as before. For random hashing, we report the average across five runs. Figure 3.8 presents the results. On *Yelp*, both *Random* and *Sort-based* have comparable errors although *Sort-based* is marginally higher, especially as l increases. But on *Flights*, the gap is larger for some values of l although the gap narrows as the l increases. The test error with the whole $\mathcal{D}_{FK}$ ($l = m$) for *NoJoin* on *Flights* was 0.14 (see Table 3.2). Thus, it is surprising to see an error of only about 0.18 even with such high domain compression. Even more surprisingly, the test error on *Yelp* goes down after domain compression from 1.31 to about 1.22. Overall, these results suggest that FK domain compression, especially with *Sort-based*, is a promising way to resolve the large-domain issue rather than dropping FK.

3.6.2 Foreign Key Smoothing

Another issue caused by a large $|\mathcal{D}_{FK}|$ is that some FK values might not arise in the train set but arise in the test set or during deployment. This is not the cold start issue, since all FK values are from within the closed $\mathcal{D}_{FK}$, but rather an issue of there not being enough labeled examples to cover all of $\mathcal{D}_{FK}$ well. Typically, this issue is handled using *smoothing*, e.g., Laplacian smoothing for Naive Bayes by adding a pseudocount of 1 to all frequency counts [109]. While similar techniques have been studied for probability estimation using decision trees [118], to the best of our knowledge, this issue has not been handled in general for classification using

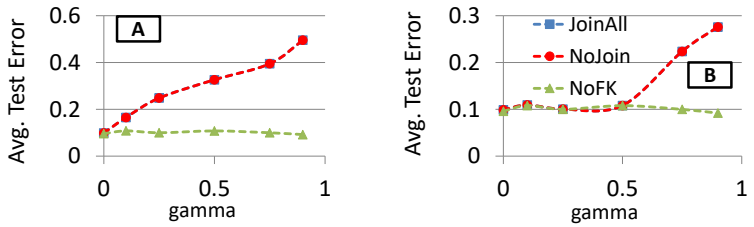


Figure 3.9: Smoothing. (A) Hashing. (B) $\mathbf{X}_R$-based.

decision trees. In fact, popular decision tree packages in R simply crash if this issue arises! Note that SVMs, ANNs, and other numeric feature space-based models do not have this issue, since they use one-hot encoding of FK.

We consider a simple solution approach: smooth by *reassigning* an FK value not seen during training to an FK value that was seen. The reassignment can be done in many ways but for simplicity sake, we consider only two unsupervised methods: *random* reassignment and distances using foreign features ($\mathbf{X}_R$). Note that the latter is only feasible in cases where the dimension tables have been procured; the idea is to use the auxiliary information in $\mathbf{X}_R$ to smooth FK rather than just using *JoinAll*. We smooth using $\mathbf{X}_R$ as follows: given a test example with FK not seen during training, obtain an FK seen during training whose corresponding $\mathbf{X}_R$ feature vector has the minimum distance with the given test example's $\mathbf{X}_R$ (ties broken randomly). The distance measure is just a sum of the l_0 distance for categorical features (count of pairwise mismatches) and l_2 distance for numeric features (Euclidean distance).

The intuition for $\mathbf{X}_R$-based smoothing is that if $\mathbf{X}_R$ is part of the "true" distribution, it may yield lower errors than random smoothing, but if $\mathbf{X}_R$ is just noise, both methods become similar. We empirically compare these methods using the `OneXr` simulation scenario in which a lone feature $X_r \in \mathbf{X}_R$ determines the target (with some Bayes error). We introduce a parameter γ that is the ratio of the number of FK values not seen during training to $|\mathcal{D}_{FK}|$. If $\gamma = 0$, smoothing is not needed; as γ increases, more smoothing is needed. Figure 3.9 presents the results. We see that $\mathbf{X}_R$-based smoothing yields much lower test errors for both *NoJoin* and *JoinAll*. In fact, the smoothed approaches' errors are comparable to *NoFK* and the Bayes error for low values of γ

(< 0.5). As γ gets closer to 1, the errors of $\mathbf{X}_R$-based smoothing also increase but not as much as random smoothing. Overall, these results suggest that one could get "the best of both worlds" in a way: even if foreign features are available, rather for using them always as in *JoinAll*, an often viable alternative is to use them as side information for smoothing foreign key features with *NoJoin*, thus still yielding some of the runtime and usability benefits of *NoJoin*.

3.6.3 Discussion and Limitations

Our results confirm that it is often safe to avoid KFK joins even for popular high-capacity classifiers. Thus, data scientists can use the tuple ratio rule to easily reduce the burden of data sourcing for such classifiers too, not just linear models. We also showed that it is possible to avoid joins safely regardless of whether features are categorical or numeric. This has a new implication for further theoretical analysis of our results because the analysis in [100] relied on the finiteness of the hypothesis space due to all features being categorical. But an infinite hypothesis space does not preclude a finite VC dimension [139]. Extending the theoretical analysis to our more general setting is an open problem. While we focused on star schemas, our results can be easily extended to snowflake schemas as well due to the transitivity of FDs. Our results also apply to single-table data with an acyclic set of FDs, as noted in [100], since a BCNF decomposition can yield a multi-table scenario.

We recap the limitations and assumptions of our work to help data scientists apply our idea in the right context. We focused only on popular classification models but our results hold for both binary and multi-class targets and both categorical and numeric features. If a foreign key is not generalizable (e.g., search ID in *Expedia*), it cannot be used directly as a feature and so, its corresponding join should not be avoided. Finally, we leave it to future work to study the interplay of our work with cold start techniques and latency trade-offs during model serving.

3.7 Conclusion

It is high time for the data management community to look beyond just building faster ML systems and help reduce the pains of data sourcing for ML. Understanding how fundamental data properties and schema information can simplify end-to-end ML workflows is one promising avenue in this direction. While the idea of avoiding joins safely has been adopted in practice for linear classifiers, in this comprehensive study, we show that it works as well or better for popular high-capacity classifiers too. This goes against the intuition that high-capacity classifiers are typically more prone to overfitting. We hope that our work spurs discussions and new research on simplifying data acquisition for ML.

Chapter 3 contains material from "Are key-foreign key joins safe to avoid when learning high-capacity classifiers?" by Vraj Shah, Arun Kumar, and Xiaojin Zhu, which appeared in Proceedings of the 2018 Very Large Data Bases Conference (VLDB'18). The dissertation author was the primary investigator and author of this paper.

Chapter 4

ML Data Prep Zoo: Towards Automating and Benchmarking ML Data Prep

4.1 Introduction

In this chapter, we dive deeper into our vision of a new line of community-driven research towards automating and benchmarking ML data prep. Surveys of data scientists show that ML data prep often dominates their time and effort, even up to 80% [152]. It is tedious grunt work involving tasks such as identifying feature types and extracting feature values. Today, it is performed mostly manually in tools like Python and R, reducing data scientists' productivity and raising costs. Modern datasets also often have 1000s of columns, worseing this issue. Furthermore, SalesForce, Google, and other cloud vendors are starting to offer end-to-end AutoML platforms for enterprises; manual data prep at this scale of millions of datasets is untenable [13].

Challenge: Semantic Gap. While the DB community has long studied data cleaning/prep for SQL analytics, little work has studied the peculiarities of ML data prep. The *semantic gap* between what an *attribute* is in a DB/data file and what a *feature* is for ML means many tasks have fallen through the cracks. Thus, a pressing grand challenge for the community is to construct

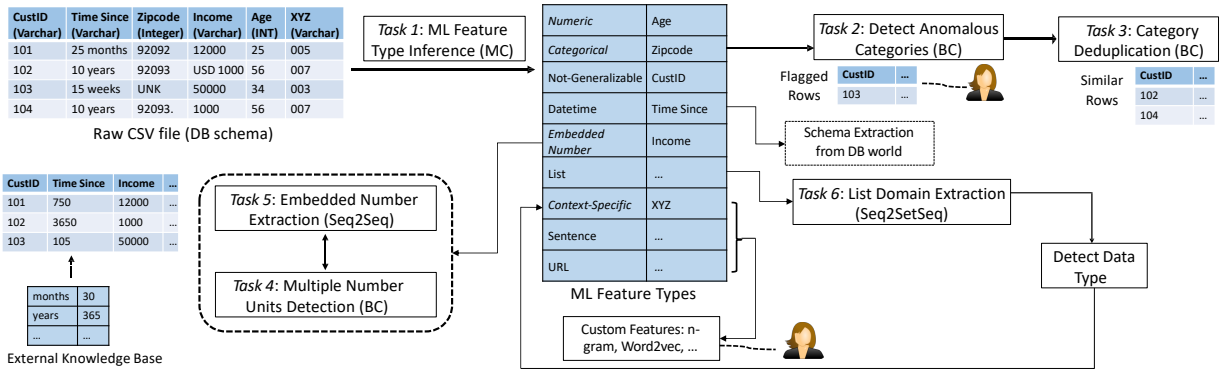


Figure 4.1: Illustrating major data prep tasks. The user loads a customers table to train, say, a churn predictor. BC stands for binary classification. MC stands for multi-class classification. Seq2Seq stands for sequence-to-sequence learning. Seq2SetSeq stands for sequence-to-set-of-sequence learning.

a shared understanding/terminology of such tasks, understand why they are hard to automate, and standardize evaluation of automated tools.

Our Vision. To meet the above challenge, we envision a community-driven effort for automating ML data prep. Our philosophy is to *abstract specific ML data prep tasks and cast them as applied ML tasks*. This raises 3 questions. What are the tasks and what is their role? How to cast them as applied ML tasks? How to create benchmark datasets for comparing tools? In particular, we believe the critical limiting factor for impactful and replicable research in this space is not fancier algorithms or theory but the *availability of large high-quality labeled datasets* for ML data prep tasks. As an analogy, the formalization of the ImageNet task and dataset spurred major recent advances in ML-based vision.

We now present our vision of the *ML Data Prep Zoo*, a repository of common ML data prep task definitions, benchmark labeled datasets, and pre-trained ML models. Figure 4.1 illustrates 6 tasks we have defined so far based on our conversations with data scientists. We next explain these tasks and how to cast them as applied ML tasks. We also discuss key research questions in realizing this vision and explain the different choices. Finally, we describe the ML Data Prep Zoo repository for our datasets and models and announce competitions for community contributions.

4.2 Data Prep Tasks for ML

4.2.1 Task 1: ML Feature Type Inference

Description and Example. The first step is to infer the ML feature types. Most ML models recognize only *numeric* or *categorical* features. This task is surprisingly hard to automate accurately due to the *semantic gap* between DB and ML schemas. For example, `Zipcode` in Figure 4.1 is an integer; so, Pandas will call it a numeric feature, which is nonsensical! This semantic gap is bridged today manually by converting it to categorical. This issue is common in real datasets, since categories are often stored as integers, e.g., disease codes, product types, etc. Real datasets also often have 100s of features, which means the manual grunt work quickly adds up.

Casting as an ML Task. We bridge the semantic gap by casting this task as an ML classification task. We explain this further in Chapter 5. The raw features are a whole column, including name such as "ZipCode" and sample values such as 90292, 92093, etc. in the above example. Two classes are *numeric* and *categorical*. But there is often not enough information in the data file to identify a column type, even for humans. This necessitates more classes; we created a 9-class vocabulary. (1) *Numeric* and (2) *Categorical*: These are for columns that can be (almost) directly used for the target model, e.g., `Age` in Figure 4.1 is *Numeric*, while *Zipcode* is *Categorical*. (3) *Usable-with-Extraction*: These include types such as Datetime, Sentence, Embedded Number, URL, and List. Such columns have "messy" values, preventing direct use as numeric or categorical features, e.g., `Income` and `TimeSince` require custom extraction before being used as numeric features. Such extraction is hard to automate fully, but we later discuss a few common extraction tasks that can be cast as applied ML tasks. (4) *Not-Generalizable*: Such columns can *not* be used as features for the target model because they are not "generalizable," e.g., `CustID` is a primary key. (5) *Context-Specific*: This is a catch-all for columns whose type is hard to tell even for humans, e.g., `XYZ` has integers but is it really numeric (like `Age`) or categorical

(like `Zipcode`)? To ascertain the type of such columns, data scientists typically need to manually check the application's data documentation.

4.2.2 Tasks for Categorical

While a *Categorical* column can be used directly, data scientists often seek to resolve two issues with its domain to boost target model accuracy: *missing value categories* and *duplicate categories*. For instance, we saw both"-999" and "unknown" for missing values and both "CA" and "California" for California in real datasets. One may want to discard missing value categories and instead use statistical techniques for handling missing values. One may also want to deduplicate categories to reduce domain size, which helps in the bias-variance tradeoff. Thus, we formalize two new data prep tasks as binary classification: **Task 2: Detect Anomalous Categories** to flag missing value categories and **Task 3: Category Deduplication** to flag pairs of categories that are duplicates. The column name and its domain are the raw features for both tasks. Task 2 is an instance of the entity matching problem in the data cleaning literature but with much less metadata for devising similarity scores; one could consider Siamese neural networks for this task.

4.2.3 Tasks for Usable-with-Extraction

Usable-with-Extraction columns require more processing to extract numeric and/or categorical features, e.g., `Income`. Figure 4.1 has "USD" prefixing a number, while `TimeSince` has "months," "years," etc. suffixing numbers. Data scientists often write regular expressions or custom code to extract such values. While it is perhaps impossible to automate all such extractions, we identify three common tasks that can be cast as applied ML tasks.

Task 4: Multiple Number Units Detection: Are the units of an embedded number the same? If not, we need to standardize the units, likely with human intervention and/or external

knowledge bases about units. In Figure 4.1, `TimeSince` has multiple units. If yes, we get **Task 5: Embedded Number Extraction**: What is the embedded number? For instance, extract 1000 from "USD 1000." This can be seen as both a Seq2Seq task and a sequence-to-regression task. An encoder-decoder CNN/RNN may fit this task. One could also consider joint multi-task learning for Tasks 4 and 5.

Task 6: List Domain Extraction: Some columns have lists in a string separated by commas, space, semicolons, etc. Data scientists typically write custom code to *extract the domain* of the list values and use the domain to get new numeric/categorical features for the target model. This is a complex task that converts a sequence to a *set of sequences* representing domain entries. Once could consider more complex neural architectures for this task.

Other Featurization Routines In our current scope, we leave other standard featurization routines for custom processing to the user. For instance, to process a full English sentence in a *Usable-with-Extraction* column, data scientists may want to use bag-of-words, n-grams, or embeddings like Word2Vec or Doc2Vec. Such feature engineering decisions are orthogonal to our focus and are often application-specific. Other feature types such as dates and timestamps can be processed using standard DB techniques, while URLs and custom objects may require human intervention. One could consider character-level CNNs and RNNs for this task.

4.3 Research Questions and Options

4.3.1 Metrics and Featurization

The accuracy metrics for Tasks 1 to 4 are standard, but for Tasks 5 and 6, we may need to define new metrics. For Task 5, edit distance and/or squared loss are candidates with differing results, e.g., "12" is closer to "$12.99" under the latter but not the former although edit distance helps sequence extractors. Task 6 has a complex structured prediction output, which may need

a complex loss function (ideally, still differentiable) and multiple accuracy metrics. Even the featurization of the raw column is an open question, since the ML models for our tasks also need numeric, categorical, or string features. Several options exist: obtain n-grams or embeddings from column names and sample values, get summary statistics, and so on. Characterizing which of these features matter the most is also part of our research, since such featurization matters for both accuracy and inference latency at deployment time.

4.3.2 Creating Large Labeled Datasets

This is our central research challenge. To the best of our knowledge, there are no large benchmark labeled datasets for any of our 6 data prep tasks. Manual labeling for each task each could yield best accuracy but it is highly time-consuming and expensive. There are three alternative approaches: crowdsourcing, active learning, and weak supervision.

Crowdsourcing labels is common in ML practice, but we face a major quality issue: most crowd workers are lay users, not data scientists who "get" data prep. In fact, our pilot run for crowdsourcing labels for Task 1 on the FigureEight platform resulted in too much noise even with 5 labels per example. Thus, how to structure crowd labeling questions better is an open research question. Active learning with a data scientist in the loop is another option. But a key disadvantage here is that we need to fix the task's ML model beforehand. Finally, weak supervision is a promising approach here, since it is often possible to write small labeling functions (LFs) to encode structural heuristics and dictionary lookups for some tasks. A denoising framework like Snorkel [122] can potentially help boost accuracy over the LFs' outputs. Snuba-on-Snorkel [158] can automate the production of LFs for some classification tasks. But an open challenge is that Snorkel current does not support complex prediction outputs like in Tasks 5 and 6.

4.3.3 Creating Human-in-the-loop Tools

Our work cannot end at getting ML models for our tasks. To complete the loop,we need to integrate them for inference in popular data prep ecosystems. There are two main kinds of tools: programmatic (e.g., R, Pandas, and TFDV) and visual (e.g., Excel and Trifacta). Each presents its own set of interesting implementation challenges. For the former, simple APIs can be introduced to plug our trained ML models. For the latter, it is an open research question as to how to create appropriate interface mechanisms that can exploit both our ML models' predictions and human-in-the-loop correction capabilities. For instance, the user could "guide" an ensemble of ML models based on column semantics or specific column values they see. Looking even further out, we can integrate ML models with programming-by-example and program synthesis approaches, especially for Tasks 5 and 6 that require value extraction. This requires resolving ambiguity in the program search space and defining new ranking schemes aimed at reducing manual extraction effort.

4.4 Conclusion

We announce the ML Data Prep Zoo, a living public repository (on GitHub) of labeled data for ML data prep tasks [36]. We release all datasets we create as CSV files. We also release our trained ML models in Python for the defined tasks. Our first release is for Task 1 and Task 3. Our trained models include logistic regression, Random Forest, kernel SVM, and a character-level CNN for Task 1. The Zoo also tabulate the accuracy of the baselines and our models on each task. We invite contributions from the research community to augment these datasets, create new data for the other tasks, and/or define new tasks along with their own labeled data and models. We have a leaderboard for public competitions on the hosted datasets with multiple accuracy and runtime metrics, inspired by the ImageNet competition. We invite researchers to use our data to create better featurization and models to automate ML data prep tasks.

Chapter 4 contains material from "The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML" by Vraj Shah and Arun Kumar, which appeared in Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning (DEEM'19). The dissertation author was the primary investigator and author of this paper.

Chapter 5

ML Feature Type Inference: Benchmarking and Automating for ML

5.1 Introduction

In this chapter, we initiate the first work on benchmarking and automating a critical ML data prep task, ML feature type inference. The paradigm of automated machine learning (AutoML) is beginning to help democratize machine learning for the masses [80]. Cloud vendors have released AutoML platforms such as Google's Cloud AutoML [10] and Salesforce's Einstein [13] that build ML models on millions of datasets from thousands of small-and-medium enterprises automatically. The central goal of these platforms is to get an accurate model for the prediction task while achieving maximum possible automation of the end-to-end ML workflow, especially on structured data, including data transformations and feature engineering, as well as model building and hyperparameter tuning. The automation of these steps has been intensively studied in the ML/data mining [80, 68] and database communities [83, 97]. However, a crucial gateway step to this whole workflow has received much less attention so far: *ML feature type inference*.

Datasets are typically loaded as files into the AutoML platforms. As Figure 5.1 illustrates,

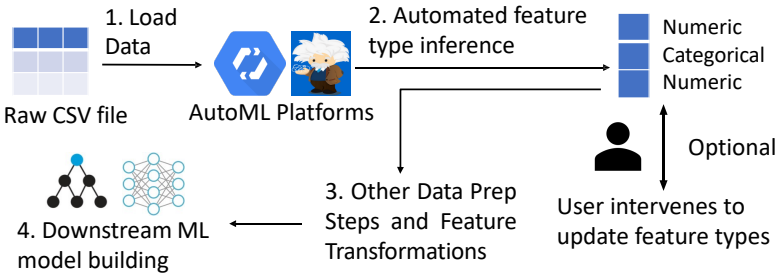


Figure 5.1: Typical workflow in AutoML platforms.

ML feature type inference is the very first step needed for ML over structured data. Features could be *Numeric*, *Categorical*, or something else, as shown in Figure 5.1. Determining the correct feature type is crucial for the whole workflow to work well: what data transformations to apply, how to extract features, and how to feed signals to the downstream models. For instance, if a column is inferred to be of type Timestamp, then several useful features such as day, month, and year are often extracted automatically for the downstream model. Thus, the accuracy of feature type inference is critical for the downstream model's accuracy, and in turn, the effectiveness of the entire ML platform.

Feature type inference is also performed automatically by many ML platforms, e.g. TransmogrifAI in Einstein [17], Tensorflow Data Validation (TFDV) in TensorFlow Extended [45], and AutoGluon from AWS [62]. But surprisingly, there is no objective evaluation to date of how good their automation of this task is. Thus, mistakes in their automated feature type inference can propagate and may degrade the workflow. For instance, consider what TFDV does on the illustrative dataset for a common ML task, customer churn prediction in Table 5.1. It wrongly calls many *Categorical* features with integer values as *Numeric*, e.g., *ZipCode*. This can cause the downstream model to produce garbage results. Moreover, *Income* is inferred as *Categorical* even though it has numbers embedded. Such issues can lead to loss of information and can potentially reduce the accuracy of the model, or even cause it to fail in some scenarios.

One might ask: *Why cannot AutoML platform users manually verify their feature types?* From our conversations with AutoML platform engineers at Salesforce and Google, we learned

that their AutoML tools are used on over tens of thousands of datasets, adding up to millions of features in production settings. Forcing users to manually annotate features can lead to a tedious, slow, and error-prone process that also *violates the promise of automation.* Many domain users who may not have much ML expertise may not like the platform asking them to manually mark ML feature types. Thus, AutoML platform engineers prefer ever more accurate automation of this task. Clearly, this requires them to objectively measure the accuracy of their AutoML tool on the given task.

5.1.1 This Work's Focus

Our Focus. We initiate work on benchmarking and objectively quantifying the task of *ML feature type inference* in existing open-source industrial-strength AutoML tools. We formalize and standardize this task by creating a benchmark labeled dataset. This will enable an objective progress measurement, akin to ImageNet's role in vision [127]. Moreover, this will help objectively evaluate and improve AutoML platforms by enabling answers to key questions: *How good are AutoML tools? How can one do better? How does the accuracy of type inference affect downstream ML model's accuracy?*

Challenge. We first explain why feature type inference is hard to automate for existing rule-based or syntax-based systems. Datasets are typically loaded from RDBMSs, data lakes, or filesystems as flat CSV files into AutoML platforms. Thus, there exists a *semantic gap between feature types for ML and attribute types* in databases/files. The latter tells us the syntactic datatypes of columns such as integer, real, or string. This semantic gap means reading syntax as semantics often leads to nonsensical results. For instance, consider Table 5.1 again. Attributes such as *CustID*, *Salary*, and *ZipCode* are stored as integers, but only *Salary* is useful as *Numeric*. *CustID* is unique for every customer, hence it can not be generalized for ML. *ZipCode* is *Categorical*, even though it is stored as integers. In fact, this issue is ubiquitous in real-world datasets, since categories are often encoded as integers, e.g., item code, state code, etc.

Table 5.1: A simplified *Customers* data for churn prediction.

CustID	*Gender*	*Salary*	*ZipCode*	*XYZ*	*Income*	*HireDate*	*Churn*
1501	'F'	1500	92092	005	'USD 15000'	'05/01/1992'	'Yes'
1704	'M'	3400	78712	003	'25384'	'12/09/2008'	'No'

Scope. Our focus is on relational/tabular data, which can be stored in any format (CSV, JSON, XML, etc.) and with any filesystems. Note that our focus is not to study any upstream processing steps that users might perform when they load their files into the AutoML tool. Also, our focus is not on feature engineering and transformation steps over the columns with the inferred types. We focus only on the ML feature type inference step. Admittedly, this is just one step in the entire end-to-end ML workflow, but we believe that studying this step in depth is critical to improve existing AutoML platforms, as we find that accurate type inference is critical for achieving high downstream model accuracy. Equally importantly, the predictions are more *interpretable* with accurate feature types.

5.1.2 Benchmark Comparisons

Our Labeled Dataset and Label Vocabulary. Creating labeled data for the task requires a common formalized label vocabulary, which is important to create because the dichotomy of *Numeric* vs. *Categorical* is not usually enough for categorizing feature types of raw columns. For instance, column *HireDate* in Table 5.1 stores *Date* type values. Thus, we need more classes. We survey existing AutoML data prep tools and collect their feature type vocabulary into a common, practically useful set of labels that can be reused by any AutoML platform, as Figure 5.2 shows. *We gather and hand-label the very first large meta-dataset for benchmarking feature type inference*. Our dataset has 9921 columns from 1240 real data files from sources such as Kaggle and UCI ML repository. Our labeling process took about 90 man-hours across 5 months.

Current Limitation. We admit that files on Kaggle and UCI ML repository may not be representative of the truly "in-the-wild" dataset as it may have undergone some pre-processing. But, it is impractical for researchers to get access to large numbers of publicly releasable data

from enterprises and organizations due to legal restrictions. Thus, Kaggle and UCI are the closest sources we have to the real-world data. We believe that our exploratory work is the first step in the direction of objectively evaluating AutoML tools. We hope that this work starts a conversation around enhancing such benchmark datasets.

Approaches to Type Inference. There are open-source tools such as Pandas [107], TransmogrifAI [17], TFDV [45], and AutoGluon [62] that automate this task. They all happen to be either rule-based or syntax-based. In contrast to prior approaches, our labeled dataset also presents an alternative approach to type inference: use ML itself to automate this task. We cast ML feature type inference as a multi-class classification problem and use ML models to bridge the semantic gap. We extract signals from raw data files that a typical data scientist may look at to identify the feature type. We summarize the signals in a feature set, which we use to build standard ML models on our labeled data. We empirically compare the ML-based approach enabled by our labeled data and existing public tools on our labeled *test* dataset.

Semantic Type Detection Tools. Recent tools such as Sherlock [79] and AutoType [167] perform column-level semantic type detection for automated data discovery and cleaning. The semantic type vocabulary of these tools is not directly usable for the AutoML setting because a semantic type can belong to multiple ML feature types. This is by design because the application motivations are different: semantic type detection tools are aimed at Business Intelligence (BI) tool users to browse attributes more easily, not AutoML users. Thus, it is complementary to our focus. To understand whether such tools can be ported to the AutoML setting, we use a rule-based approach to map Sherlock's semantic types to our vocabulary and evaluate it on our dataset.

Downstream Benchmark Suite. To understand the impact of the accuracy of ML feature type inference task on the downstream models, we create a *downstream benchmark*: 30 curated real-world datasets containing classification and regression tasks from diverse application domains such as healthcare, retail, sports, etc. The benchmark enables us to answer two key questions: (1) How does wrong type inference affect downstream performance? (2) How accurate are the

downstream models delivered by the prior tools and the ML-based approach using our labeled data relative to performance with true feature types?

Empirical Evaluation and Analysis. An empirical comparison of different approaches on our labeled data shows that the ML-based approach delivers a lift of an average 14% and up to 38% in accuracy compared to existing tools for identifying feature types. We then evaluate and compare different ML models on our dataset. Overall, Random Forest outperforms the other models and achieves the best 9-class accuracy of 92.6%. We perform an ablation study on our ML models to characterize what types of features are useful.

Our empirical evaluation on the downstream benchmark suite shows that an ML-based approach using our labeled data delivers the most accurate downstream model against the prior tools for 47 out of 60 downstream models. In addition, we find that the wrong types inferred by existing tools often lead to a significant decrease in the downstream model's accuracy relative to their true accuracy. For instance, Pandas underperforms over truth in 45 out of 60 cases. Finally, we release a repository containing our labeled dataset, trained ML models, downstream benchmarks, and announce a leaderboard for community contributions.

In summary, this work makes four key contributions.

1. **A new benchmark task and dataset.** To the best of our knowledge, this is the first work to formalize and rigorously benchmark the task of ML feature type inference. We create the first large benchmark labeled datasets for this task with a readily practically useful 9-class label vocabulary.

2. **Benchmarking alternate tools and approaches.** Using our new data, we perform extensive empirical comparisons of open source and industrial (Auto)ML tools. Perhaps surprisingly, we find that even off-the-shelf ML models with standard featurization trained on our data significantly outperform all prior approaches.

3. **Downstream benchmark suite.** The curated benchmark offers evidence that the down-

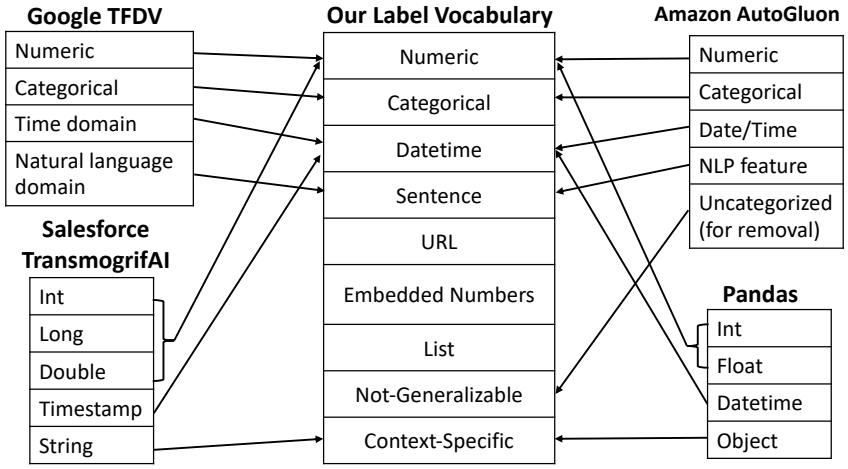


Figure 5.2: Feature type vocabulary mapping of TFDV, Pandas, TransmogrifAI, and AutoGluon to our vocabulary

stream model's performance can benefit by accurately determining feature types. We find that an ML model trained on our data for feature type inference often leads to more accurate downstream models than prior tools.

4. **Real-world impact.** Google collaborated with us to integrate our best performing ML models into TFDV to improve its inference of *Categorical* [137]. Google engineers are now reviewing it on internal benchmarks for adoption. AWS and OpenML [155] have also expressed interest in adopting our data and models for production use. Also, we release a public competition on our labeled dataset to invite contributions to create/augment datasets, better featurizations, and models.

5.2 Our Dataset

This section discusses our efforts in creating the labeled dataset. We discuss how we design the label vocabulary, the data sources, the signals we extract from the columns that enable us to inspect the columns succinctly, and the labelling process.

5.2.1 Label Vocabulary

Most ML models ultimately operate over only 2 (final) feature types: *Numeric* (continuous set) and *Categorical* (discrete set). Thus, each example (or column) has to be labelled as either of the two classes. However, we find that this bifurcation is not enough. This is because many other column types such as *Date*, *URL*, and *Primary Keys* are inevitable in the raw data file. Moreover, we find that the data file may not contain enough information to determine the feature type of a column, even for humans, e.g., column *XYZ* in Table 5.1. Thus, we need more classes. We surveyed how the existing open source data prep tools such as Google's TFDV [45], TransmogrifAI in Salesforce Einstein [17], and AutoGluon from Amazon AWS [62] approach type inference and perform type-specific feature transformations. Figure 5.2 shows the feature type vocabulary of these tools. Inspired by this, we distill a common and practically useful set of labels for our vocabulary. We discuss the labels below.

(1) Numeric. These attributes are quantitative in nature and can directly be utilized as a *Numeric* feature for the downstream ML model. For instance, *Salary* is *Numeric*, while ID attributes such as *CustID* or integers representing encodings of discrete levels are not.

(2) Categorical. These attributes contain qualitative values that can directly be utilized as *Categorical* features for the downstream ML model. There are two major sub-classes: nominal and ordinal. Ordinal features have a notion of ordering among its values, while nominal do not. For instance, *Year* is ordinal, while *ZipCode* is nominal. Names and coded real-world entities from a known finite domain set are also *Categorical*. One often needs to alter the syntax of *Categorical* features for the downstream model, e.g., one-hot encoding in Scikit-learn or explicitly cast as a "factor" variable in R.

(3) Datetime. This class represents attributes containing date or timestamp values, e.g., *"7/11/2018"*, and *"21hrs:15min:3sec."* One may choose to extract custom features, either *Numeric* or *Categorical* or both through standard featurization routines. For instance, the month of the year can be *Categorical*, while time can be *Numeric.* Note that, such feature engineering decisions are

not focus of this work since they are typically application-specific.

(4) Sentence. This class represents attributes containing textual values with semantic meaning. For instance, a passage of text may provide rich semantic information for a sentiment analysis application. One may choose to extract custom features, either *Numeric* or *Categorical*, or both through standard featurization routines. For instance, the AutoML platform developer can route such columns to an *n*-gram featurization routine or a routine to get Word2Vec embeddings from an English sentence for the downstream model. Again, we leave such downstream feature engineering decisions that come after type inference to the AutoML platform developer.

(5) URL. This class is for attributes whose values follow the URL standards [38]. This requires that the attribute values begin with a protocol followed by a sub-domain and a domain name. Any following information such as a file path is optional.

(6) Embedded Number. This class denotes attributes with "messy" syntax that preclude their direct use as *Numeric* or *Categorical* features. Thus, they require some form of processing before being used as as features. For instance, a number may be present along with string(s) denoting a measurement unit (*"30 Mhz" or "USD 45"*) and/or special characters (*"5,00,000"*). In all cases, a number is typically extracted and the units are standardized (if applicable). One would typically use regular expressions or custom Python/R scripts for such extraction, e.g., converting *"USD 45"* to 45.

(7) List. These attributes contain a list of items separated by a delimiter. One may write custom scripts to extract the domain of the list values and get new features for the downstream model.

(8) Not-Generalizable. An attribute in this class is a primary key in the table or has (almost) no informative values to be useful as a feature. Similarly, a column with only one unique value in the whole table offers no discriminative power and is thus useless. Such attributes are most unlikely to be used as features for the downstream model because they are not "generalizable." For example, *CustID* belongs to this class, since every future customer will have a new *CustID*. It

is quite unlikely that one can get any useful features from it. Note that an attribute categorized as *Not-Generalizable* does not mean that it can never be useful for the downstream model. One may obtain some features from such attributes through more custom processing or domain knowledge. In contrast, even though attributes such as *Income* and *Date* may have all unique values in their columns, they are still generalizable. Thus, they belong to *Embedded Numbers* and *Datetime* respectively since it is highly likely that one can extract useful features from them.

(9) Context-Specific. This class is a catch-all for attributes that require human intervention either to determine their feature types and/or to inspect their values to build custom featurization routines. The following examples illustrate this class. (1) Attributes wherein the data file does not have enough information even for a human to judge its feature type. Such columns typically have meaningless names, e.g., *XYZ* in Table 5.1. Judging the feature type would require manually tracing down the provenance of how this column came to be using external "data dictionaries" maintained by the application or speaking to the data creator. (2) Attributes whose values require manual inspection for extracting useful features, e.g., JSON objects, geo-locations, addresses, or other complex objects that contain information dump about the data.

Our 9-class label vocabulary, while limited, is already practically useful for AutoML platforms. The label vocabulary can also give other insights to an AutoML platform developer. For instance, they could look for tables to join when faced with a large-domain *Categorical* feature such as *ZipCode*. They could route attributes marked as *Embedded Numbers* or *Datetime* to suitable Python/R scripts. Moreover, they could dispatch the columns that are marked *Not-Generalizable* for any missing values or errors in data entry to appropriate libraries. Finally, they could prompt for user intervention on only the columns that are marked *Context-Specific*. This can reduce user time spent on annotation significantly.

5.2.2 Data Sources

We gather 1240 CSV data files from sources such as Kaggle and UCI ML repository. Each column of the CSV file is just one example for our task. We obtain 9921 examples from all data files. Note that we do not always use all the columns from a single data file for labeling. We explain this in Section 5.2.4. Kaggle and UCI ML are the largest public data sources that are closest to real-world datasets. However, we note a caveat that the files on Kaggle and UCI ML may have undergone some pre-processing. It is almost impossible for researchers to get access to large numbers of truly "in-the-wild" data from enterprises and other organizations and make them publicly available due to legal restrictions. But the crux of our point in this work is this: even on data files from Kaggle and UCI, existing open-source and industrial tools yield relatively poor accuracy compared to the ML models trained on our data (Section 5.4.2). Thus, we believe our work is a promising start towards objectively evaluating AutoML platforms.

5.2.3 Base Featurization

To identify the feature type of a raw column, a human data scientist may look at the column name, some sample values in the column, and even descriptive stats about the column. For instance, just by reading the attribute name, *ZipCode*, an interpretable string, a human can tell its feature type is *Categorical*. Thus, we represent the columns in a more concise way such that it emulates what a typical data scientist may look at to determine the feature type. We call this step Base Featurization. We extract the following base features for every column in the raw data file.

(1) Column name. We extract the column name as it can give crucial semantic clues for the feature type.

(2) Column values. A human would typically inspect some values in the column to make sure they make sense. For instance, values with decimal points are likely to mean *Numeric* features, while values with delimiters are likely lists. Thus, we extract 5 randomly sampled

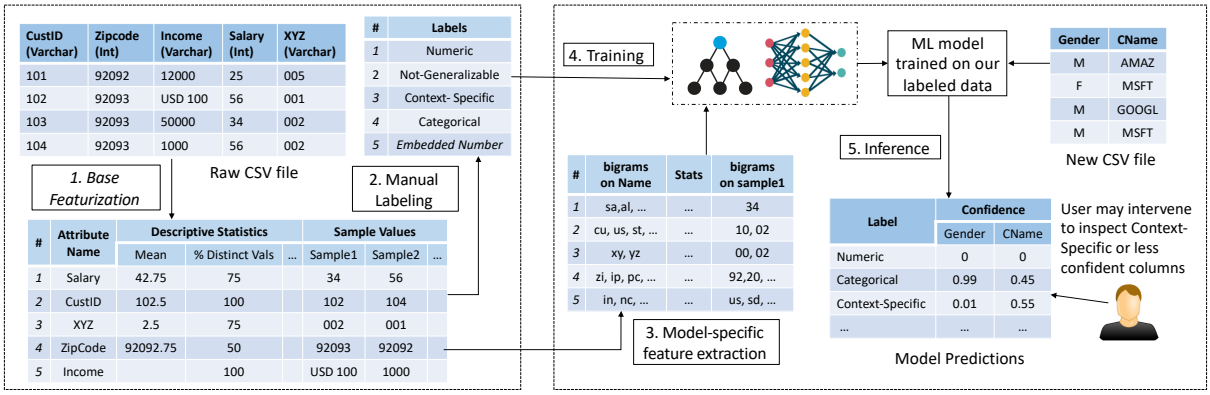


Figure 5.3: Workflow showing our labeling process and how our data is used for ML-based feature type inference.

distinct attribute values from the column. We choose 5 because we think it is a reasonable number for a human to understand the column and determine the feature type when doing manual labeling (Section 5.2.4). However, this number can very well be higher or lower. It can be even tuned when building an ML model or a heuristic. In fact, from the ablation study of the ML models built on the base features, we find that even one or two sample values may be good enough to build an accurate model (Table 5.4).

(3) Descriptive statistics. Finally, a human would look at some descriptive stats about the column. For instance, if the human finds that all values in the column are NaNs, then they might classify the column as *Not-Generalizable*. Considering this, we extract 25 descriptive stats for a column such as the total number of values, the absolute number and % of NaNs relative to total values, the absolute number and % of distinct values relative to total values, mean, and standard deviation. We provide the complete list of these features in Table 5.2.

Each column in the raw data file is an example in the new base featurized file and we manually label every example of the base featurized file. The base featurization step also helps to deliver an ML-based approach to type inference.

5.2.4 Labelling Process

We first use base featurized columns from 360 source files to label them in one of the nine classes. But, we find that they only contain a small handful of examples for the classes: *URL*, *List*, *Sentence*, *Embedded Number*, and *Datetime*. Thus, we use an additional 880 source data files to only label the examples for the under-represented classes. We extract these examples from additional sources as we did not want to create a heavily skewed class label distribution to get good confidence on all classes. Note that augmenting classes where the number of examples is under-represented is a common practice in the ML literature [124, 48, 51, 144]. Since our benchmark contains multiple class-level accuracy metrics (discussed in Section 5.4.1), inspecting them can provide more confidence with the class predictions. Furthermore, we find that many data files have a series of column names such as *xyz1*, *xyz2*, and so on. Thus, we drop the columns with a repeating series of names.

To reduce the cognitive load of labelling, we follow the following process. Initially, we manually label 500 examples. We then use Random Forest with 100 estimators to perform 5-fold nested cross-validation (CV). The model achieves a classification accuracy of around 74% on the test set (average across 5 folds). We use this model to predict a class label on all of the 9921 examples. We then group all the examples by these predicted labels and inspect all of them manually. Such grouping helps reduce the cognitive load caused by class context switches during labeling. *The labeling process took about 90 man-hours across 5 months.*

We tried to crowdsource labels for our dataset on the FigureEight platform but abandoned this effort because the label quality was too low across two trial runs. In our pilot run, we used a concise label vocabulary with 5 classes: *Numeric*, *Categorical*, *Needs-Extraction*, *Not-Generalizable*, and *Context-Specific*. *Needs-Extraction* includes the classes: *Datetime*, *Sentence*, *URL*, *Embedded Number*, and *List*. In the first run, we got 5 workers each for 100 examples; in the second, 7 each for 415. The "golden" dataset were the 500 examples we labeled manually. We listed several rules and guidelines and provided many examples for worker training. But in

Table 5.2: List of descriptive statistics features

Descriptive Stats
Total number of values
Number of nans and % of nans
Number of unique values and % of unique values
Mean and std deviation of the column values, word count, stopword count, char count, whitespace count, and delimiter count
Min and max value of the column
Regular expression check for the presence of url, email, sequence of delimiters, and list on the 5 sample values
Pandas timestamp check on 5 sample values

the end, we found the results too noisy to be useful: in the first run, 4% of examples had 4 unique labels, 27% had 3, and 69% had 2; in the second run, these were 5%, 21%, and 49%. Majority voting gave the wrong answer in half of the examples we randomly checked.

5.2.5 Data Statistics

The distribution of class labels in our labeled dataset is: *Numeric* (36.6%), *Categorical* (23.3%), *Datetime* (7%), *Sentence* (3.9%), *URL* (1.5%), *Embedded Number* (5.7%), *List* (2.4%), *Not-Generalizable* (10.6%), and *Context-Specific* (8.9%). We provide a complete breakdown of the cumulative distribution by class for different descriptive statistics in the technical report [136]. We observe that attribute values for *Sentence*, *URL*, and *List* have more characters and words than other classes. Also, all *Numeric* sample values and 80% of the *Categorical* sample values are single token strings. Furthermore, we find that almost 90% of the *Categorical* attributes have less than 1% unique values in its columns. Interestingly, 54% of *Not-Generalizable* have either one unique value or only NaN values in their domain. Table 5.2 present all the descriptive stats used for base featurization.

5.3 Approaches Compared

In this section, we discuss the different approaches to type inference. We first discuss existing open-source tools that all happen to be either rule-based or syntax-based. We then briefly

discuss an intuitive rule-based baseline to check if a set of rules can accurately capture our labeled dataset. Finally, we explain how our labeled dataset is used to build ML models.

5.3.1 Existing Tools

Figure 5.2 shows the feature type vocabulary of these tools and how they map to our label vocabulary.

Tensorflow Data Validation (TFDV). TFDV is a tool to analyze and transform ML data in TensorFlow Extended (TFX) pipeline [45]. TFDV uses heuristics to infer ML feature types such as numeric, categorical, time or date domain, or natural language text from the descriptive statistics about the column. The users can then review the inferred feature types and can update them manually.

Pandas. Pandas is a Python library that provides tools for data analysis and data transformations. It infers syntactic types such as integer, float, or object [107]. It also provides a utility function that can check the column for the datetime type.

TransmogrifAI. This is an AutoML library for structured data in Salesforce's AutoML platform called Einstein [17]. TransmogrifAI supports rudimentary automatic feature type inference over primitive types such as Integer, Long, Double, Timestamp, and String. It also has an extensive vocabulary for feature types such as email, phone numbers, zipcodes, etc. However, users have to manually specify these types for their data.

AutoGluon-Tabular. AutoGluon is an end-to-end AutoML framework from AWS [62]. It classifies each column into numeric, categorical, date/time, text, or columns that needs to be discarded because they can't be classified into any of the classes.

Sherlock. Sherlock [79] is a distantly-supervised deep-learning-based tool that identifies 78 semantic types such as *Age*, *Code*, *Duration*, etc. But the semantic types are not directly usable for AutoML because the same semantic type can span different ML feature types. For instance, *Duration* type can be either *Numeric* (e.g., time elapsed in seconds), *Categorical* (e.g.,

time duration belonging to a discrete set), *Datetime* (e.g. the exact timestamp), or even *Sentence* (e.g., duration mentioned in words).

We find that out of 78 semantic types, 55 types can be uniquely mapped to one single class of our label vocabulary. The number of types that are mapped to 2, 3, and 4 classes of our label vocabulary are 18, 3, and 2, respectively. We release the mapping between Sherlock semantic types and our label vocabulary in the technical report [136]. We use a rule-based approach on top of Sherlock to identify one single feature type given a column. Note that a type from Sherlock vocabulary can map to multiple types from our label vocabulary. We use a rule-based approach to exclusively map a semantic type to automatically map it to our label vocabulary. We give two examples below to illustrate how we perform this mapping.

(1) To map *capacity*, we use the following rules in order. If the column has less than 20 unique values, then we label them as *Categorical*. We then check if we can cast the column to either *int* or *float* to label them as *Numeric*. Next, we check if the average number of space separated words is greater than 3 to map the column to *Sentence*. Finally, we use a regular expression to check if there are numbers followed or preceded by a set of commas and alphabets for *Embedded Number*, otherwise we map the column to *Categorical*.

(2) For *duration*, we first check if the column has less than 20 unique values to map them to *Categorical*. Next, we check if we can cast the column to be either *int* or *float* to label them as *Numeric*. We perform a pandas timestamp check for *Datetime*. We check if the average number of space separated words is greater than 3 to map the column to *Sentence*, otherwise we map them to *Categorical*.

5.3.2 Rule-based Baseline

Figure 5.4 shows the rule-based approach. We use this approach to validate if a set of rules can accurately represent our labeled dataset. We write 11 rules to capture all the classes using a flowchart-like structure. We provide two examples below. (1) To identify *List*, non-empty sample

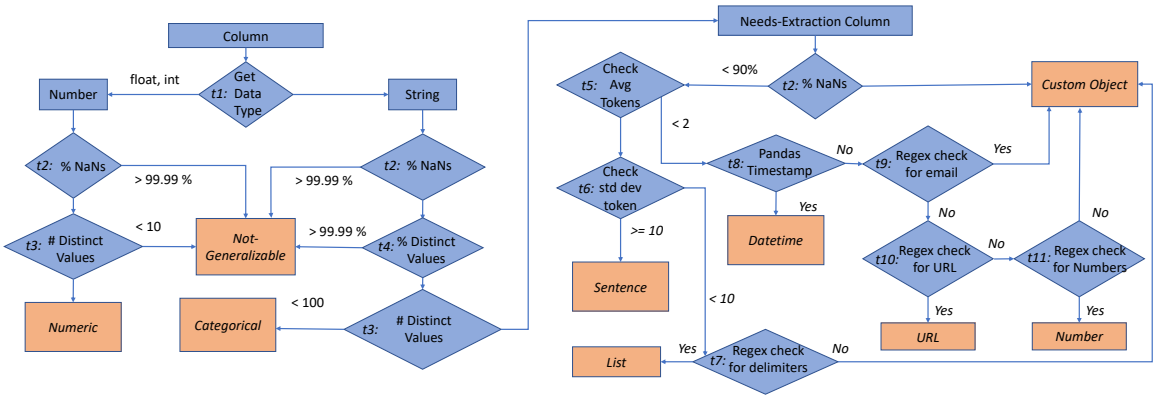


Figure 5.4: Flowchart of the rule-based baseline.

values are matched with a regular expression based check of a series of characters separated by a type of delimiter such as ; | , etc. (2) If either of the % of NaNs or % of unique values in the column are greater than 99.99% then we mark it as *Not-Generalizable*.

5.3.3 ML-based Approach using our Data

As shown in Figure 5.3, we use our labeled data to build standard ML models. Base Featurization is a common step for all ML models. Some ML models cannot operate on the raw characters of attribute names or sample values. Thus, we extract hand-crafted feature sets from the attribute names and sample values. We then train several classical ML models, *k*-NN with a distance function tuned for our task, and a CNN. Finally, the pre-trained model is used to infer feature types for columns in an "unseen" CSV file. At the scale of AutoML platforms where there are potentially millions of columns, human intervention can be costly and slow. The models output predictions and the corresponding confidence scores for each class. Thus, an ML-based approach allows users to intervene to prioritize their effort towards *Context-Specific* types or columns with low confidence scores that may need more human attention.

Feature Extraction. The attributes with similar names can likely belong to the same class. For instance, both attributes *temperature_jan* and *temperature_feb* are *Numeric*. Similarly, knowing that the sequence of characters are numbers followed by a "/," can give an indication of

Datetime. Based on these intuitions, we extract an *n*-gram feature set from the attribute names and sample values.

Notation. We denote the descriptive stats by $\mathbf{X}_{stats}$, the attribute name by $\mathbf{X}_{name}$, and randomly sampled attribute values by $\mathbf{X}_{sample}$ (first sampled value referred to as $\mathbf{X}_{sample1}$ and similarly for other values). We leverage the commonly used bigram features on the attribute name (denoted by $\mathbf{X2}_{name}$) and sample value ($\mathbf{X2}_{sample}$).

Classical ML models. We consider classical models: Logistic Regression, RBF-SVM, and Random Forest. Note that they cannot operate on raw characters of attribute names or sample values. Thus, we use features: $\mathbf{X}_{stats}$, $\mathbf{X2}_{name}$, $\mathbf{X2}_{sample1}$, and $\mathbf{X2}_{sample2}$. For scale-sensitive models such as RBF-SVM and logistic regression, we standardize $\mathbf{X}_{stats}$ to have mean 0 and standard deviation 1.

Nearest Neighbor. Most implementations of *k*-NN use a simple Euclidean distance. But, we can adapt the distance function for the task at hand by defining the weighted distance function as:

$$d = ED(X_{name}) + \gamma \cdot EC(X_{stats})$$

Here, *ED* (resp. *EC*) is the edit distance (resp. euclidean distance) between X_{name} (resp. X_{stats}) of a test example and a training example. γ is the parameter that needs to be tuned during training.

CNN. Inspired by the success of CNN on short text classification tasks [173, 172], we leverage a character-level CNN for our task. Figure 5.5 (A) shows the architecture of CNN model. The layers of CNN are shown in Figure 5.5 (B). The network takes attribute name, descriptive stats, and sample values as input and outputs the class from the label vocabulary. The attribute name and sample values are first fed into an embedding layer. The embedding layer takes as input a 3*D* tensor of shape (*NumSamples, SequenceLength, Vocabsize*). Each sample (attribute name or sample value) is represented as a sequence of one-hot encoded characters. *SequenceLength* represents the length of this character sequence and *Vocabsize* denotes the number of unique

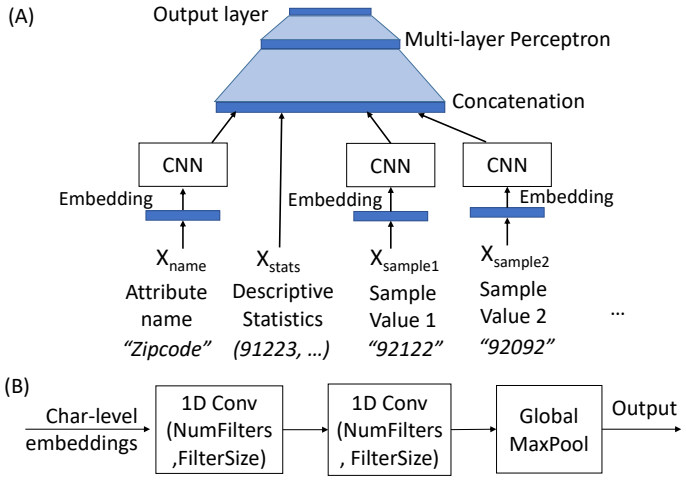


Figure 5.5: (A) The end-to-end architecture of our deep neural network. (B) CNN block's layers.

characters represented in corpus. The embedding layer maps characters to dense vectors and outputs a 3*D* tensor of shape (*NumSamples, SequenceLength, EmbedDim*), where *EmbedDim* represents the dimensionality of embedding space. The weights are initialized randomly and during training, the word vectors are tuned such that the embedding space exhibits a specialized structure for our task.

The resultant tensor from the embedding layers is fed into a CNN module, which consists of three cascading layers, 2 1-D Convolutions Neural Network, followed by a global max-pooling layer. The size of the filter (*FilterSize*) and number of filters (*NumFilters*) are tuned during training. We concatenate all CNN modules with descriptive statistics and feed them to a multi-layer perceptron on top. In the output layer, we use a softmax activation function that assigns a probability to each class of the label vocabulary. The whole network can be trained end-to-end using backpropagation.

5.4 Empirical Study and Analysis

We now empirically compare the industrial open source tools and ML models on the accuracy of type inference. This is the *very first empirical comparison* of this sort of these

Table 5.3: Binarized class-specific accuracy of different approaches on our benchmark labeled held-out test dataset. The bold fonts highlight the most accurate approach/model per class.

Feature Type	Metric	Open-source Industrial Tools				Sherlock + Rules	Baseline	Models trained on our data		
		TFDV	Pandas	TransmogrifAI	AutoGluon		Rule-based	Log Reg	CNN	Rand Forest
Numeric	Precision	0.657	0.614	0.605	0.646	0.599	0.773	0.909	0.929	0.934
	Recall	1	1	1	1	0.359	0.946	0.943	0.941	0.984
	Accuracy	0.814	0.776	0.767	0.805	0.683	0.882	0.946	0.953	**0.97**
Categorical	Precision	0.396	-	-	0.667	0.311	0.577	0.808	0.846	0.913
	Recall	0.652			0.534	0.707	0.457	0.884	0.928	0.943
	Accuracy	0.691			0.831	0.567	0.798	0.925	0.945	**0.966**
Datetime	Precision	0.985	0.956	1	1	0.89	0.559	0.951	0.925	0.945
	Recall	0.475	0.915	0.454	0.844	0.801	0.135	0.972	0.965	0.972
	Accuracy	0.962	0.991	0.961	0.989	0.979	0.931	0.994	0.992	**0.994**
Sentence	Precision	0.472	-	-	0.516	0.354	1	0.913	0.725	0.865
	Recall	0.457			0.902	0.554	0.043	0.793	0.804	0.902
	Accuracy	0.951			0.956	0.932	0.956	0.987	0.977	**0.989**
Not-Generalizable	Precision	-	-	-	0.465	0.692	0.216	0.732	0.81	0.934
	Recall				0.53	0.042	0.507	0.732	0.66	0.86
	Accuracy				0.883	0.893	0.747	0.947	0.937	**0.978**
Context-Specific	Precision	-	0.08	0.074	-	0.192	0.211	0.747	0.741	0.859
	Recall		0.295	0.295		0.168	0.195	0.621	0.663	0.705
	Accuracy		0.609	0.582		0.851	0.853	0.944	0.946	**0.961**

tools, thanks to our new benchmark labeled dataset. The headline result is that our ML models substantially surpass these prior tools on test accuracy.

5.4.1 Methodology and Setup

Methodolody. We partition our labeled dataset into a train and held-out test set with 80:20 ratio. We perform 5-fold nested cross-validation of the train set, with a random fourth of the examples in a training fold being used for validation during hyper-parameter tuning. For all the classical ML models, we use the Scikit-learn library in Python. For CNN, we use the popular Python library Keras on Tensorflow. We use a standard grid search for hyper-parameter tuning, with the grids described in detail below.

Logistic Regression: There is only one regularization parameter to tune: C. Larger the value of C, lower is the regularization strength, hence increasing the complexity of the model. The grid for C is set as $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100, 10^3\}$.

RBF-SVM: The two hyper-parameters to tune are C and γ. The C parameter represents the penalty for misclassifying a data point. Higher the C, larger is the penalty for misclassification. The $\gamma > 0$ parameter represents the bandwidth in the Gaussian kernel. The grid is set as follows: $C \in \{10^{-1}, 1, 10, 100, 10^3\}$ and $\gamma \in \{10^{-4}, 10^{-3}, 0.01, 0.1, 1, 10\}$.

Random Forest: There are two hyper-parameters to tune: *NumEstimator* and *MaxDepth*. *NumEstimator* is the number of trees in the forest. *MaxDepth* is the maximum depth of the tree. The grid is set as follows: *NumEstimator* $\in \{5, 25, 50, 75, 100\}$ and *MaxDepth* $\in \{5, 10, 25, 50, 100\}$.

k-Nearest Neighbor: The hyper-parameter to tune are the number of neighbors to consider (k) and the weight parameter in our distance function (γ). We use all integer values from 1 to 10 for k. The grid for γ is set as $\{10^{-3}, 0.01, 0.1, 1, 10, 100, 10^3\}$.

CNN Model: We tune *EmbedDim*, *numfilters* and *filtersize* of each Conv1D layer. The MLP has 2 hidden layers and we tune the number of *neurons* in each layer. The grid is set as follows: *EmbedDim* $\in \{64, 128, 256\}$, *numfilters* $\in \{32, 64, 128\}$, *filtersize* $\in \{2\}$, and *neurons* $\in \{250, 500, 1000\}$. In order to regularize, we use dropout with a probability from the grid: $\{0.25\}$. Rectified linear unit (ReLU) is used as the activation function. We use the Adam stochastic gradient optimization algorithm to update the network weights. We use its default parameters.

We also did a 5-fold leave-data file out cross-validation to "stress-test" the ML models for new data files. The raw data files were split into 60:20:20 train, validation, and test partitions where each partition has all columns of a particular data file. Thus, the test partition has columns of the raw data files that the model has not seen before. The trends of the leave-data file out approach are similar to the former approach, we discuss its results in the Section 5.4.3.

Table 5.4: Full 9-class test accuracy of the ML models trained on our data with different feature sets. $X^*_{name}, X^*_{sample1}, X^*_{sample2}$ denote bigram features ($X2_{name}, X2_{sample1}, X2_{sample2}$) for classical ML models and raw character-level features ($X_{name}, X_{sample1}, X_{sample2}$) for CNN and k-NN. The bold fonts highlight the most accurate feature set for that model.

	X_{stats}	X^*_{name}	$X^*_{sample1}$	X_{stats}, X^*_{name}	$X_{stats}, X^*_{sample1}$	$X^*_{name}, X^*_{sample1}$	$X^*_{sample1}, X^*_{sample2}$	$X_{stats}, X^*_{name}, X^*_{sample1}$	$X_{stats}, X^*_{name}, X^*_{sample1}, X^*_{sample2}$
Logistic Regression	0.6862	0.7293	0.6603	0.8428	0.7763	0.8043	0.7144	0.8578	**0.8643**
RBF-SVM	0.8213	0.777	0.6521	0.8724	0.7845	0.8159	0.7131	**0.8761**	0.8712
Random Forest	0.9121	0.7785	0.6657	**0.9259**	0.8956	0.8346	0.7374	0.9216	0.9096
CNN	0.6809	0.8019	0.6805	0.8692	0.7965	0.8655	0.7763	**0.8788**	0.8701
k-NN	0.8605	0.7839	-	**0.8796**	-	-	-	-	-

Experimental Setup. We use CloudLab [61] with custom OpenStack profile running Ubuntu 18.04 with 10 Intel Xeon cores and 192GB of RAM. For TFDV, Transmogrifai, AutoGluon, and Pandas, we use version number 0.22.2, 0.7.0, 0.0.11, and 0.25.3 respectively.

Metrics. Our key metric is prediction accuracy for the 9-class task. We also use class-specific binarization metrics such as precision, recall, F1 score, and confusion matrix.

5.4.2 Comparison of All Approaches

We compare ML models trained on our dataset against open-source tools on our held-out test data. Figure 5.2 showed the feature type vocabulary of these tools and how they map to our vocabulary. Since none of these tools support our full 9-class vocabulary, we report results on binarization of our classes: *Numeric* vs. all Non-Numeric, *Categorical* vs. all Non-Categorical, and similarly for others.

Results. Table 5.3 presents the precision, recall, and overall 2 x 2 diagonal accuracy results of all approaches on our benchmark labeled held-out test set[1]. Table 5.5 shows the train, cross-validation, and test accuracy results of all models trained on our dataset with 5-fold

[1] We have released version 2 of our labeled dataset where 32 examples are relabeled after feedback on Github [37]. We find only minor changes in the results without altering any of our trends, conclusions, or takeaways discussed here. Note that our labeled dataset is a living public repository on Github which we anticipate to grow in the future. Please refer to our public repository for the up-to-date results.

Table 5.5: Full 9-class train, validation, and test accuracy of the ML models trained on our data with different feature sets. $X^*_{name}, X^*_{sample1}, X^*_{sample2}$ denote bigram features ($X2_{name}, X2_{sample1}, X2_{sample2}$) for classical ML models and raw character-level features ($X_{name}, X_{sample1}, X_{sample2}$) for CNN and k-NN. The bold fonts highlight the most accurate feature set for that model.

Model		X_{stats}	X^*_{name}	$X^*_{sample1}$	X_{stats}, X^*_{name}	$X_{stats}, X^*_{sample1}$	$X^*_{name}, X^*_{sample1}$	$X^*_{sample1}, X^*_{sample2}$	$X_{stats}, X^*_{name}, X^*_{sample1}$	$X_{stats}, X^*_{name}, X^*_{sample1}, X^*_{sample2}$
Logistic Regression	Train	0.6954	0.8553	0.7139	0.9135	0.8288	0.9236	0.7975	0.9471	0.9571
	Validation	0.6927	0.7438	0.6551	0.8477	0.7743	0.8226	0.7117	0.8668	0.8749
	Test	0.6862	0.7293	0.6603	0.8428	0.7763	0.8043	0.7144	0.8578	**0.8643**
RBF-SVM	Train	0.892	0.9114	0.7133	0.9475	0.8779	0.9166	0.8392	0.9598	0.9605
	Validation	0.8203	0.7768	0.6529	0.8691	0.7847	0.8308	0.718	0.8822	0.8780
	Test	0.8213	0.7785	0.6521	0.8724	0.7845	0.8159	0.713	**0.8761**	0.8712
Random Forest	Train	0.9771	0.9168	0.7404	0.9817	0.9734	0.9447	0.8406	0.9803	0.9787
	Validation	0.9114	0.775	0.6604	0.9236	0.8938	0.837	0.7342	0.9195	0.9162
	Test	0.9121	0.777	0.6657	**0.9259**	0.8956	0.8346	0.7374	0.9216	0.9096
CNN	Train	0.7077	0.9545	0.7433	0.9846	0.8798	0.9855	0.8588	0.9727	0.9891
	Validation	0.7016	0.8167	0.6863	0.8768	0.7966	0.8892	0.7903	0.89	0.8821
	Test	0.6808	0.8019	0.6805	0.8692	0.7965	0.8655	0.7763	**0.8788**	0.8701
k-NN	Validation	0.8728	0.8002	-	0.8889	-	-	-	-	-
	Test	0.8605	0.7839	-	**0.8796**	-	-	-	-	-

cross-validation methodology. Table 5.6 presents the binarized class-specific F1 score of different approaches on our held-out test set. Table 5.7 shows the confusion matrices of the rule-based approach, our Random Forest, and Sherlock. We present the results in-depth below.

(1) We see that the ML models achieve significantly higher accuracy than all industrial tools across the board for all feature types. For instance, a lift of 28% and 14% in accuracy in predicting *Categorical* compared to TFDV and AutoGluon respectively. Of all approaches, Random Forest achieves the highest accuracy in inferring the types.

(2) Interestingly, all the existing tools have a high recall on *Numeric* but very low precision. This is because their heuristics are syntactic, which leads them to wrongly classify many *Categorical* features such as *ZipCode* as *Numeric*. The ML models have a slightly lower recall on *Numeric*. This is because, with many features thrown, they get slightly confused and could wrongly predict a *Numeric* type as non-numeric. But, the ML models have much higher precision and high overall accuracy.

Table 5.6: Binarized class-specific F1 score of different approaches on our benchmark labeled held-out test dataset. The bold fonts highlight the most accurate approach/model per class.

Feature Type	Open-source Industrial Tools				Sherlock + Rules	Baseline	Models trained on our data		
	TFDV	Pandas	TransmogrifAI	AutoGluon		Rule-based	Log Reg	CNN	Rand Forest
Numeric	0.793	0.761	0.754	0.785	0.449	0.851	0.926	0.935	**0.958**
Categorical	0.493	-	-	0.593	0.432	0.51	0.844	0.885	**0.928**
Datetime	0.641	0.935	0.624	0.915	0.843	0.217	**0.961**	0.945	0.958
Sentence	0.464	-	-	0.656	0.432	0.082	0.849	0.762	**0.883**
Not-Generalizable	-	-	-	0.495	0.079	0.303	0.732	0.727	**0.895**
Context-Specific	-	0.126	0.118	-	0.179	0.203	0.678	0.7	**0.774**

(3) Heuristics for identifying *Datetime* by all the existing tools have high precision, even higher than the ML models. However, their rules do not capture many *Datetime* type instances (e.g., an attribute named *BirthDate* "19980112"); thus, they have a much lower recall.

(4) The heuristic rules of AutoGluon and TFDV are largely dependent upon the number of words in a string for accurately inferring *Sentence* type. Thus, a column with most of its values having a large number of words will likely get inferred as *Sentence* by these tools. However, a *Categorical* or *Context-Specific* column (e.g., containing JSON object) can satisfy the criteria provided by the rules. Thus, AutoGluon and TFDV have low precision on *Sentence*. On the other hand, the ML-based approaches have much higher precision.

Other Commercial Tools. There exist other commercial tools that also automate the ML feature type inference task such as Google AutoML Tables [9], DataRobot [5], and Trifacta [18]. However, since these systems are closed source, we do not know how these tools work. It is also hard to evaluate their accuracy because: (1) DataRobot has no public/free trial version of their platform. We got no response to our demo request. (2) AutoML Tables and Trifacta only offer GUI-based usage where users must upload the raw CSV files manually to identify the feature types. Both these tools do not provide any programmatic way for evaluation. So, we cannot evaluate their accuracy automatically. We manually uploaded 5 CSV files from our raw data. All 15 categoricals encoded as integers were (wrongly) classified as *Numeric* by both tools. Since

Table 5.7: Confusion matrices (actual class on row and predicted class on column) of (A) Rule-based baseline (B) Random Forest, and (C) Sherlock.

(A)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	669	0	0	0	0	1	0	37	0
Categorical	52	209	8	0	0	2	1	128	57
Datetime	4	7	19	0	0	0	1	90	20
Sentence	0	25	0	4	0	0	2	24	37
URL	0	12	0	0	8	0	0	6	6
Embedded Numbers	0	35	4	0	0	18	0	37	5
List	1	6	0	0	0	0	0	42	3
Not-Generalizable	35	59	0	0	0	2	0	109	10
Context-Specific	105	9	3	0	0	1	3	32	37

(B)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	696	3	0	0	0	0	0	2	6
Categorical	12	431	0	4	0	0	0	1	9
Datetime	0	2	137	0	0	2	0	0	0
Sentence	0	3	0	83	0	0	0	3	3
URL	0	2	0	0	30	0	0	0	0
Embedded Numbers	0	5	1	0	0	92	0	0	1
List	0	1	0	3	0	5	43	0	0
Not-Generalizable	3	13	6	4	1	0	0	185	3
Context-Specific	34	12	1	2	0	0	0	7	134

(C)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	254	334	5	10	0	0	0	0	104
Categorical	63	323	1	62	0	5	0	0	3
Datetime	1	25	113	2	0	0	0	0	0
Sentence	0	31	0	51	0	2	0	0	8
URL	0	25	0	0	0	0	0	1	6
Embedded Numbers	1	59	0	1	0	36	0	2	0
List	0	17	0	24	0	1	0	1	9
Not-Generalizable	59	123	6	11	0	2	0	9	5
Context-Specific	46	101	2	5	0	4	0	0	32

it is hard to draw any generalizable conclusion if these tools have the same issues as TFDV, AutoGluon, and TransmogrifAI, we leave it to future work to assert this more systematically.

5.4.3 Comparison of ML-based Approaches

Rule-based Baseline. The 9-class classification accuracy on the held-out test set is only 54%. We observe that this approach achieves 95% and 46% recall in classifying *Numeric* and *Categorical* respectively. The recall for *Categorical* is low because a category encoded as a

number is wrongly classified as *Numeric*. Admittedly, our rules are not exhaustive and one can always come up with more rules to improve the accuracy. However, writing rules for every little corner case is excruciating and will likely never be comprehensive.

Sherlock. Sherlock with a rule-based approach that maps their semantic types to our label vocabulary has an accuracy of just 42%. This is because their semantic type vocabulary is not suitable towards identifying ML feature types. The number of Sherlock semantic types (out of 78) that are mapped to ML feature types are: 14 to *Numeric*, 50 to *Categorical*, 4 to *Datetime*, 7 to *Sentence*, 11 to *Embedded Number*, 2 to *List* and *Not-Generalizable*, and 18 to *Context-Specific*. Since *Categorical* type occur most frequently, more examples in our labeled dataset are disproportionately confused with this feature type. For instance, many integer *Numeric* attributes are confused with semantic types that often contains discrete set of integers (such as *Credit* and *Class*). Interestingly, Sherlock has a high precision of 89% in identifying *Datetime* correctly, even with just 4 semantic type mapped to *Datetime*.

Classical ML Models. Table 5.4 presents the 9-class accuracy results of the classical ML models using different feature sets[1]. We present the 5-fold held-out train and validation accuracy in Table 5.5. For logistic regression, we see that the descriptive stats alone are not enough, as it achieves an accuracy of just 69% on the held-out test set. But, for RBF-SVM and Random Forest, the accuracy with stats alone is already 82% and 91% respectively. Incorporating bigrams of the attribute name into logistic regression leads to a whopping 15% lift in accuracy. However, adding more sample values does not give any rise in accuracy, except for logistic regression. Overall, Random Forest achieves the best 9-class accuracy of 93% using bigrams on the attribute name along with descriptive statistics.

CNN and Nearest Neighbor. Table 5.4 also shows the CNN and k-NN accuracy[1]. We see that with just X_{name}, the CNN accuracy is already 82%. The descriptive stats lift the accuracy further by 8%. We find that sample values are not that useful, yielding only a minor lift. With k-NN, we observe that with only Euclidean distance on descriptive statistics, the accuracy is

Table 5.8: 5-fold training, cross-validation, and held-out test accuracy of models with leave-datafile-out methodology. *k*-NN use our weighted edit distance function.

Model		$[X_{stats}, X2_{name}]$
Logistic Regression	Train	0.9201
	Validation	0.8376
	Test	0.8411
RBF-SVM	Train	0.9612
	Validation	0.8554
	Test	0.8491
Random Forest	Train	0.9821
	Validation	0.9323
	Test	0.9199
k-NN	Validation	0.8537
	Test	0.8476

already at 86%. The edit distance on the attribute name approach achieves an accuracy of 78%. Finally, our *k*-NN weighted edit distance function achieves a high 88% accuracy.

Leave-datafile-out methodology We perform 5-fold leave-datafile-out cross validation to "stress-test" our models for new data files. In this methodology, the raw data files are split into 60:20:20 train, validation, and test partitions where each partition has columns of the same source data file. Thus, the test partition has columns of the raw data file that model has not seen before. Table 5.8 present the train cross-validation, and test accuracy results of the classical ML models and *k*-NN with this methodology on the 3-gram features from attribute name and descriptive stats. We observe that the results are comparable to what we found with *k*-fold cross-validation methodology.

5.4.4 Analysis of Errors

We now explain the behavior of the best performing Random Forest on our held-out test dataset (shortened henceforth as "OurRF") by inspecting the raw datatype of the column values. Table 5.9 shows examples of columns and the corresponding prediction made by OurRF. We present the full confusion matrix of the predicted class by OurRF vs actual data type of the attribute value in Table 5.7. We intuitively explain the errors by class below.

Table 5.9: Examples of errors made by RandomForest. *Numeric* (*NU*), *Categorical* (*CA*), *Datetime* (*DT*), *Sentence* (*ST*), *Not-Generalizable* (*NG*), *Embedded Number* (*EN*), *URL*, *List* (*LST*), and *Context-Specific* (*CS*) are feature types.

#	Attribute Name	Sample Value	Total Values	% Distinct Values	% NaNs	Label	RF Prediction
A	s1p1c2area	50	9597	3.6	45.2	*NU*	*CS*
B	Tenure Status	Own house, rent lot	41544	0.02	0	*CA*	*ST*
C	End	March 4, 1797	45	97.8	2.2	*DT*	*EN*
D	Name	Battle of Riverrun	38	100	0	*ST*	*NG*
E	%White	18.90%	192	58.9	0	*EN*	*CA*
F	Countries	ru; uk; mx	1359	32.9	46.3	*LST*	*EN*
G	q19TalToolResumeScreen	#NULL!	25090	0.008	6	*NG*	*CA*
H	Livshrmd	151	9597	1.17	42.3	*CS*	*NU*

Numeric and Context-Specific. We see that OurRF is less likely to misclassify a *Numeric* attribute whose values are floats or negative numbers compared to integers. We observe that with integers, OurRF gets most confused with *Context-Specific* class, e.g., *s1p1c2area* (Table 5.9 example(A)). This is possibly because of the non-sensical attribute name. Similarly, *Context-Specific* integers are most commonly misclassified with *Numeric* (Table 5.9 example(H)).

Categorical and Not-Generalizable. When the sample values are strings with more than one token, OurRF is more likely to misclassify *Categorical* as *Sentence* or *Context-Specific* (Table 5.9 example(B)). *Not-Generalizable* types are often confused with *Categorical*. For instance, *q19TalTool-ResumeScreen* (Table 5.9 example(G)) has only 2 values in its domain: "NULL!" and "ResumeScreen." However, OurRF treats "NULL!" as a separate category. Thus, OurRF is lacking in its semantic understanding ability of sample values.

Other types. We find that our model achieves high precision and recall in inferring other types such as *Datetime* and *URL*. In addition, *List* types are often confused with *Embedded Number* (Table 5.9 example(C)) even though there is no number available for extraction. This can be due to few available training examples for *List* type.

Table 5.10: Number of examples of *Country* and *State* type in train and held-out test splits of our labeled dataset

	Train set	Held-out test set
Country	56	10
State	29	14

Table 5.11: Accuracy of Random Forest trained using $(X_{stats}, X2_{sample1})$ feature set with the 10-class vocabulary (extending our vocabulary one at a time with either *Country* or *State*) on the held-out test set. 9-class accuracy of Random Forest with the same feature set was *0.896*.

		Adding Semantic type to our Vocabulary									
		Country					State				
		10-class Accuracy	Precision	Recall	F1 Score	Binarized Accuracy	10-class Accuracy	Precision	Recall	F1 Score	Binarized Accuracy
Adding N labeled examples to our training dataset	N=100	0.882	0.927	0.911	0.919	0.991	0.875	0.967	0.781	0.864	0.986
	N=200	0.881	0.892	0.972	0.93	0.992	0.875	0.942	0.851	0.894	0.988

5.4.5 Extensibility of our Benchmark

We discuss the extensibility of our benchmark and our ML-based approach in three parts as follows.

Part A. How extensible is our benchmark and ML-based approach to support new data types?

Setup. We conduct an experiment to understand the overhead of supporting additional types by expanding our 9-class label vocabulary. To illustrate the effort needed for this extension, we choose two semantic data types that are commonly used in business intelligence applications: *Country* and *State*. We extend our vocabulary to 10-classes by adding "enough" labeled examples of these two types one at a time. We then retrain our Random Forest on the extended vocabularies. We describe this process in detail with the following three steps.

(1) When creating our labeled dataset, we had annotated examples of *Country* and *State* type with *Categorical*, as they represent a discrete closed domain. Thus, we revisit our *Categorical* examples to identify *Country* and *State* types. Table 5.10 shows their number of examples in the train and held-out splits of our labeled data. We find that the number of examples is low to train

Random Forest and draw any general conclusions for its predictions on both types.

(2) To get access to more labels, we leverage the distantly-supervised labeled data from the Sherlock data repository. This contains data columns with headers belonging to 78 semantic types [79], including *Country* and *State*. We first relabel the *Categorical* examples identified as *Country* in step 1 to a tenth class. We then add 100 randomly sampled (weakly) labeled examples for *Country* from Sherlock data repository to our held-out test set. Thus, our test split has a total of 110 *Country* examples. We add N ($N = 100$ or $N = 200$) randomly sampled *Country* examples from Sherlock data to our train set. We form two train splits where one has 100 more labeled examples for *Country* than the other. We do not alter any of the labeled examples for the rest of the 9-classes.

(3) We use the two train splits to build two Random Forest models operating on $\mathbf{X}_{stats}$ (25 descriptive stats) and $\mathbf{X2}_{sample1}$ (bigrams on the first sample value) feature sets on the 10-classes. We use the same methodology that we discussed in Section 5.4.1 for evaluation.

We repeat the steps 2 and 3 to extend our 9-classes with *State* and we again build two Random Forest models ($N = 100$ or $N = 200$) with the same feature set.

Results. Table 5.11 presents the accuracy results of the retrained Random Forest models with the extended 10-class vocabularies on the held-out test set. We find that precision and recall are already high with just 156 training examples for *Country* ($N = 100$ case). Random Forest makes most of the wrong predictions in accurately identifying abbreviations of the countries, e.g., AFG, ALB, etc. Many of such examples are wrongly predicted as *Categorical*. When Random Forest is retrained to support *State*, it achieves a recall of 78% with just 129 training examples. The recall is lower than that of *Country* because its domain is more complex since *State* includes examples of not just the United States, but also of many other countries. Moreover, we again notice that Random Forest makes wrong predictions in accurately identifying abbreviations, e.g, CA, AL, etc. *Categorical* again causes the most misclassifications, which are reduced by adding more training examples ($N = 200$ case), as we notice that recall and F1-score improve

significantly relative to the $N = 100$ case.

Takeaways. We summarize the key takeaways from the above experiment below.

(1) Additional programming cost for feature type inference to support more types is negligible. We run the same scripts that we used to train the Random Forest model for our 9-classes.

(2) Labeling cost is marginal if one has access to semantic type datasets such as Sherlock data repository. Since a semantic type can be mapped to one or multiple ML feature types, one would need to revisit parts of our labeled data to change the labels of relevant examples. From Table 5.11, we find that the number of labels required to accurately identify semantic types is not too high. This is because such types are rich with information that is easier to detect with ML models. Moreover, adding more training labels will always help improve their recall. Note that there already exist complementary approaches and models for identifying semantic types. Thus, we explain why we chose not to label new examples of semantic types in Part C below.

(3) There exist negligible or no feature engineering cost. We do not add or remove any descriptive statistic feature to support the additional types. Table 5.11 shows that we can get high accuracy for the new types even with the same feature set that we used for the 9-classes. Thus, our features provide meaningful signals even for the newly added classes. We provide more evidence of why our feature set is robust in Part B below.

Part B. Why our featurization is robust and broadly applicable to cover any semantic type?

We designed three custom features to help our ML models identify *Datetime*, *URL*, and *List* types. These features include two regular expression-based boolean checks to identify *URL* and *List*, and one Pandas timestamp check to identify *Datetime*. For instance, the regular expression based boolean feature for *List* is used to identify if the string contains a series of characters separated by a delimiter such as `;` `|` `,` etc. The rest of the descriptive statistics features do not apply exclusively to a particular class from our 9-classes, but they are generically

applicable to cover any class.

Setup. To understand if our featurization is robust and broadly applicable, we drop the three custom type-specific features one at a time from X_{stats} (our descriptive statistics feature set). We retain other features $X2_{name}$ (bigrams on the column name) and $X2_{sample1}$ (bigrams on the first sample value) as it is. We then retrain our ML models and validate the held-out test accuracies of *Datetime*, *URL*, and *List* types. If the drop in accuracy of the three classes is significant, then it would imply the type-specific feature being predominantly important. On the other hand, if the drop is marginal then it would imply the robustness of the rest of the features to provide meaningful signals for these classes.

Results. Table 5.14 presents the ablation study results on the held-out test set with the metric being the 9-class accuracy, precision, and recall of the three classes. We choose model type family from both extremes of bias-variance tradeoff: Logistic Regression and Random Forest. We find that 9-class accuracy drops only negligibly when the three features are dropped one at a time for both models. Moreover, their precision and recall remain either unchanged or drops marginally with the newly trained model for almost all of the cases. The only cases where the drop appears significant is with Random Forest on the precision of *List* and Logistic Regression on recall of *URL*. This is because the held-out test set contains 32 and 52 examples of *URL* type respectively. Thus, misclassifying just two to three examples more leads to a 5-10% drop in precision/recall.

Takeaways. We conclude from this analysis that our featurization is robust and broadly applicable to cover any new class. This is because the signals that we extract from the raw column are column name (high-level signal), column values (low-level signals), and 25 descriptive stats (aggregate signals over the column). Thus, these standard features would provide meaningful signals even for a newly added class. Nevertheless, we do not claim that our featurization is comprehensive and unbeatable. We have publicly released raw data files as part of our benchmark and we invite contributions to devise new featurizations. Thus, our feature sets can be easily

Table 5.12: Examples of *Categorical* type from our labeled dataset along with semantic type predicted by Sherlock. Our Random Forest predicts all of them as *Categorical*.

#	Attribute Name	Type predicted by Sherlock	Total Values	# Distinct Values	Sample 1	Sample 2
A	ad744	grades	9597	3	-99	0
B	ad7125	ranking	9597	3	0	1
C	applicant_race_name_1	type	466566	7	White	Asian

expanded.

Part C. Why it may be more feasible to leverage prior work such as Sherlock directly for identifying semantic types?

If one chooses to explore semantic types on top of our label vocabulary, then they can leverage complementary approaches such as Sherlock in conjunction with our ML models, rather than labeling new examples of such types for our models. To illustrate this, we now discuss how using Sherlock can complement our ML models to help identify fine-grained semantic types. We study this in the context of *Categorical* examples of our labeled data by leveraging the 78 class semantic vocabulary of Sherlock.

Table 5.12 shows three examples of *Categorical* type columns from our labeled data along with semantic type predicted by Sherlock. *Note that we don't need to distinguish the semantic type of these columns to be used by ML, because ultimately they would be used as a Categorical feature.* Admittedly, detecting semantic types can help one discover and leverage auxiliary datasets. However, this goal is orthogonal to our focus. We focus on identifying ML feature types that dictate how columns will be consumed by ML, especially in AutoML environments. Other goals such as schema matching, data discovery, and data exploration are orthogonal to our focus. This makes semantic types different and complementary to our ML feature types.

We first find that the notion of what exactly constitutes a semantic type in Sherlock is ambiguous. We give two examples below to illustrate why it is not possible to unambiguously identify the true semantic type for every single *Categorical* example.

Table 5.13: Accuracy results of Sherlock to identify semantic types on our labeled data. OurRF denotes our best performing Random Forest on our held-out test dataset.

	True Semantic Type		
	Country	State	Gender
#Examples in Test Set	10	14	6
#Examples predicted correctly by Sherlock	5	9	5
Recall of Sherlock	**50%**	**64.3%**	**83.3%**
#Examples predicted Categorical by OurRF	10	14	6
Given Categorical Prediction from Our Random Forest			
#Examples predicted correctly by Sherlock	5	9	5
Recall of Sherlock	**50%**	**64.3%**	**83.3%**

(1) Consider example C in Table 5.12, which semantic type does `applicant_race_name_1` belongs to? Is it *class*, *category*, *classification*, or *type*? Sherlock predicts it as *type*, but its not possible for us to verify its true semantic type, even after inspecting the column description in the source metadata. However, our Random Forest predicts it as *Categorical*, which is how it will be used by ML.

(2) It is not trivial to determine the semantic type even for columns with non-sensical names such as `ad744` and `ad7125` (example A and B in Table 5.12). These columns are drawn from one source CSV file and they represent the same real-world entity ("Adaptation Climatic Variation") but for different seasons. Thus, they would belong to the same semantic type, but Sherlock gives random and different predictions for them. However, our Random Forest again predicts both of them as *Categorical*.

Considering this, we limit our analysis to a subset of *Categorical* columns from our labeled data where we can unambiguously identify their true semantic type from Sherlock's label vocabulary: *Country*, *State*, and *Gender*.

Results. Table 5.13 shows the number of examples of the three semantic types in our held-out test set. We showcase two approaches on these examples: (1) We run Sherlock independently. (2) We first get predictions from our best performing Random Forest (OurRF), followed by running Sherlock on top of OurRF's *Catgeorical* predictions. Table 5.13 also presents the

Table 5.14: Ablation study of our feature set by dropping the three type-specific features one at a time from X_{stats} with Logistic Regression and Random Forest on our held-out test set. The bold font marks the cases where the corresponding type-related feature is dropped.

Feature Set	9-class Accuracy	Datetime			URL			List		
		Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Logistic Regression										
[X_{stats}, $X2_{name}$, $X2_{sample1}$]	0.853	0.945	0.972	0.958	0.969	1	0.984	0.875	0.807	0.84
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – list-specific feature from X_{stats}	0.855	0.95	0.95	0.95	0.969	1	0.984	**0.913**	**0.807**	**0.857**
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – url-specific feature from X_{stats}	0.853	0.95	0.95	0.95	**0.967**	**0.906**	**0.936**	0.915	0.827	0.869
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – datetime-specific feature from X_{stats}	0.849	**0.936**	**0.936**	**0.936**	0.969	1	0.984	0.875	0.807	0.84
Random Forest										
[X_{stats}, $X2_{name}$, $X2_{sample1}$]	0.922	0.951	0.972	0.961	0.969	0.969	0.969	1	0.769	0.869
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – list-specific feature from X_{stats}	0.915	0.958	0.972	0.964	0.969	0.969	0.969	**0.951**	**0.75**	**0.839**
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – url-specific feature from X_{stats}	0.913	0.951	0.972	0.961	**0.969**	**0.969**	**0.969**	0.952	0.769	0.851
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – datetime-specific feature from X_{stats}	0.914	**0.938**	**0.957**	**0.947**	0.969	0.969	0.969	0.951	0.75	0.839

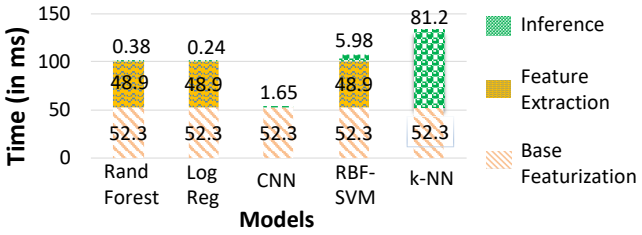


Figure 5.6: Comparison of prediction runtimes and breakdown for all models.

accuracy results of Sherlock on the three semantic types with both these approaches. We find that the recall of Sherlock with both approaches is identical on *Country*, *State*, and *Gender* type. Thus, Sherlock is truly complementary and it can be used independently or together with our ML models to identify the semantic types.

5.4.6 Prediction Runtimes and Extensions

We evaluate the running time of ML models in the online phase, i.e, to make predictions on a new column. This involves base featurization, model-specific feature extraction (only needed

for the classical models), and inference time. The measurements were made on the test set and averaged. Figure 5.6 shows the time breakdown of base featurization, model-specific feature extraction time, and inference time of ML models. Note that base featurization is common for all models and model-specific feature extraction is needed only for the 3 classical ML models. All the models finish in under 0.2 sec per column. For classical models, the additional feature extraction dominates the overall runtime. Since SVM and k-NN are distance-based methods, they have the highest runtime. Overall, CNN is the fastest.

Our benchmark and ML-based approach can be easily extended to support new additional types, including semantic types [79]. We showcase the the effort needed for this extension in the context of two semantic data types that are commonly used in BI applications: *Country* and *State*. We find that the overhead of supporting these additional types in terms of programming cost, feature engineering cost, and labeling cost is minimal to almost none. We present the complete discussion in Section 5.4.5.

5.4.7 Robustness of ML models

We now analyse the robustness of our ML models' predictions to different samples of the column. Note that our base features including five descriptive stats are generated by leveraging five random sample values from the column. Thus, a different set of column values can potentially alter the prediction of our ML models.

We perform a Monte Carlo-style study where we randomly perturb every column from our held-out test dataset 100 times. In each run, we obtain the first five non-empty distinct sample values from the column. We then use our ML models trained on X_{stats}, $X2_{name}$, and $X2_{sample1}$ to get 100 predictions for every column of our held-out test data. We finally count the percentage of times (out of 100) the predictions of the ML models remain the same as the one on the original unperturbed column. Figure 5.7 presents the CDFs of these counts on the two models across the held-out test set. Table 5.15 shows the different nth percentiles of the % of

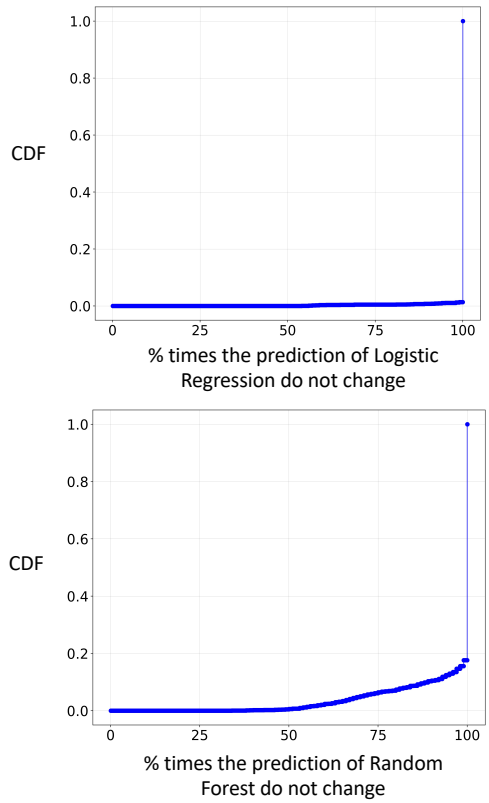


Figure 5.7: CDFs of the % of times the predictions of Logistic Regression and Random Forest do not change after perturbing columns from the held-out test data. Number of simulation runs are set to 100 for all held-out test set columns.

times the predictions of the two models change after perturbation across the held-out test dataset. We find that both Logistic Regression and Random Forest are really robust to the randomness of sampling introduced by our base featurization process. For instance, on 5% of the test examples, the prediction of Random Forest changed only 30% of times. Moreover, we observe that Logistic Regression is more robust to variability in sample values than Random Forest.

5.5 Downstream Benchmark Suite

To complete the loop on type inference, we now empirically study if doing feature type inference accurately is essential for downstream model accuracy. Thus, we verify if there are cases where doing wrong type inference may improve, reduce, or match the downstream accuracy

Table 5.15: Summary statistics of the percentage of times the predictions of Logistic Regression and Random Forest do not change after perturbing columns from the held-out test data.

nth Percentile (over held-out test set)	% times the prediction of do not change	
	Logistic Regression	Random Forest
50	100	100
20	100	100
10	100	89
5	100	70
1	100	54
0.1	95	39
0.01	57	35

relative to true feature types. From Section 5.4.3, we saw that type inference accuracy is highest for the Random Forest (OurRF) among all ML-based approaches. Thus, we compare the OurRF against the industrial and open source tools on a suite of downstream tasks we collected and curated.

5.5.1 Datasets

The impact of type inference is dependent on the dataset and the downstream prediction task. Since there are unboundedly many datasets and downstream tasks, for the sake of tractability we got 30 "unseen" datasets from Kaggle, UCI ML repository, and OpenML [155] for evaluation. Since classification tasks are more common in practice, we got 25 datasets for such tasks, and 5 for regression tasks. Table 5.17 and Table 5.18 presents the downstream datasets with descriptions such as their number of columns, target classes, and different feature types and attribute types they contain. We ensure representation of various combinations of feature types with many different data types (*ints*, *floats*, *string*, *dates*, *timestamps*, and even *primary keys*). We did not cherry-pick a dataset to particularly suit one approach over another. Overall, we have 566 columns across 30 downstream datasets. We manually label all the columns with their true feature type. The datasets and their source details are available on the Github repository [37].

Table 5.16: (A) Type Inference accuracy on 30 downstream datasets. (B) Number of downstream datasets where tools underperform, match, or outperform the ground truth downstream performance or the best performing tool. OurRF is the Random Forest for type inference trained on our data. LR denotes downstream linear model (Logistic/Linear regression) and RF denotes downstream Random Forest.

(A)	Pandas	TFDV	AutoGluon	OurRF
Column Coverage	300	535	553	566
Type inference accuracy given coverage	90.3%	75%	71.4%	91.2%

(B)	Logistic Regression				Random Forest			
	PD	TFDV	AGL	OurRF	PD	TFDV	AGL	OurRF
Underperform truth	23	18	19	11	21	17	16	9
Match truth	6	10	10	16	7	11	12	19
Outperform truth	1	2	1	3	2	2	2	2
Best performing tool for a dataset	9	11	10	23	10	14	16	24

5.5.2 Models and Metrics

In terms of downstream model evaluation, we present both extremes of bias-variance tradeoff [74]: L2-regularized Logistic regression (high bias, low variance) for classification, L2-regularized Linear regression (high bias, low variance) for regression, and Random Forest (low bias, high variance) for both classification and regression. Thus, we have 60 downstream models in total. We use the accuracy metric scaled to 100 for the classification tasks and the root mean squared error (RMSE) metric for the regression tasks.

5.5.3 Tools compared

We compare Pandas (PD), TFDV, AutoGluon (AGL), and OurRF, relative to the truth on 30 downstream datasets. We map the feature types inferred by these tools to our label vocabulary as per Figure 5.2. Columns that are inferred *Numeric* are retained as is, *Categorical* columns are one-hot encoded, *Sentence* columns are routed through TF-IDF [121], *URL*s are specially processed through a word-level bigrams, *Not-Generalizable* columns are dropped, and the rest of the types are featurized with bigrams. After featurization, we use the same methodology as in

Table 5.17: Accuracy comparison of downstream regression models using inferred types from "OurRF" against Pandas (PD), TFDV, and AutoGluon (AGL), relative to RMSE with true feature types. |A| and |Y| are the number of columns and target classes in that dataset resp. * denotes the cases where OurRF prediction is either *EN* or *CS*, where user intervention can help improve model accuracy or generalization.

Feature Types	Raw Attribute Types	Dataset	|A|	Linear Regression – L2 Regularization					Random Forest				
				Truth	PD	TFDV	AGL	OurRF	Truth	PD	TFDV	AGL	OurRF
CA	Int	MBA	2	0.363	+0.05	+0.05	+0.05	-0	0.384	+0.09	+0.08	+0.09	-0
NU + CA	Int	Vineyard	3	2.97	+2	+2	+2	-0	2.7	+0.37	+0.37	+0.37	-0
	Int, String	Apnea	3	2206.2	+62.5	-0	-0	-0	1355.7	+1972.7	-0	-0	-0
DT	Date	Accident	1	466	-0	+384.6	-0	-0	589.7	-0	+474.8	-0	-0
NU + CA + EN + NG	Int, String	Car Fuel	11	11.3	-0.09	+0.16	+0.14	+0.01*	11.7	+0.33	+1.1	+0.9	+0.03*

Section 5.4.1 for evaluation. Note that one can plug-in any alternate featurization scheme to derive more useful features. However, such feature engineering decisions can be application-specific and are not the focus of this work.

5.5.4 Results

Type Inference Results. Table 5.16 (A) shows the type inference accuracy of all tools on the downstream datasets. We see that OurRF can correctly infer the feature types for 516 out of 566 columns in these 30 datasets. Pandas has a seemingly high accuracy of 90% but note the low coverage of columns by its vocabulary, which makes it benefit from high recall. It cannot predict on the other columns at all. The accuracy of TFDV and AutoGluon is much lower than OurRF; their coverage is also slightly lower than OurRF.

Downstream Model Performance Table 5.17 and Table 5.18 present the end-to-end comparison of downstream models built with feature types inferred by Pandas, AutoGluon, TFDV, and OurRF relative to the true feature types. Table 5.16 (B) offers summary statistics on how the tools perform relative to the ground truth and other tools. We find that, for a given dataset and a downstream model, OurRF performs worse than the best performing tool for only 13 out of 60 downstream models. Moreover, OurRF underperforms the truth (perfect feature

Table 5.18: Accuracy comparison of downstream classification models using inferred types from "OurRF" against Pandas (PD), TFDV, and AutoGluon (AGL), relative to accuracy with true feature types. PK denote primary keys.

Feature Types	Raw Attribute Types	Dataset	|A|	|Y|	Logistic Regression					Random Forest				
					Truth	PD	TFDV	AGL	OurRF	Truth	PD	TFDV	AGL	OurRF
NU	Int, Float	Cancer	9	2	60.8	+0	+0	+0	+0	66.7	+0	+0	+0	+0
	Int	Mfeat	216	10	92.5	+0	+0	+0	-2.7	91.8	+0	+0	+0	-2.3
CA	String	Nursery	8	5	92.8	-0.9	+0	+0	+0	98.2	-3.9	+0	+0	+0
	String	Audiology	69	24	73	-1.3	+0	-1.3	+0	72.2	-0.9	+0	-1.3	+0
	Int	Hayes	4	3	74.1	-14.1	-14.1	-14.1	+0	78.5	-14.1	-14.1	-14.1	+0
	Int	Supreme	7	2	99.3	-14.5	-17.1	-14.5	+0	99.4	+0	+0	+0	+0
	Int, String	Flares	10	2	90.8	+0	+0	+0	+0	89.2	+0.3	+0.3	+0.3	+0
	Int, String	Kropt	6	18	39.4	-6.9	-6.9	-6.9	+0	68.8	-3.4	-3.4	-3.4	+0
	Int, String	Boxing	3	2	80.7	-24.4	-25.2	-25.2	-34.1	78.5	-17	-11.9	-11.9	-28.9
NU + CA	Int, String	Flags	28	2	68.2	-6.2	-3.6	-6.7	-4.1*	75.9	-1	-2.6	-2.6	-3.1*
	Int,Float,String	Diggle	8	2	99.9	+0	+0	+0	-5.8	99.9	+0	+0	+0	+0
	Int, Float	Hearts	13	2	84.9	-0.7	-1.6	-0.7	+0	86.2	-1.3	-3	-1.3	+0
	Int, Float	Sleuth	10	2	68.9	-3.3	-3.3	-3.3	+0	76.7	+0	+0	+0	+0
CA + NG	Int, String	Apnea2	3	2	92	-6.7	-0.6	-0.6	-0.6	90.1	-2.3	-0.8	-0.8	-0.8
NU + CA + ST	Int, String	Auto-MPG	8	3	89.1	-4.8	-8.6	-8.6	-15.9	95.2	+0.5	-18.9	-18.9	-20.5
NU + CA + EN	Int,Float,String	Churn	19	2	79.1	-0.7	+0.1	-0.1	+0.2	78.7	-0.2	-0.9	-0.8	-0.3
NU + DT + EN	Int, Float, String, Date	NYC	6	15	55.8	+0	-0.1	-0.3	-0.3	67.6	+0	+0.5	+0.8	+0.8
ST	String	BBC	1	5	97.1	-6.9	+0	+0	+0	96.3	-13.1	+0	+0	+0
DT + ST	String, Date	Articles	3	2	98.8	-2.1	+0	+0	+0	99.0	-3.2	+0	+0	+0
NU+CA+ST+NG	Int,String,PK	Clothing	10	5	66.7	-9.2	-9.1	-9.2	+0	64.2	-2.2	-4.9	-2.6	+0
NU + DT + NG	Int, String, Time, PK	IOT	4	2	83.8	-0.3	+0	+0	+3.6*	93.8	-1.4	+0	+0	+0*
NG + CA	Int,String, PK	Zoo	17	5	75.6	-13.4	-11.1	-8.9	-2.2	77.8	-15.6	-8.9	-6.7	-4.4
NU+CA+EN+NG	Int,Float,String	PBCseq	18	2	68.6	-1.3	+0.5	+0.5	+6.2*	73	-1.2	-0.1	-0.1	+2.2*
NU + CA + LST + NG + CS	Int, Float, String, PK	Pokemon	40	36	65.84	-52.2	-52.4	-52.6	-0.6	88.1	-3.9	-3.2	+0	+0
NU + CA + DT + URL + NG + CS	Int,Float,Date, String, Time	President	26	57	39.5	-7.9	-7.9	-8	-0.9	81.7	-29.4	-23.1	-28.8	-2.1

type predictions) for only 20 downstream models. In contrast, Pandas, TFDV, and AutoGluon underperform for significantly more models: 44, 35, and 35 respectively. Figure 5.8 presents the CDF of the magnitude of difference in downstream model performance with different ML feature type inference approaches compared to Truth. We find that the drop in percentage classification accuracy with OurRF relative to the Truth is less than 0.88 for 75% of the downstream models. In contrast, the 75th percentile delta drop in percentage accuracy for Pandas, TFDV, and AutoGluon relative to the Truth is 6.9, 7.7, and 6.9 respectively for the same downstream classification

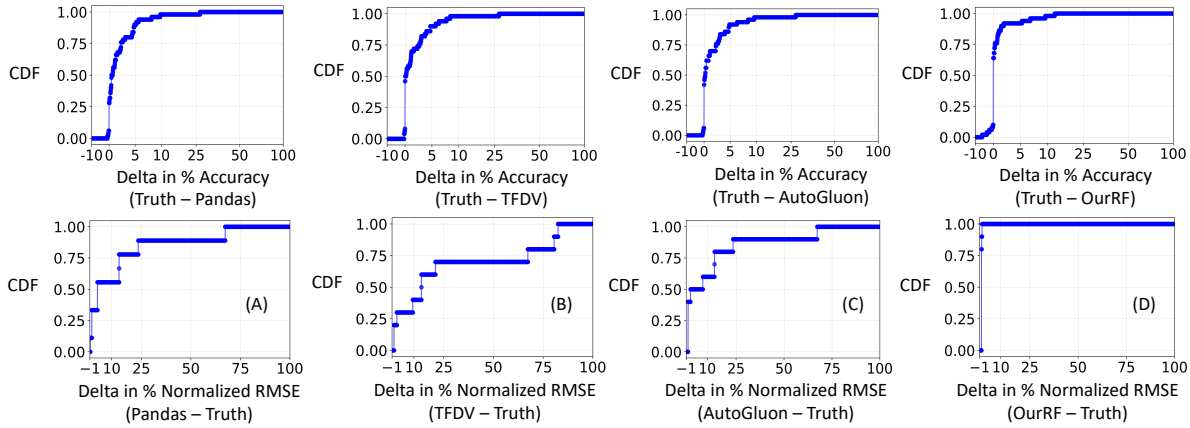


Figure 5.8: CDFs of the differences between downstream model performance with different approaches for feature type inference relative to performance with perfect feature type inference (Truth). The first and second row show CDFs on 50 downstream classification models and 10 regression models respectively. This includes both the downstream Linear model and downstream Random Forest model with (A) Pandas (B) TFDV (C) AutoGluon, and (D) Our best performing Random Forest (OurRF). For regression models, we normalize the RMSE relative to the corresponding true RMSE.

models. Moreover, the median increase in percentage normalized RMSE relative to truth is 0 with OurRF, in contrast to the median of 8.3, 13.7, 4.5 for Pandas, TFDV, and AutoGluon respectively. Thus, existing tools underperform the truth (perfect predictions of feature types) by a much higher magnitude than OurRF. We explain the results in-depth below.

1. Why does wrong type inference hurt downstream accuracy? Table 5.17 and Table 5.18 shows that wrong type inference almost always leads to a drop in accuracy compared to the accuracy with true feature types. Moreover, the amount of drop depends upon how many feature types are wrongly classified and how predictive those features are for the target. For instance, wrong type inference leads AutoGluon and TFDV to underperform on 35 out of 60 downstream models. This led to a reduction of an average of 7% and up to 52% in accuracy compared to the ground truth-based model. We explain the common patterns of how wrong type inference affected downstream accuracy below.

(a) *MFeat* has 216 *Numeric* integer columns, presenting a best-case scenario for prior tools as they have the highest possible recall in inferring *Numeric*. Thus, they classify all columns

correctly. However, OurRF confuses 7 of them with *Categorical*, possibly because of their low domain sizes, thus leading to a drop in accuracy. We verified that this is the primary reason why OurRF underperforms truth and prior tools on datasets like *Auto-MPG* and *Diggle*.

(b) On *Zoo*, out of 4 *Not-Generalizable* columns, one column is erroneously predicted as *Categorical* by OurRF. Thus, using a feature that offers no discriminative power leads to a drop in accuracy compared to the ground truth. In contrast, AutoGluon classifies all of them incorrectly. Other tools like TFDV and Pandas do not even support *Not-Generalizable* in their vocabulary. Thus, the drop is much larger for the prior tools. We observe the same pattern across many datasets like *Pokemon*, *President*, and *Car Fuel*.

2. Why does wrong type inference of *integer Categorical* often not hurt downstream Random Forest? Although the categories encoded as integers in *Supreme*, *Flags*, *Sleuth*, and *Vineyard* are misclassified by Pandas, AutoGluon, and TFDV, the accuracy of Random Forest either does not drop or drops only marginally. This is because the *Categorical* features in these datasets are either ordinal and/or have binary domain size. Random Forest has zero bias and thus can potentially represent all categories by doing splits on integers. Linear models, which have lower VC-dimension, cannot do this. Thus, the linear models often see much higher accuracy with OurRF than prior tools.

3. How can OurRF exploit user intervention to lift accuracy? *Car Fuel* has two *Embedded Number* columns. Although they are predicted correctly by OurRF, a human can intervene to extract their values to use them as *Numeric* instead of the current bigramization. Thus, a user-in-the-loop can further improve downstream model. Moreover, such intervention can even help *Flags* where a *Categorical* feature was erroneously predicted as *Context-Specific* by OurRF.

4. Why is outperforming truth not necessarily beneficial? On *IOT*, we observe the lift in accuracy due to a *Numeric* column called "temp" (denoting temperature) being classified as *Context-Specific*. This may not be desired because interpretability can be a concern in this

Table 5.19: Number of downstream datasets (out of 25) where tools with both numeric and one-hot encoded representation of integer columns outperform the tool baseline with an exclusive type specific representation, underperform truth, OurRF, NewRF, or the best performing tool. NewRF is the Random Forest trained on our labeled dataset adapted to produce multiple feature representations.

	Logistic Regression				Random Forest			
	PD	TFDV	AGL	NewRF	PD	TFDV	AGL	NewRF
Underperform truth	20	18	18	13	20	19	16	13
Underperform tool baseline	3	1	3	6	11	10	8	5
Outperform tool baseline	11	7	8	7	11	8	7	7
Outperform tool baseline but worse than NewRF	6	2	4	-	9	5	5	-
Best performing tool for a dataset	6	8	7	18	5	6	7	18

application. Predictions are more explainable when using temperature data as *Numeric* feature than bigrams.

A *Not-Generalizable* unique identifier column denoting the "case number" on *PBCseq* is predicted as *Numeric* by OurRF. Even though we notice a significant lift in accuracy compared to the ground truth, this is not necessarily beneficial in the deployment setting, where every newly conducted study will have a new case number. Thus, it is very unlikely that the downstream model will generalize.

Downstream Model Performance with Double Representation. We now give multiple representations of the column at the same time to the downstream model and study if this can help recover the gap in performance caused by wrong type inference. We study this in the context of *Numeric* vs *Categorical* dichotomy of the integer columns of our downstream datasets. Thus, with existing AutoML tools, instead of routing integer columns predicted as *Numeric* to an exclusive numerical representation and *Categorical* columns to an exclusive one-hot encoded representation, we route integer column to both representations regardless of the predicted feature type.

Note that representation of columns in multiple ways for the downstream models is completely orthogonal to whether correct feature types are known or not. Multiple representations can be performed even when the correct feature types are known. Thus, we adapt OurRF to support

the double representation of integer columns by leveraging the class confidence probabilities of the prediction. We first use a threshold-based rule to check if the prediction made by OurRF is confident enough to route the column to an exclusive feature representation corresponding to the predicted type. We set the threshold to the probability value of 0.4, i.e., the confidence of the prediction should be at least twice the random guessing accuracy on the integer column. When the class confidence probability is below the threshold, we route the column to both *Numeric* and *Categorical* representation. We denote this adapted Random Forest with "NewRF."

Table 5.19 shows summary stats on how the tools supporting double representation of integer columns perform relative to the same tools supporting an exclusive type-specific representation on our 25 downstream benchmark datasets. We observe that the accuracy of the downstream models obtained with existing AutoML tools and NewRF do improve with the double feature representation on some datasets. We find that the amount of accuracy improvement is typically higher for the downstream linear model than the downstream Random Forest. e.g., an average 3.4% and up to 29.6% on the linear model and of an average 1.3% and up to 21% on Random Forest with TFDV. Despite this, existing AutoML tools underperform NewRF on most datasets with both downstream models. In addition, NewRF underperforms the truth on 26 downstream models. In contrast, Pandas, TFDV, and AutoGluon underperform for significantly more models: 40, 37, and 34 respectively. Overall, we notice that NewRF performs worse than the best performing tool for only 14 out of 50 downstream models. Thus, accurate inference of feature types is again critical to building accurate downstream models.

One can even consider exploring different subsets of feature representations for integer columns. However, this explodes exponentially in the number of columns, which will waste a lot more runtime and resources, while still having the same interpretability issues. Note that the discussion of how feature types should be represented is completely orthogonal to our focus. Knowing a feature type correctly will always provide more information, which can be used to build better featurization schemes.

5.5.5 Summary

Overall, OurRF achieves a high accuracy of 91.2% for inferring feature types on 30 unseen datasets from Kaggle, UCI ML repository, and OpenML. Moreover, we find that wrong feature type inference almost always leads to an accuracy drop for the downstream model relative to the ground truth, except for the Random Forest on ordinal and/or binary domain *Categorical*. More importantly, our labeled dataset is valuable to build an accurate downstream model because even standard ML models like Random Forest trained on our labeled data achieve the highest accuracy against existing tools for 47 out of 60 downstream models.

5.6 Discussion

5.6.1 Public Release and Leaderboard

We have released a public repository on GitHub with our entire labeled data for the ML feature type inference task [37]. We also release the pre-trained ML models: k-NN, logistic regression, RBF-SVM, Random Forest, and the CNN. The repository tabulates the precision, recall, and accuracy of all models and existing open-source approaches. The repository includes a leaderboard for public competition on the hosted dataset with 9-class classification accuracy and per-class precision, recall, and binarization accuracy being the metric. We release the downstream benchmark suite containing 30 datasets and the associated code for running the benchmark. Also, we release the raw 1240 CSV files and we invite researchers and practitioners to use our datasets and contribute to augmenting them and creating better featurizations and models.

5.6.2 Takeaways

We make all the models and featurization routines available for use by wrapping them under functions in a Python library [37]. The ML models can be integrated for feature type

inference into existing data prep environments. *We have already integrated our pre-trained models with TFDV to improve its inference of Categorical [137].* We are also planning to collaborate with AWS and OpenML on more such integration. We welcome inquiries from more practitioners interested in adopting or enhancing our benchmark. For visual tools such as Excel and Trifacta [18], designing new user-in-the-loop interfaces that account for both model's prediction and human's judgement remains an open research question.

5.7 Conclusion

Although ML feature type inference is a crucial data prep step that serves as an entry point to ML workflow, the industrial players have signifcantly underestimated the importance of this task. Our work shows that ML feature type inference is a critical task in the end-to-end ML pipeline. Our objective benchmark (both on the specific task and several downstream tasks) exposes the shortcomings of the existing solutions. Thus, our first work in this space can substantially advance the science of building ML platforms and help objectively validate and improve them.

Chapter 5 contains material from "Towards Benchmarking Feature Type Inference for AutoML Platforms" by Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar, which appeared in Proceedings of the 2021 International Conference on Management of Data (SIGMOD'21). The dissertation author was the primary investigator and author of this paper.

Chapter 6

Category Deduplication: Assessing Importance for ML and Making Automation Accurate

6.1 Introduction

In this chapter, we dive deeper into a common and specific data prep task of category deduplication, i.e., consolidating duplicates in the *Categorical* column. This involves manual grunt work that is both tedious and time-consuming. Even AutoML users are often asked to manually perform many data prep steps before using their platforms [8]. Surveys of AutoML users have repeatedly identified such challenges in conducting data prep [166, 52].

Consider a simplified dataset to be used for a common ML classification task, *Customers Churn prediction* in Table 6.1. Duplicates, categories referring to the same real-world object occur in many *Categorical* columns such as *Gender*, *State*, *Country*, and *Contract*. Note that *Name* is not *Categorical* since it offers no discriminative power and cannot be generalized for ML. The presence of duplicates within a *Categorical* column can potentially dilute the signal strength that

Table 6.1: Customers data used for ML Churn Prediction.

CustId	*Name*	*Age*	*Gender*	*State*	*Country*	*Contract*	*Churn*
101	John	42	Male	California	US	monthly	'Y'
102	Jerry	29	Mail	CA	United states	Month-to -month	'N'

one can extract for ML. Thus, an ML practitioner would often deduplicate categories before ML. Even, AutoML platforms often suggest users to manually inspect *Categoricals* and consolidate duplicates whenever they arise, as part of their guidelines for obtaining an accurate model [7]. This can involve non-trivial amount of deduplication effort at a *Categorical* column-level as duplicates can arise as misspellings, abbreviations, and synonyms, even within the same column. Note that this problem is related but complementary to entity deduplication issues studied in the data cleaning literature, as we will explain shortly.

Considering the laborious nature of the deduplication task, we ask: *How do Categorical duplicates impact commonly used ML classifiers? Is category deduplication effort even worthwhile for ML? Is it always needed regardless of the employed Categorical encoding scheme?* We take a step towards answering these questions by developing an in-depth scientific understanding of the importance of category deduplication for ML. Our objectives are two-fold. (1) Perform an extensive empirical study to measure the impact of *Categorical* duplicates on ML and distill the findings into actionable insights for handling them. This can help ML practitioners decide when and how to prioritise their cleaning effort. Moreover, this can enable AutoML platform builders design better ML workflows. (2) Present critical artifacts that can help advance the science of building AutoML platforms by providing researchers an apparatus to tackle open questions in this direction.

Our Approach. We identify that the impact on ML accuracy in presence of *Categorical* duplicates can be characterized with several confounders such as duplication properties (e.g., # duplicates per entity), training data properties (e.g., # available training examples), *Categorical* encoding method (e.g., *One-hot*), and ML model (e.g., high variance model such as Random

Forest). Figure 6.1 details these confounders. Considering this, we make three component contributions to covers our goals. (1) We produce a labeled data to study how real-world duplicates arise. This helps us set up the duplication properties in the simulation study. More importantly, this can serve as a valuable artifact for the community to address many important open questions such as automating deduplication itself. (2) We phenomenalise the impact on ML accuracy by empirically benchmarking with real-world data containing duplicates in presence of multiple confounders. (3) The significance of each confounder is hard to discern when all confounders act together. Thus, we use simulation study to disentangle the impact with all confounders and explain the phenomenon discretely. We explain each component below.

1. Hand-Labeled Data. To understand the behavior of *Categorical* duplicates, we first ask several important questions: *How do duplicates manifest themselves in the real-world columns? Do they happen often? How much can the domain size of the Categoricals be reduced with deduplication?* We address them by creating the first hand-labeled dataset where true entities within a *Categorical* column are annotated with corresponding duplicates. Our dataset includes 1248 string *Categorical* columns from 217 raw CSV files. The labeling process took about 120 man-hours across 5 months.

The utility of our labeled dataset is two-fold. (1) Get an average-case and worst-case estimate of several properties of duplicates such as their occurrences, cardinalities, and the fraction of entities that are diluted with duplicates. This enables us to control the synthetic duplication process that is real-world representative in the simulation study. (2) Provide a methodological approach to help address several critical open questions such as benchmarking and automating the deduplication task itself. For instance, this can benefit the data cleaning community which is often limited with synthetic datasets to evaluate their methods for a lack of annotated real data such as this, albeit for different cleaning operations [40].

2. Downstream Benchmark Suite. We create a benchmark suite of 14 real-world datasets to empirically benchmark the impact of *Categorical* duplicates. We make the following

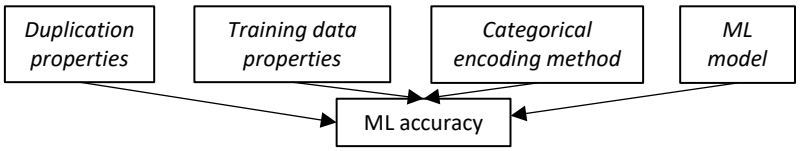


Figure 6.1: High-level summarization of confounders impacting ML accuracy in presence of duplicates.

choices with each confounder. (1) We choose three popular classifiers from the entire spectrum of bias-variance tradeoff: low-capacity Logistic Regression (`LR`) and high-capacity Random Forest (`RF`) from the two ends. We choose a high-capacity multi-layered perceptron (`MLP`) with VC dimension lower than `RF` somewhere in between them [138]. `LR` and `RF` are also the two most popular classifiers among ML practitioners, as per Kaggle data science survey [143]. (2) We focus on three easy to understand *Categorical* encoding schemes, *One-hot*, *String*-based (applicable for tree learners), and *Similarity*. We choose the first two since they are already popular in practice, as per study on OpenML workflows [101]. We choose *Similarity* as it offers category duplicates with a feature vector representation that is similar to that of their true entities. In contrast, *One-hot* leads to duplicates with feature vectors orthogonal to their true entities. There exists a large stock of encoding methods for representing *Categoricals* [73]. We leave studying them to future work for tractability sake. We make empirical observations in the two extremes of (3) Data regime in terms of the training examples per category. (4) Amount of duplication in terms of the parameters that characterize duplicates. (5) Relevancy of *Categorical* column with duplicates for the ML task.

3. Synthetic Simulation Study. We perform a simulation study to intricately study how six different confounders such as the fraction of entities with duplicates and data regime affect ML accuracy. We disentangle the structural model parameters impacting the bias-variance tradeoff by fixing the model capacity. We use high bias models such as shallow Decision Tree, a low-capacity Multi-layer Perceptron (`MLP`), and `LR` and also high-capacity `RF` and `MLP` to showcase the two extremes of each model's bias spectrum. We embed two different true distributions that present

two scenarios of how RF and hyperplane-based classifiers fit the data. The duplicates are then introduced with a process that is informed by the understanding of how they occur within our labeled data. Here, we have two objectives. (1) Confirm the validity of the observations we make with the downstream benchmark suite. (2) Disentangle and characterize the effect of duplicates with multiple confounders individually to make the impact interpretable. Moreover, we expose a critical shortcoming of the commonly used *Categorical* encodings, *One-hot* and *String* when duplicates arise during the deployment in the test set but not during training.

Relationship to Prior Work. Entity Matching (EM) solutions [93, 174, 105] operate at a tuple-level since they have access to the entire feature vectors of the two tables. Note that tuple-level duplicates do not necessarily imply duplication in *Categorical* strings, and also vice versa. Thus, problem of EM is orthogonal to category deduplication. Admittedly, it is possible to view category deduplication as an extension of row-level deduplication but doing so is non-trivial. *Regardless, our focus is to study only the impact of category deduplication on ML and not how to perform deduplication.* We leave it to future work to automate this, including potentially extending existing row-level deduplication works.

Note that a *Categorical* column for an ML model assumes values from a finite closed domain. Thus, non-generalizable open domain person names, custom processable addresses, or even semantically rich textual descriptions used in public datasets [93, 94] and string matching literature [102, 46] are not *Categorical*; thus, they are beyond the scope of this work. Although incidental *Categorical* duplicates do arise in prior datasets [93, 103, 46], we posit that we need a systematic benchmark to characterize and understand their impact on ML accuracy that prior works do not focus on. A prior work evaluated the impact of many data cleaning steps on ML [103]. Our work is along the same direction, but they did not specifically explore *Categorical* deduplication and its causal confounders that matter for accuracy. We deep dive into *Categorical* deduplication to offer empirical rigor and understand the importance of task scientifically, rather than a coarse-grained study of cleaning for ML.

Empirical Evaluation. An empirical comparison of our downstream benchmark suite reveals that deduplication can often improve the ML accuracy significantly, e.g., the impact with duplicates on `LR`, `RF`, and `MLP` using *One-hot* encoding is significant (more than 1% accuracy) on 6, 11, and 8 datasets (out of 14) respectively. Thus, `LR` gets impacted much less than the high-capacity `RF` and `MLP`. Moreover, we find that *Similarity* encoding is more robust than other encodings to tolerate duplicates. Overall, we make *eight* such observations on the significance of the confounders with the downstream benchmark suite. We confirm all of them with simulation study. Specifically, we find that the presence of duplicates can often cause extra overfitting, which reduces as the number of training examples per category rises. Moreover, we provide explanations and insights into how ML models with different biases behave with duplicates. For instance, we find that `LR` is (perhaps counter-intuitively) more robust to duplicates than high-capacity `RF` and `MLP` because although duplicates increase feature dimensionality of *Categoricals*, `LR` often just ignores the extra dimensions by setting their weights close to 0.

Making Deduplication Automation Simpler and Accurate. Although category duplicates can often impact downstream ML accuracy substantially, none of the existing open source AutoML tools such as AutoGluon [62], AutoML Tables [9], and TransmogrifAI [17] support an automated deduplication workflow. Cleaning duplicates manually can be slow and frustrating for many users, especially non-technical lay users who were promised end-to-end automation of the entire ML workflow. Thus, we pursue a complementary research direction of accurately automating category deduplication task using our labeled dataset. We cast this task as a binary classification problem and build rule-based and learning-based approaches to evaluate them on the task. Our feasibility study shows that our labeled data is useful to deliver ML-based solutions that are more accurate than rule-based baselines.

Takeaways for Practitioners. We distill our empirical analysis into a handful of actionable takeaways for ML practitioners and AutoML developers. For instance, `LR` is more robust to adverse impact of *Categorical* duplicates than high-capacity `RF` and `MLP` as it overfits less.

Moreover, *Similarity* is more robust than other encodings to tolerate *Categorical* duplicates, thereby diminishing the utility of category deduplication. We also expose a critical shortcoming of *One-hot* and *String* encodings, when *Categorical* duplicates arising in the deployment but not during training can lead to significant degradation in ML performance.

Some of these insights may be considered folklore by practitioners, but this work is the first in-depth systematic scientific study to assess the impact of *Categorical* duplicates on ML. We explain the impact from the bias-variance tradeoff perspective to put the empirical results on a rigorous footing. Our analyses can benefit practitioners to systematically understand the various confounders that matter for accuracy. Also, this can be useful to develop better practices and design ML workflows that are robust to *Categorical* duplicates. Moreover, our work opens up new research directions at the intersection of ML theory, data management, and ML system design. Finally, we open source our entire benchmark including our labeled data, downstream suite, and simulation framework [36].

In summary, our work makes the following key contributions.

1. **A new benchmark dataset.** To the best of our knowledge, this is the first work to curate a large labeled dataset specifically for *Categorical* duplicates where the entities are annotated. We present several insights that characterizes how *Categorical* duplicates exhibit themselves.

2. **Empirical benchmarking to understand the significance of category deduplication on ML.** Our curated downstream benchmark containing "in-the-wild" datasets enables us to point out cases where ML may or may not benefit with category deduplication.

3. **Characterization of confounders with simulation study.** Our synthetic study can disentangle and explain the impact of confounders on how *Categorical* duplicates affect ML.

4. **Utility of our study.** We present the first in-depth scientific empirical study to systematically

Table 6.2: Notations used to study the impact of *Categorical* duplicates on ML.

Symbol	Meaning
C	Set of category values in the column A_l
E	Set of unique real-world entities referred by categories from C
ED	Subset of real-world entities that have at least 1 duplicate; ED $\subseteq E$
occ(Z)	Sum of occurrences of all categories present in set Z; Z $\subseteq$ C
D	A set of non-empty sets of duplicate values for each entity in ED; |D| = |ED|

charactertize when and why category deduplication can help/not help ML. We present several practical insights for practitioners. We identify open questions for further research and how our labeled data can be a key enabler to address them. Also, we open source our benchmark to enable more community-driven contributions.

5. **Utility of our labeled data.** We show that even classical ML models (trained on our labeled dataset) with standard features can outperform the rule-based solutions for accurately automating category deduplication task.

6.2 Background

6.2.1 Assumptions and Scope

We focus on the ML classification setting over structured data. We call the ML model to be trained over the data as the "downstream model." This refers to the ML model that is built on the table after deduplication has been done. Note that our goal is not to study the upstream deduplication process itself, which is handled manually in this work. We leave designing automated upstream deduplication mechanisms to future work. We focus on understanding how duplicates manifest themselves in real-world and how they impact the performance of the downstream models. Specifically, we study them in the context of string *nominal Categorical* features, which do not have a notion of ordering among its values. Note that a *Categorical* feature contains mutually exclusive values from a known finite domain set. In contrast, *Text* type features

Table 6.3: A simplified example to illustrate our notions with *State* column categories.

Category set C_i ($1 \leq i \leq \|C\|$)		Occurrence of Category (occ($\{C_i\}$))	Entity set E_j ($1 \leq j \leq \|E\|$)	
New York	C_1	60	New York	E_1
NY	C_2	30		
new york	C_3	10		
California	C_4	70	California	E_2
Ca	C_5	30		
Wisconsin	C_6	100	Wisconsin	E_3

can take arbitrary string values. Thus, generic open domain addresses or person names are not *Categorical*. We study duplicates arising in *Categorical* column, which is not the actual target for the prediction task.

6.2.2 Definitions

We present terms and notations needed to study the effect of *Categorical* duplicates in the context of implications for ML accuracy. We first draw upon notations from a mix of both database theory [106] and ML literature [74] for known concepts. A relational table is defined by schema $R(A_1, A_2, ..., A_n, Y)$ with a relation (instance) r. We use $\mathcal{A}$ to denote a set of columns $\{A_1, A_2, ..., A_n\}$ and Y is the target column for prediction. Note that, formally, a column is referred to as an attribute [106]. Let $A_l (l \in [1, n])$ be a *Categorical* column with a domain $dom(A_l) \subseteq \mathcal{L}$, where $\mathcal{L}$ is the set of strings with finite length. A relation r is defined over $\mathcal{A}$ as a set of mappings with $\{t^p : \mathcal{A} \rightarrow \bigcup_{l=1}^{n} dom(A_l), p = 1...|r|\}$, where for each tuple $t^p \in r$, $t^p(A_l) \in dom(A_l)$, $|r|$ is the number of examples in the the table.

Note that *Categorical* strings are not directly consumable by most ML models. Thus, an encoding scheme is required to transform the set of columns $\mathcal{A}$ to a feature vector to train an ML model. We explain this further in Section 6.4.1. We now reuse and adapt terminologies from existing database [106, 49] and ML literature [74] together for terms that we need for the rest of this work. Table 6.2 lists the notations and Table 6.3 explains the terms used with an example. For simplicity of exposition, we focus on one *Categorical* column with duplicates, $A_l \in \mathcal{A}$.

DEFINITION (CATEGORY). A Category set $C^l = \{C^l_1, C^l_2, ..., C^l_{|C^l|}\}$ contains all unique domain values occurring in the column A_l. Note that C^l is also referred to as the active domain of A_l relative to relation r [106], i.e., C^l=$adom(A_l, r)$=$\{c \in dom(A_l) \mid \exists t^p \in r, t^p(A_l) = c\}$. We drop the superscript (C^l) and simplify the active domain operation with C only to make it succinct for follow up set algebra. Each distinct value in the column is defined as "category." For Table 6.3 example, $C = \{$*New York, NY, new york, California, Ca, Wisconsin*$\}$.

DEFINITION (ENTITY). An Entity set $E \subseteq C$ represents a subset of *Categories* that conceptually refer to different real-world objects. A category from set C can be uniquely mapped to an entity from set E. Let the mapping function be denoted by $M : C \rightarrow E$. In Table 6.3, there are three unique real-world state objects, i.e., $E = \{$*New York, California, Wisconsin*$\}$. Note that entities are defined at a conceptual level; thus, referring to New York as new York or NY is identical. But for ease of exposition, we assume the category that most frequently represents an entity (ties broken lexicographically) in the column to be the true entity. There exist multiple categories representing the same entity, i.e., $M(C_1)$=$M(C_2)$=$M(C_3)$=E_1=$\{$*New York*$\}$.

DEFINITION (OCCURRENCE). We define Occurrence (or percentage Occurrence) of category C_i as percentage of times C_i represents E_j in the column. For instance, whenever real-world *New York* entity occurs, 30% and 10% of the times *NY* and *new york* represents them respectively. *New York* is referred to as the entity since it occurs more than *NY* and *new york*.

We define the *Occurrence* function as $occ : Z \rightarrow [0, 100]$. The input Z is a subset $Z \subseteq C$ such that all categories of the subset map to a unique entity E_j ($j \in [1, |E|]$), i.e., $E_j = M(Z_1) = M(Z_2) = ... = M(Z_{|Z|})$. The output is the sum of occurrence values for all categories present in the input set which is a real number in $[0, 100]$. $occ(Z) = occ(Z_1) + occ(Z_2) + ... + occ(Z_{|Z|})$. For instance, $occ(\{C1\})$ = 60, $occ(\{C2, C3\}) = 40$, and $occ(\{C1, C4\})$ = Undefined.

DEFINITION (DUPLICATE). There exist a duplicate for E_j whenever E_j = $M(Z_1)$ = $M(Z_2)$ = ... = $M(Z_{|Z|})$ and $|Z| > 1$. Whenever E_j occurs, the % times it is represented by

Table 6.4: Three pairs of duplicate tuples from three different datasets of the Magellan data repository [93].

Dataset Name	Left Tables			Right Tables		
Restraunt	**Address**	**Phone number**	**Name**	**Address**	**Phone number**	**Name**
	1929 Hillhurst Ave, Los Angeles, CA	(323) 644-0100	Alcove Cafe & Bakery	1929 Hillhurst Ave, Los Angeles, CA 90027	(323) 644-0100	Alcove Cafe & Bakery
Ebooks	**Author**	**Title**	**Price**	**Author**	**Title**	**Price**
	John D.T. White	101 Things You May Not Have Known About the US Masters	5.99	John White	101 Things You May Not Have Known About the US Masters	5.49
Citations	**Author**	**Entry Type**	**Title**	**Author**	**Entry Type**	**Title**
	David A. Cohn and Zoubin Ghahramani and Michael I. Jordan	article	Active Learning with Statistical Models	Cohn, David A and Ghahramani, Zoubin and Jordan, Michael I	article	Active learning with statistical models

Z_1, Z_2, and Z_n are $occ(Z_1), occ(Z_2)$, and $occ(Z_n)$ respectively. Without the loss of generality, we assume that $occ(Z_1)$¿=$occ(Z_2)$¿=...¿=$occ(Z_{|Z|})$. Since Z_1 most frequently represents the entity (ties broken lexicographically), the other categories $Z_2, ..., Z_n$ are referred to as duplicates of the entity E_j. We define $ED \subseteq E$ as the subset of the entities that contain at least one duplicate, i.e., $\exists Z \subseteq C$ s.t. $|Z| > 1$ and $M(Z_1) = ... = M(Z_{|Z|}) = ED_j (j \in [1, |ED|])$. We define a duplicate set $D_k (k \in [1, |ED|])$ for every entity in ED such that $D_k = \{Z_2, Z_3, ..., Z_{|Z|}\}$ represents a set of duplicate values, e.g., $ED_1 =$ *California,* $D_1 = \{Ca\}$ and $ED_2 =$ *New York,* $D_2 = \{$*new york, NY*$\}$.

DEFINITION (CATEGORY DEDUPLICATION). *Category Deduplication* is the task of mapping categories from set C to an entity from set E with the mapping function M. The new column after the assignment is called the *deduplicated* column. The category set C and entity set E of the *deduplicated* column are identical.

DEFINITION (COLUMN RELEVANCY). Let $Acc(\mathcal{A})$ be the % classification accuracy obtained by the ML model with a set of columns $\mathcal{A}$ to be used as features in the input. *Relevancy* of a column $A_l \in \mathcal{A}$ is defined as $Acc(\mathcal{A}) - Acc(\mathcal{A} - \{A_l\})$. This quantifies the predictive power of the column A_l for the downstream task.

6.3 Hand-Labeled Dataset

We create a labeled dataset of *Categorical* columns where *Entities* in each column is annotated with their duplicates whenever present. This enables us to understand how real-world duplicates manifest themselves and what do the sets E, ED, D and their occurrences look like. We now discuss how this dataset is created, the types of real-world duplicates present, and our dataset analysis with stats and important insights into the behavior of duplicates.

6.3.1 Existing EM datasets

Labeled datasets used in the entity deduplication literature such as Magellan [93] involve duplicates at a tuple-level. But this is an orthogonal problem to category deduplication. Tuple-level duplicates do not necessarily imply duplication in the *Categorical* strings. Likewise, duplication in a *Categorical* column may not lead to row-level duplicates. We present three pairs of duplicate records from three different datasets of the Magellan data repository in Table 6.4. Note that generic open domain attributes such as author person names and addresses are not *Categorical* features for ML. Instead, such features are context-specific where either custom features are extracted or are completely dropped as they may not generalize for ML. Moreover, *Title* in Citations datasets has rich semantic information and is typically used as a *Text* or *Sentence* type feature. A *Categorical* feature assumes mutually exclusive values from a known finite domain set. We find that almost all of the Magellan datasets involve duplication in non-*Categorical* features such as generic person names, company names, addresses, and textual values with rich semantic information. Thus, they are not relevant for us to study category deduplication. We focus exclusively on the *Categorical* features and curate the first labeled dataset of entities annotated with duplicates within a *Categorical* column.

Table 6.5: Duplication types with examples from our hand-labeled data

Duplication Types		Column name	*Category* Examples
1	Capitalization	*Country*	*"United States" , "united States"*
2	Misspellings	Gender	"Male" , "Mail" , "Make" , "msle"
3	Abbreviation	State	"California" , "CA"
		preparer_title	"Senior Counsel" , "Sr. Counsel"
4	Difference of Special Characters	City	"New York" , " New York, "
		Colour	"Black/Blue" , "Black-Blue"
5	Different Ordering	Colour	"GoldWhite" , "WhiteGold" , "Gold/White"
6	Synonyms	Gender	"Female" , "Woman"
		Venue	"Festival Theatre", "Festival Theater"
7	Presence of Extra Information	City	"Houston" , "Houston TX" , "Houston TX 77055"
8	Different grammar	Colour	"triColor" , "tricolored"
		Venue	"Auditorium" , "TheAuditorium"

6.3.2 Data Sources

We constructed a large real-world dataset of 9921 columns manually annotated with a standardized 9-class label vocabulary of ML feature types [135]. The classes include feature types such as *Numeric*, *Categorical*, *Datetime*, *Sentence*, and *Not-Generalizable* (e.g., primary keys). We use [135]'s dataset to obtain raw CSV files that contain at least one string *Categorical* column in order to be inspected manually for duplicates. Overall, we obtain 1248 string *Categorical* columns spanning over 217 raw CSV files.

6.3.3 Labeling Process

Among the *Categorical* columns we collected, we do not know which columns contain duplicates beforehand. This necessitates us to manually scan through all the 1248 *Categorical* columns and look for duplicates in them. We follow the below process at a column-level to reduce the cognitive load of labeling. For every *Categorical* column, we use an Excel spreadsheet to enumerate its category set along with the count of times each category appears in the column. Before scanning the category set, we sort the categories by their appearance count in descending order and their values in lexicographic order. This helps up catch the true entities early on in the file. Recall that we call the category that most frequently represents a real-world object the

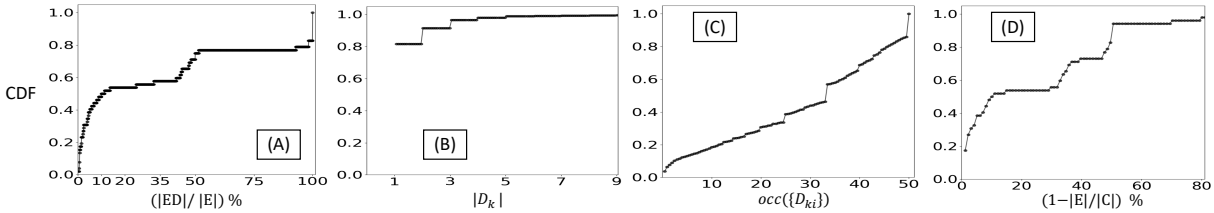


Figure 6.2: CDF over all *Categorical* columns with at least 1 duplicate on (A) percentage of entities that have at least one duplicate. (B) Duplicate set sizes over all $k \in [1, |ED|]$. The maximum duplicate set size is 148. (C) Duplicate set occurrences over all $k \in [1, |ED|], i \in [1, |D_k|]$. (D) percentage of reduction in domain size with deduplication.

true entity. As we scan the category set, we annotate duplicates with their corresponding entities in the column. Thus, we construct sets E, ED, and D, along with their occurrences for all the columns. *The entire labeling process took roughly 120 man-hours across 5 months.*

6.3.4 Types of Duplicates

We find that there exist *eight* types of duplication from our labeled dataset. We present these types with examples in Table 6.5. The differences shown are relative to the representation of the true entity. We now clarify some of the types. *Type 4* denotes the difference of any non-alphanumeric special characters including comma, period, and white spaces. *Type 5* denotes different ordering within multi-valued categories. *Type 8* categories have either a common stem/lemma, presence of stopwords, or a common singular representation. Note that a duplicate can have duplication of multiple types and an entity can have numerous duplicates, each belonging to multiple types, e.g., given ED_1= New York and $D_1 = \{$*new-york., NY*$\}$, "*new-york.*" has both *Type 1* and *4* duplication, and the entity *New York* has duplicates with duplication of *Type 1*, *3*, and *4*.

6.3.5 Data Statistics and Takeaways

We annotated 56573 entities across all 1248 string *Categorical* columns. We find 4% of those entities have the presence of at least one duplicate with a total of 3475 duplicates. Thus,

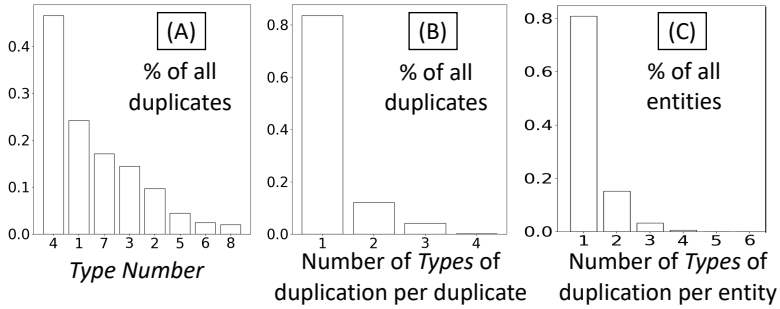


Figure 6.3: Histogram plots to illustrate how duplication types (from Table 4) arise across all columns in all files. x and y denote x-axis and y-axis respectively. (A) y% duplicates have duplication of at least one x *Type Number*. (B) y% duplicates have x different duplication types within. (C) y% entities have duplicates with x different duplication types within.

a large fraction of the entities are already clean without any duplication. Overall, 52 columns from 33 raw CSV files have the presence of at least one duplicate. There are three parameters that quantify the amount of duplication within a column. (1) Fraction of entities that have at least one duplicate ($|ED|/|E|$). (2) Duplicate set size for all entities present in the column (set D). (3) Duplicate occurrences $occ(\{D_{ki}\}), k \in [1, |ED|], i \in [1, |D|]$. Figure 6.2 plots the cumulative distribution function (CDF) of these parameters over all columns in our labeled dataset that has at least 1 duplicate. We also report CDF of the % reduction in domain size of the columns with deduplication. We summarize the presented results with key takeaways below.

(1) We first explain the worst-case scenario that can arise due to duplicates. This is critical to understand because in a real-world deployment setting, the developers have to factor in the tail percentiles when building end-to-end automation pipelines. We find that almost 17% of the columns that have duplicates have them in all of their entities! Furthermore, 7% of duplicates across all columns occur at 50%, i.e., the representation of the duplicate and the true entity are same. Additionally, 1% of all entities have more than five duplicates. However, the presence of more than 10 duplicates per entity is quite unlikely (less than 0.5%). Finally, deduplication can reduce the domain size of the *Categorical* column substantially by up to 99%. Overall, we find that whenever duplicates arise in the column, they can occur quite often.

(2) We now discuss the presence of duplicates with the average case scenario. We find

that whenever an entity is diluted with duplicates, almost 90% of the time they have one or two duplicates! Duplicate set sizes follow a long-tail distribution, most entities have small duplicate set sizes and very few entities have a lot of duplicates. This can make catching duplicates and deduplicating them particularly challenging, as they can go unnoticed. Moreover, the occurrence of duplicates approximately follows a uniform distribution, i.e., all occurrence values up to 50% are roughly equally likely. Finally, $|ED|/|E|$ values of 10-35% fall close to the median.

(3) We present how different duplication types (from Table 6.5) are represented in our labeled data in Figure 6.3. We find that the *Difference of Special Characters* and *Capitalization* issues represents the two most common types of duplicates. On the other hand, *Synonyms* and *Grammar* issues are relatively less common. Moreover, whenever duplicates exist, 17% of the time they belong to more than one duplication type (maximum observed is 4 duplication types). Also, 19% of entities have duplicates that can be mapped to multiple types (maximum observed is 6 duplication types).

6.4 Downstream Benchmark

We now empirically study the impact of category duplicates on the downstream ML tasks. We curate a downstream benchmark suite of 14 real-world datasets, each containing a column with duplicates. We use this to empirically evaluate and compare three commonly used *Categorical* encoding schemes both with and without the presence of duplicates. Finally, we make several important observations on the different confounders that impact the relationship of *Categorical* duplicates with downstream ML classifiers.

6.4.1 Models and Encodings

We choose three popular downstream classifiers used among the ML practitioners as per Kaggle data science survey [143]: `LR`, `RF`, and a two-layered MLP. These popular models also

present representative choices from the bias-variance tradeoff spectrum [74]: high bias and low variance approach with LR and low bias and high variance approaches with RF and MLP. Note that MLP's VC dimension is in between the LR and RF [138].

We encode the *Categorical* columns with three commonly used encoding schemes: (1) *One-hot* encoding (*OHE*). (2) *String* encoding (*StrE*) [119] (3) *Similarity* encoding (*SimE*) [47]. *OHE* is the standard approach to encode nominal *Categoricals* as it follows their two properties. (1) Each category is orthogonal to one another. (2) Pairwise distance between any two categories is identical. With a category set C^l (for A_l) closed during training, *OHE* sets feature vector X_l^p=$[\mathbb{1}(t^p(A_l) = C_1^l), ..., \mathbb{1}(t^p(A_l) = C_{|C^l|}^l)]$, where $\mathbb{1}(.)$ is the indicator function and $p = 1..|r|$. RF with *OHE* performs binary splits on the data. RF can also handle raw "stringified" *Categorical* values by performing set-based splits on the data. We refer to this as *StrE*. Note that *StrE* is not applicable for LR, since it cannot handle raw string values. Both *OHE* and *StrE* assume that the *Categorical* domain is closed with ML inference, i.e., new categories in the test not seen during training are handled by mapping them to a special category, "*Others*." *SimE* takes into account the morphological variations between the categories. The feature vector for category set C^l is given as $X_l^p = [Sim(t^p(A_l), C_1^l), ..., Sim(t^p(A_l), C_{|C|}^l)]$, where $Sim(.)$ is a similarity metric defined as the dice-coefficient over n-gram (n ranges from 2 to 4) strings [43]. This feature vector can be computed even for any new categories arising in test set which are unseen during training for *SimE*.

6.4.2 Real Datasets and Labeling

We collect 14 datasets from real open-source data portals such as Chicago city, New York state, California state, Pittsburgh health, Open source mental illness project, and real data surveys from FiveThirtyEight, EveryDayData, and Kaggle. Specifically, we obtain the following datasets: Midwest Survey [24], Mental Health [27], EU IT [29], Relocated Vehicles [22], Health Sciences [21], Salaries [23], TSM Habitat [19], Building Violations [39], Wifi [31], Etailing [32],

Table 6.6: Statistics of the column containing duplicates in our downstream benchmark datasets. $|r|$, $|\mathcal{A}|$, and $|Y|$ are the total number of examples, number of columns, and number of target classes in the given dataset respectively. $|rC|$ denotes the number of training examples (averaged over 5 folds) per category of the set C. P is the fraction of $|E|$ (averaged across 5-folds) that has at least 1 duplicate being mapped to "*Others*" category in the validation set with *OHE* and *StrE*. We use colors green, blue, red with hand-picked thresholds to visually present and better interpret the cases where the amount of duplication is low ($1-|E|/|C| < 0.25$), moderate ($1-|E|/|C| > 0.25$ & < 0.50), and high ($1-|E|/|C| > 0.50$) respectively. We use the following thresholds with the same colors to better interpret the data regime: low ($|rC| < 5$), moderate ($|rC| > 5$ & < 25), and high ($|rC| > 25$). Note that the data regime moves up with deduplication as category set size has shrunk.

Datasets	|r|	$\|\mathcal{A}\|$	$\|Y\|$	Amount of Duplication					Data Regime		P %
				$\frac{\|ED\|}{\|E\|}$ %	median $\|D_k\|$	median occ($\{D_{ki}\}$)	|C|	1-$\frac{\|E\|}{\|C\|}$ %	|rC|	|rC| after dedup (Increase w.r.t Raw)	
Midwest Survey	2778	29	9	33.1	2	4	1008	64	2.5	6.5 (2.6x)	23.6
Mental Health	1260	27	5	40	3.5	2.3	49	69.4	23.2	81.2 (3.5x)	25.3
Relocated Vehicles	3263	20	4	33.2	1	20	1097	35.8	2.5	3.8 (1.5x)	14.9
Health Sciences	238	101	4	36.4	2	6	56	60.7	3.6	8.3 (2.3x)	26.4
Salaries	1655	18	8	24	1	25	647	29.2	1.8	2.2 (1.2x)	10.9
TSM Habitat	2823	48	19	11	1	25	912	11.4	2.6	2.9 (1.1x)	14.6
EU IT	1253	23	5	24	1.5	12.5	256	34.8	3.9	5.9 (1.5x)	19.5
Halloween	292	55	6	31.3	2	11.1	163	50.9	1.5	3 (2x)	22.8
Utility	4574	13	95	38.4	1	20	199	30.7	16.2	24.3 (1.5x)	6.2
Mid or Feed	1006	78	5	21.4	6	0.8	37	62.2	20.6	59.7 (2.9x)	24.3
Wifi	98	9	2	30.3	2.5	12.5	69	52.2	1.3	2.5 (1.9x)	26.1
Etailing	439	44	5	47.8	4	5.9	71	67.6	5.3	14.3 (2.7x)	28.7
San Francisco	148654	13	2	10.7	1	25	2159	9.8	46.3	50.9 (1.1x)	3.2
Building Violations	22012	17	6	51	2	4.8	270	63	53.7	145 (2.7x)	4.4

Halloween [26], Mid or Feed [30], San Francisco [28], and Utility [33]. Each dataset has a column with duplicates. We manually deduplicate them by following the labeling process described in Section 6.3.3. We leave automating deduplication task with a learning-based approach using our hand-labeled data to future work. To make sure that an upstream deduplication model is not evaluated on the same data it was trained with, we keep the downstream dataset separate from the hand-labeled data.

Table 6.6 presents the statistics with different confounders that can potentially impact the ML performance over four of our datasets. There are four data-dependent confounders that can potentially impact the ML performance. (1) Three parameters characterizing duplicates: $|ED|/|E|$,

Table 6.7: Classification accuracy comparison of downstream models with different *Categorical* encoding schemes on *Raw* (column with duplicates) vs. *Deduped* (deduplicated column) data. Accuracy results for *Deduped* are shown relative to the *Raw* as delta drop in % accuracy. Green, blue, and red colors denote cases where the *Deduped* accuracy relative to *Raw* is significantly higher, comparable, and significantly lower (error tolerance of 1%) respectively.

Dataset	Logistic Regression (LR)						Random Forest (RF)							
	Relevancy OHE		*OHE*		*SimE*		*Relevancy OHE*		*OHE*		*StrE*		*SimE*	
	Raw	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*
Midwest Survey	10.6	+9.4	57.2	+9.4	66.7	+2.1	4.6	+11.5	49.1	+11.5	59.2	+10	64.9	+4.4
Mental Health	-1.3	+1.3	46.9	+1.3	46.3	+0.6	0.2	+1.1	47.9	+1.1	47.8	-0.1	47.4	-1.7
Relocated Vehicles	18.1	+4	82.9	+4	88.4	+0.4	6.1	+3	72.5	+3	81.3	+4.1	88.3	-0.1
Health Sciences	-1.3	+0.9	58.7	+0.9	60	+1.8	-1.8	+2.2	53.3	+2.2	61.8	+0	60	-2.7
Salaries	-1.1	+0.1	30.4	+0.2	32.4	-1.3	-1	+1.7	64.7	+1.7	69.6	+1.3	94.6	+0.4
TSM Habitat	0	+0	50.7	+0	50.7	+0	4.8	+0.4	71.2	+0.4	84.1	+1.4	71.2	+0.4
EU IT	0	+0	29.1	+0	29.1	+0	2.1	+1.2	41.2	+1.2	43.6	-0.6	47.8	+4
Halloween	0.4	+3.4	42.6	+3.4	49.8	+1.1	-1.9	+1.5	40	+1.5	36.2	+1.5	34.7	-4.9
Utility	1.4	-0.2	42.4	-0.2	43	+0.3	-6.7	+1.4	58.8	+1.4	46.3	+1.2	43.2	+1.4
Mid or Feed	0	+1.7	40.5	+1.7	41.5	-1.2	-1	+2.5	40.2	+2.5	35.7	-0.2	36.2	+1.8
Wifi	-2.1	+1.1	64.2	+1.1	58.9	+8.4	-1.1	+5.3	60	+5.3	57.9	+4.2	50.5	+3.2
Etailing	0.7	-0.5	41.1	-0.5	38.9	+1.8	-2.5	+2	40	+2	44.5	+1.1	38.2	+3
San Francisco	26.9	-0.1	86	-0.1	85.5	+0	24.3	+0.1	83.4	+0.1	83.9	-0.3	86	+0
Building Violations	0.1	+0	91.6	+0	91.9	+0	0	-0.1	97.5	-0.1	97.3	+0.1	97.6	+0

$|D_k|$, $occ(D_k)$. We use the quantity % reduction in domain size with deduplication (1-$|E|/|C|$) to summarize the amount of duplication. (2) Data regime relative to the complexity of the prediction task. We simplify it as the number of training examples per category value ($|rC|$). We ensure that our selected datasets sufficiently represent different ranges of values (high vs. low measured relatively) in each confounder spectrums. This allows us to make empirical observations by assessing their significance in Section 6.4.4. We use simulation study (Section 6.5) to disentangle the impact individually with all confounders. We hope our work inspires more data benchmark standardization in this space with industry involvement.

6.4.3 Methodology

We partition each dataset into an 80:20 split of train and test set. We perform 5-fold cross-validation and use a fourth of the examples in the train set for hyper-parameter search. We tune the regularization parameter for `LR`. We tune the number of trees and their maximum depth

for RF with values for each ranging from 5 to 100. The MLP architecture comprises of 2 hidden units with 100 neurons each and is L2 regularized.

Experimental Setup. We use CloudLab [61] with custom OpenStack profile running Ubuntu 18.04, 10 Intel Xeon cores, and 160GB of RAM. We use python scikit-learn [115], H2O [11], and SimilarityEncoder [15] packages to implement *OHE*, *StrE*, and *SimE* respectively. We use a standard grid search for hyper-parameter tuning, with the grids described in detail below.

Logistic Regression: There is only one regularization parameter to tune: C. Larger the value of C, lower is the regularization strength, hence increasing the complexity of the model. The grid for C is set as $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100, 10^3\}$.

Random Forest: There are two hyper-parameters to tune: *NumEstimator* and *MaxDepth*. *NumEstimator* is the number of trees in the forest. *MaxDepth* is the maximum depth of the tree. The grid is set as follows: *NumEstimator* $\in \{5, 25, 50, 75, 100\}$ and *MaxDepth* $\in \{5, 10, 25, 50, 100\}$.

MLP: The multi-layer perceptron architecture comprises of 2 hidden units with 100 neurons each. We do L_2 regularization with the regularization parameter tuned using the following grid axis: $\{10^{-4}, 10^{-3}, 10^{-2}\}$.

6.4.4 Results

Table 6.7 and 6.8 shows the end-to-end comparison of the downstream ML models built with different encoding schemes in terms of diagonal accuracy. As an example, on *Midwest Survey*, RF with *OHE* of *Categoricals* delivers a 9-class classification accuracy of 49.1% on the *Raw* dataset. Cleaning its duplicates (*Deduped*) lead to an 11.5% lift in accuracy relative to the *Raw*. Table 6.9 shows summary statistics of how the different encoding schemes perform with the two ML models and also relative to one another on the 14 datasets. Finally, we present the generalization performance of the ML classifiers with the overfitting gap (difference between

Table 6.8: Downstream accuracy comparison of (high-capacity) MLP with different *Categorical* encoding on *Raw* vs. *Deduped*.

Dataset	Relevancy *OHE* on *Raw*	OHE		SimE	
		Raw	*Deduped*	*Raw*	*Deduped*
Midwest Survey	16.4	54.7	+9.5	63.4	+3.8
Mental Health	-3.8	42.4	+2	43.2	-0.4
Relocated Vehicles	21.9	83.6	+3.6	89.6	+0
Health Sciences	-4	55.1	+4.9	56.4	+1.8
Salaries	-5.6	22	+0.5	19.9	+5.4
TSM Habitat	0.1	50.7	-2.7	50.7	-2.7
EU IT	6.9	13.4	-2.4	6.8	+5
Halloween	-1.5	41.9	+4.2	43	+0.8
Utility	16.1	65.1	+2.3	73.2	+2.5
Mid or Feed	-0.5	34	+2	32.7	+0.2
Wifi	-6.3	52.6	+2.1	48.4	+3.2
Etailing	0.2	40.2	-3	37.2	+0
San Francisco	26.9	86	+0.1	86.1	-0.1
Building Violations	0.1	97.2	+0	97.4	+0

train and validation set accuracies) on both *Raw* and *Deduped* in Table 6.10. We summarize our results with *eight* important observations below.

O1. We find that there exist several downstream cases where *Deduped* improves the accuracy of ML over *Raw* for any encoding scheme. For instance, the delta increase in accuracy with *Deduped* on RF with *OHE* is of median 1.6% and up to 11.5% compared to *Raw* (across 14 datasets). Moreover, the delta increase in accuracy is of median 2% and up to 9.5% for MLP.

O2. Delta increases in accuracies with *Deduped* are typically higher with RF and MLP than LR. The median delta increases in accuracy with RF and MLP using *OHE* are 1.6 and 2, compared to 0.6 for LR. Thus, LR is more robust to duplicates than both high-capacity classifiers, RF and MLP.

O3. *Deduped* helps RF using *OHE* the most, *StrE* the second most, and *SimE* the least (see Table 6.9). Interestingly, the median lifts in accuracies due to deduplication with *SimE* are just 0.4 and 0.5 on RF and MLP respectively. Overall, *SimE* improves the ML performance in just ˜40% downstream cases. This is because, *SimE* considers morphological variations between the category strings and maps a duplicate to a similar feature vector as the true entity. So, duplicates

Table 6.9: Summary statistics to illustrate the impact of deduplication on ML models using different encoding schemes with 14 downstream datasets. $*$ and † denote two and one cases where both encoding schemes perform the best resp.

	LR		Random Forest (RF)			MLP	
	OHE	***SimE***	***OHE***	***StrE***	***SimE***	***OHE***	***SimE***
% lift in accuracy with Deduped							
Mean	1.5	1	2.4	1.7	0.7	1.7	1.4
Median	0.6	0.4	1.6	1.2	0.4	2	0.5
75th percentile	1.6	1.6	2.4	1.5	2.7	3.3	3
Max	9.4	8.4	11.5	10	4.4	9.5	5.4
# downstream datasets where							
>1% lift in accuracy on *Deduped*	6	5	11	8	6	8	6
Best performing encoding on *Raw*	6*	10*	5	3	6	6†	9†
Best performing encodin on *Deduped*	5*	11*	5	3	6	8*	8*

are often located close to their true entities in the feature space. Thus, any further lift in accuracy due to deduplication is marginal.

O4. Deduplication reduces the overfitting gap for all models (from Table 6.10). Thus, deduplication can reduce the variance component of the test error and improve the generalization ability of the classifiers. Since `RF` and `MLP` are more prone to overfitting than `LR`, their lifts in accuracies with *Deduped* are more significant. This also explains the observation *O2.*

O5. If the magnitude of overfitting gap on *Raw* is insignificant (less than 1%), then the amount of possible reduction in overfitting with *Deduped* is also small. Thus, it's not worthwhile to deduplicate if the overfitting gap on *Raw* is already low, to begin with. We observe this will all the datasets where the overfitting gap is close to 1%, e.g., *San Francisco* and *Building Violations*. We observe this across the three downstream classifiers.

O6. Deduplication increases the column *Relevancy* for all models, i.e., the column becomes more predictive for the downstream tasks after deduplication. Note that the amount of increase in column *Relevancy* with *Deduped* also quantifies the accuracy lift with *Deduped*.

O7. The accuracy lifts with *Deduped* on all the models are more significant when the

Table 6.10: Delta drop in % overfitting gap in accuracy with *OHE*. The overfitting gap for *Deduped* is shown relative to the *Raw*.

Dataset	LR		Random Forest (RF)		MLP	
	Raw	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*
Midwest Survey	24.4	-9.4	50.7	-14.2	45.1	-10.4
Mental Health	11.7	-3.5	42.3	-7.2	26.7	-0.2
Relocated Vehicles	17	-4.1	27.3	-3.1	16.4	-3.6
Health Sciences	9.3	-5.9	35	-8.1	44.9	-4.9
Salaries	1.9	+0.2	34.6	-1	1.4	-0.5
TSM Habitat	1.9	-0	28	-0	0.1	+0.5
EU IT	1.2	-0	53.1	-6.6	1.4	+0.9
Halloween	38.3	-3.5	50.9	-5.8	58.1	-4.2
Utility	0.7	-0.3	41.2	-1.4	26.1	-3
Mid or Feed	34.2	-12.8	58.4	-1.1	66	-2
Wifi	11.1	-2.1	26.2	+1.3	47.4	-2.1
Etailing	41.2	-7.7	54.4	-1.6	59.7	+2.9
San Francisco	0.5	-0	-0.2	-0	1.1	-0.1
Building Violations	0.2	+0.1	1.8	-0.1	1.1	-0.2

column has high *Relevancy* unless there exist a high-data regime (a large number of training examples per category). Thus, if a column has already high *Relevancy* on *Raw*, to begin with, it may be worthwhile conservatively to deduplicate, e.g., *Relocated Vehicles* and *Midwest Survey.*

O8. High-data regime is robust to the impact of duplicates than the low-data regime, regardless of the amount of duplication. Even a high amount of duplication has a negligible impact in the high-data regime, e.g., *Building Violations* has a massive 63% reduction in domain size due to deduplication, but there exist a large number of training examples per category. We do not see any lift in accuracy with deduplication on any of the ML models.

Results with Additional Evaluation Metrics We also rerun our downstream benchmark suite using additional evaluation metrics such as macro/micro average of precision, recall, and F1-score. We check if using these additional evaluation metrics alter any empirical observations or conclusions in Section 6.4.4. Note that the micro average of precision, recall, and F1-score is identical to the accuracy of multi-class classification [69], which we already reported in Tables 6 and 7. Thus, we use the macro average of precision, recall, and F1-score [69] as evaluation

Table 6.11: Comparison of downstream models in terms of Macro F1 score with different *Categorical* encoding schemes on *Raw* (column with duplicates) vs. *Deduped* (deduplicated column) data. Results for *Deduped* are shown relative to the *Raw* as delta drop in % F1 score. Green, blue, and red colors denote cases where the *Deduped* F1 score relative to *Raw* is significantly higher, comparable, and significantly lower (error tolerance of 1%) respectively.

Dataset	Logistic Regression				Random Forest						MLP			
	OHE		*SimE*		*OHE*		*StrE*		*SimE*		*OHE*		*SimE*	
	Raw	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*
Midwest Survey	55.7	+10.3	65.4	+2.7	44.9	+12.6	56.5	+11.7	63.4	+5	54.3	+9.7	63.3	+3.7
Mental Health	42	-0.6	40.1	+0.8	40.3	+0	39.3	-1.3	38	+0.8	39.3	+2.7	41.1	+0.5
Relocated Vehicles	82.8	+4	88.4	+0.4	71.6	+3.5	81.3	+3.7	88.3	-0.1	83.6	+3.6	89.5	+0
Health Sciences	56.1	+0.7	57.4	+2.2	51.5	+3.3	59.1	-0.5	59.2	-3.7	54.7	+5.4	56.6	+1.8
Salaries	27.4	+1.3	30.5	-2	57.6	+2.1	64.5	+1.9	93.8	+0.4	15.9	+3.4	14.7	+8.7
TSM Habitat	34.1	+0	34.1	+0	68.5	+0	82.2	+1.6	85.7	+0.6	24.6	+4	19.1	+12.3
EU IT	16.1	+0	16.1	+0	33.6	+1.1	36.8	+0.2	43.1	+2.7	9.3	-2.2	4.2	+4.8
Halloween	37.1	+3.8	45.7	+0.5	34.2	+3	33.2	+1.7	31.1	-4.9	38.8	+4.1	40.9	-0.6
Utility	37	+0.1	38.5	+0.3	58.2	+1.5	44.9	+1.4	41.8	+1.7	65.2	+2	73.4	+2.2
Mid or Feed	37.2	+0.2	39.1	-3.6	35.2	+1.8	26.6	-0.3	26.1	+2.6	33	+1.9	31.2	+0.4
Wifi	54.9	+1.5	50.7	+8.2	52.7	+8.5	54.3	-4.5	50.2	+1.9	51.6	+2.3	48.3	+2.6
Etailing	37.2	-2.3	37.5	-0.1	33.3	+3	36.3	+1.4	32.9	+3.1	39.4	-3.1	36.7	+0
San Francisco	85.9	-0.1	85.6	-0.1	83	+0.3	83.3	+0.3	86.1	-0.1	86	+0.1	86	+0.1
Building Violations	89.4	+0	89.3	+0.1	97.5	-0.1	97	+0.1	97.4	+0.1	97.5	+0.1	97.2	+0.4

metrics and rerun our downstream benchmark suite. We check if evaluating with macro F1 score alters the conclusion made with % diagonal classification accuracy as the metric in regard to the varied confounder.

Overall, we find that none of the empirical conclusions made with diagonal accuracy change even with these additional evaluation metrics. Table 6.11 presents the comparison of downstream models with different *Categorical* encoding schemes in terms of macro F1 scores. Table 6.12 (similar to Table 8) showcases the aggregate statistics over all evaluation metrics. Finally, we present the generalization performance of the ML classifiers with the overfitting gap, difference between train and validation macro F1 scores in Table 6.13 here (similar to Table 9). We confirm the validity of all observations *O1-O8* made with the downstream benchmark suite with the additional evaluation metrics. As an example, we still find that the delta increases in macro F1-score, precision, and recall with *Deduped* are higher with Random Forest and

Table 6.12: Summary statistics over 14 downstream datasets in terms of Macro F1-score, precision, and recall to showcase the impact of deduplication on ML models.

F1 score	LR		Random Forest			MLP	
	OHE	*SimE*	*OHE*	*StrE*	*SimE*	*OHE*	*SimE*
	% lift in accuracy with Deduped						
Mean	1.4	0.7	2.9	1.4	0.7	2.4	2.6
Median	0.2	0.2	2	1.4	0.7	2.5	1.2
75th percentile	1.5	0.7	3.2	1.9	2.5	3.9	3.4
Max	10.3	8.2	12.6	11.7	5	9.7	12.3
	# downstream datasets where						
>1% lift wrt metric on *Deduped*	5	3	10	8	6	10	7

Precision	LR		Random Forest			MLP	
	OHE	*SimE*	*OHE*	*StrE*	*SimE*	*OHE*	*SimE*
	% lift in accuracy with Deduped						
Mean	1.8	1.1	4.2	1.2	0.7	1.7	1.9
Median	0.9	0.3	3.2	1	0.5	1.9	1.7
75th percentile	2.9	2.1	6.2	1.9	2.8	3.4	2.8
Max	10.3	8.1	14.7	11.9	5.8	9.8	9.8
	# downstream datasets where						
>1% lift wrt metric on *Deduped*	7	4	10	7	6	9	8

Recall	LR		Random Forest			MLP	
	OHE	*SimE*	*OHE*	*StrE*	*SimE*	*OHE*	*SimE*
	% lift in accuracy with Deduped						
Mean	1.6	0.9	2.4	1.9	0.7	2.3	2.7
Median	0.9	0.4	1.6	1.3	0.6	2.1	1.3
75th percentile	1.6	1.1	2.4	2.1	2.7	4.1	3.7
Max	9.4	8.4	10.8	10	4.4	9.5	15
	# downstream datasets where						
>1% lift wrt metric on *Deduped*	7	4	10	9	6	9	7

Multi-layer Perceptron (MLP) than Logistic Regression. Moreover, we again find that *Similarity* encoding is more robust than other encoding schemes to tolerate duplicates. Note that the focus of this work is on interpreting the impact on ML with features having different duplication properties and not on disentangling the impact at a per-class basis.

Overall, there exist *six* confounders that can potentially impact the downstream ML: $|ED|/|E|, |D_k|, occ(D_k)$, $|rC|$, column *Relevancy*, and fraction of entities that have duplicate(s) being mapped to "Others" along with their occurrences in the unseen test set. Beyond our observations, there exists a non-trivial interaction of the *six* confounders that impact ML. Thus, we use the simulation study in the next section to disentangle and study them separately.

6.5 In-depth Simulation Study

We now use a Monte Carlo-style simulation study to dive deeper into the impact of each confounder on the downstream ML. This study helps us not only validate our empirical

Table 6.13: Delta drop in % macro F1 overfitting gap of the ML models with *OHE*. The overfitting gap for *Deduped* is shown relative to the *Raw*.

Dataset	LR		Random Forest		MLP	
	Raw	*Deduped*	*Raw*	*Deduped*	*Raw*	*Deduped*
Midwest Survey	25.3	-10	55	-15.8	45.5	-10.6
Mental Health	12.5	-3.4	49.8	-7.9	28.9	+1.5
Relocated Vehicles	17	-4.1	28.2	-3.7	16.4	-3.6
Health Sciences	7.5	-6.5	35.1	-8.6	45.3	-5.4
Salaries	2.3	-0.8	41.6	-1.4	0.8	+0.6
TSM Habitat	2.1	-0	30.6	-4.8	1.3	+0.2
EU IT	0.1	-0	60	-6.8	1	+1.1
Halloween	40.4	-3.8	56.2	-7.2	61.2	-4.1
Utility	0.1	-0.9	41.8	-1.5	26	-2.9
Mid or Feed	35.7	-12.4	63.3	-0.4	67	-1.9
Wifi	11.9	-2.6	31.8	-4.8	46.8	+0.8
Etailing	43.3	-6.7	60.6	-2.3	60.6	+3
San Francisco	0.6	-0.2	0.1	+0.1	1.1	-0.1
Building Violations	0.1	+0.1	1.8	+0.1	1.2	-0.2

observations but also makes the significance of each confounder impacting ML more interpretable. Moreover, it reveals the limitations of commonly used encoding schemes when unseen duplicates arise in the test set.

Encoding Schemes. We focus this study in the context of *OHE* and *StrE*. Note that *OHE* representation masks out the actual category values by treating each category as an orthogonal representation to one another. In contrast, *SimE* require the categories to be semantically meaningful strings. We find from Table 6.5 and Figure 6.3 that an entity can have duplication of multiple types. Constructing a fine-grained simulator that generates semantically meaningful duplicates while preserving the same ground truth entity is non-trivial and intricate from the language standpoint. Thus, we leave designing an apt simulation mechanism for *SimE* to future work.

Downstream Models. The structural model parameters such as the number of tree estimators and maximum allowed tree depth for `RF` and the specific `MLP` architecture can largely impact the bias-variance tradeoff. Thus, we fix them to disentangle their impact and better

illustrate our findings by presenting two extremes of RF's and MLP's bias spectrum. We use high-bias models such as shallow decision tree with a restricted tree depth of 5 (denoted as ShallowDT), a low-capacity MLP comprising of two hidden units with 5 neurons each (denoted as LoCapMLP), and also LR. In addition, we use low-bias high-capacity RF with the number of tree estimators and maximum allowed tree depth being fixed to 50 (denoted as HiCapRF). These values represent the median best-fit parameters obtained by performing a grid search (with the grids being same as Section 6.4.3) over the synthetically generated data described in Section 6.5.1. We again use a high-capacity MLP comprising of two hidden units with 100 neurons each (HiCapMLP).

Setup. There is one relational table with Y being boolean (domain size is 2). We include three *Categorical* columns in the table and set $|\mathcal{A}|$ to 3. We set the entity set size of every columns to $|E| = 10$, i.e., all columns have a domain size of 10.

Data Synthesis. We set up a "true" distribution $P(\mathcal{A}, Y)$ and sample examples in an independently and identically distributed manner. We study two different simulation scenarios: AllA and Hyperplane. These scenarios represent two opposite extremes of how RF and hyperplane-based classifiers fits the data. AllA represents a complex joint distribution where all features obtained from $\mathcal{A}$ determine Y. Although a high-capacity model such as RF can recover this with rule-based splits, LR can extremely underfit. Hyperplane represents a distribution where the data is simply separable with a hyperplane. This distribution is well-suited for LR and MLP (where each neuron defines a hyperplane), but represents a bad-case scenario for RF that requires many numbers of splits to recover the true concept. We sample $|r|$ number of total examples, where the examples for training, validation, and test are split in 60:20:20 ratio. We then introduce synthetic duplicates in one of the columns of the table in different ways. We vary the *six* confounders one at a time and study their impact on ML accuracy along with how they trend as the parameter is varied. We generate 100 different (clean) training datasets and 10 different dirty datasets for every clean one. We measure the average test accuracy and the average overfitting gap of all models obtained from these 1000 runs.

6.5.1 Scenario AllA

Data generating process. We create a true distribution that maps all features obtained from $\mathcal{A}$ to Y. The exact sampling process is as follows. (1) Construct a conditional probability table (CPT) with entries for all possible values of $\mathcal{A}$ from 1 to $|E|$. We then assign $P(Y=0|\mathcal{A})$ to either 0 or 1 with a random coin toss. (2) Construct $|r|$ tuples of $\mathcal{A}$ by sampling values uniform randomly from $|E|$. (3) We assign Y values to tuples of $\mathcal{A}$ by looking up into their respective CPT entry. (4) We perform the train, validation, and test split of this clean dataset and obtain the binary classification accuracy of the ML models on the test split.

Duplication process. We introduce duplicates in a column $A_l \in \mathcal{A}$ of the clean dataset in the following fashion. (1) Fix fraction of entities to be diluted with duplicates, e.g., $|ED|/|E|$=30 (2) Form set ED (set of entities that are to be diluted with duplicates) by sampling uniformly randomly $|ED|$ categories from $|E|$ (E_1 to $E_{|E|}$), e.g., ED=$\{E_3, E_5, E_8\}$. (3) For every entity in set ED, fix the duplicate set size $|D_k|, \forall 1 \leq k \leq |ED|$, e.g., $|D_k|$=1,$\forall 1 \leq k \leq 3$. We assume that all entities have identical duplicate set sizes. We relax this assumption later when introducing skew in the duplication parameters. (4) Given $|D_k|$, we form the set D by introducing duplicates, e.g., D_1=$\{E_3$-$duplicate_1\}, D_2$=$\{E_5$-$duplicate_1\}, D_3$=$\{E_8$-$duplicate_1\}$. (5) Fix $occ(D_k), 1 \leq k \leq |ED|, 1 \leq i \leq |D_k|$. For every duplicate value d in D, set occurrence $occ(d)$=$occ(D_k)/|D_k|$, i.e., we assume that all the duplicates representing an entity are equally likely to occur. We also relax this assumption later. (6) We perform the same train, validation, and test split of the resulting dataset as obtained in step 4 of the data generating process. We finally obtain the test accuracy of the ML models on the dirty dataset.

Results. We use our labeled data to pick appropriate values for the confounders such that we can showcase an average and worst-case scenario. We vary all confounders one at a time while fixing the rest. We confirm the trends and observations made with *italics.*

Varying the data regime. Figure 6.4 (A) presents the delta drop in %accuracy with duplication relative to the ground truth clean dataset on `HiCapRF` as the number of training

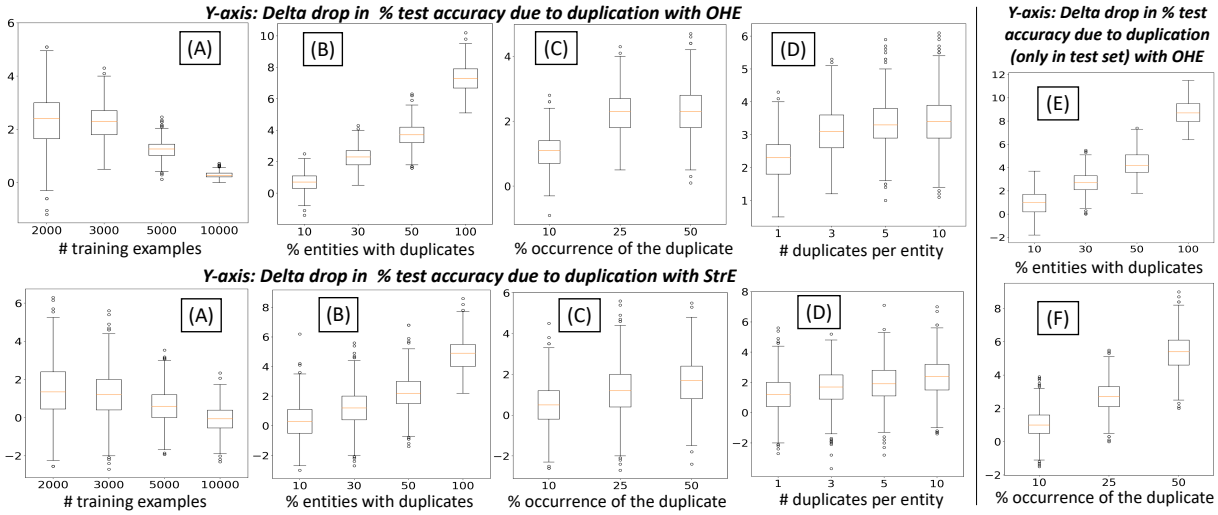


Figure 6.4: `AllA` scenario results for `HiCapRF` with *OHE* and *StrE*. (A-D) Duplicates present in train, validation, and test set. (E-F) Only test set is diluted with duplicates. (A) Vary $|r|_t$ (# training examples) while fixing $(|ED|/|E|, occ(D_k), |D_k|) = (30, 25, 1)$ (B) Vary $|ED|/|E|$ while fixing $(|r|_t, occ(D_k), |D_k|) = (3000, 25, 1)$ (C) Vary $occ(D_k)$ while fixing $(|r|_t, |ED|/|E|, |D_k|) = (3000, 30, 1)$ (D) Vary $|D_k|$ while fixing $(|ED|/|E|, |r|_t, occ(D_k)) = (30, 3000, 25)$, for all $k \in [1, |ED|]$. Parameter settings of (E) and (F) are same as (B) and (C) resp.

examples ($|r|_t$) are varied with both *OHE* and *StrE*. We find that with the rise in $|r|_t$, the delta drop in accuracy decreases. With just 3 training examples per CPT entry ($|r|_t$=3k and total entries in CPT=1k), the presence of duplicates can cause a drop of median 2.3% and up to 4.3% accuracy with *OHE*. With 10 training examples, the median and max drops in accuracy due to duplicates with *OHE* are 0.3% and 0.7% respectively. This confirms our observation on the downstream benchmark suite: *A higher data regime is more robust to duplication than a lower data regime. The same trend holds with StrE encoding and also all the other classifiers:* `LR`, `ShallowDT`, `LoCapMLP` *and* `HiCapMLP`. *Thus, a high-data regime can tolerate duplicates by remaining more agnostic to the model qbiases.* Moreover, increasing the amount of duplication for a high data regime ($|r|_t$ = 10k) has a marginal impact on the accuracy. *Thus, even high duplication has a marginal impact in the high-data regime.*

Varying the parameters that control the amount of duplication. Figure 6.4 (B-D) shows how different duplication parameters influence `HiCapRF`. We notice a clear trend: *the drop*

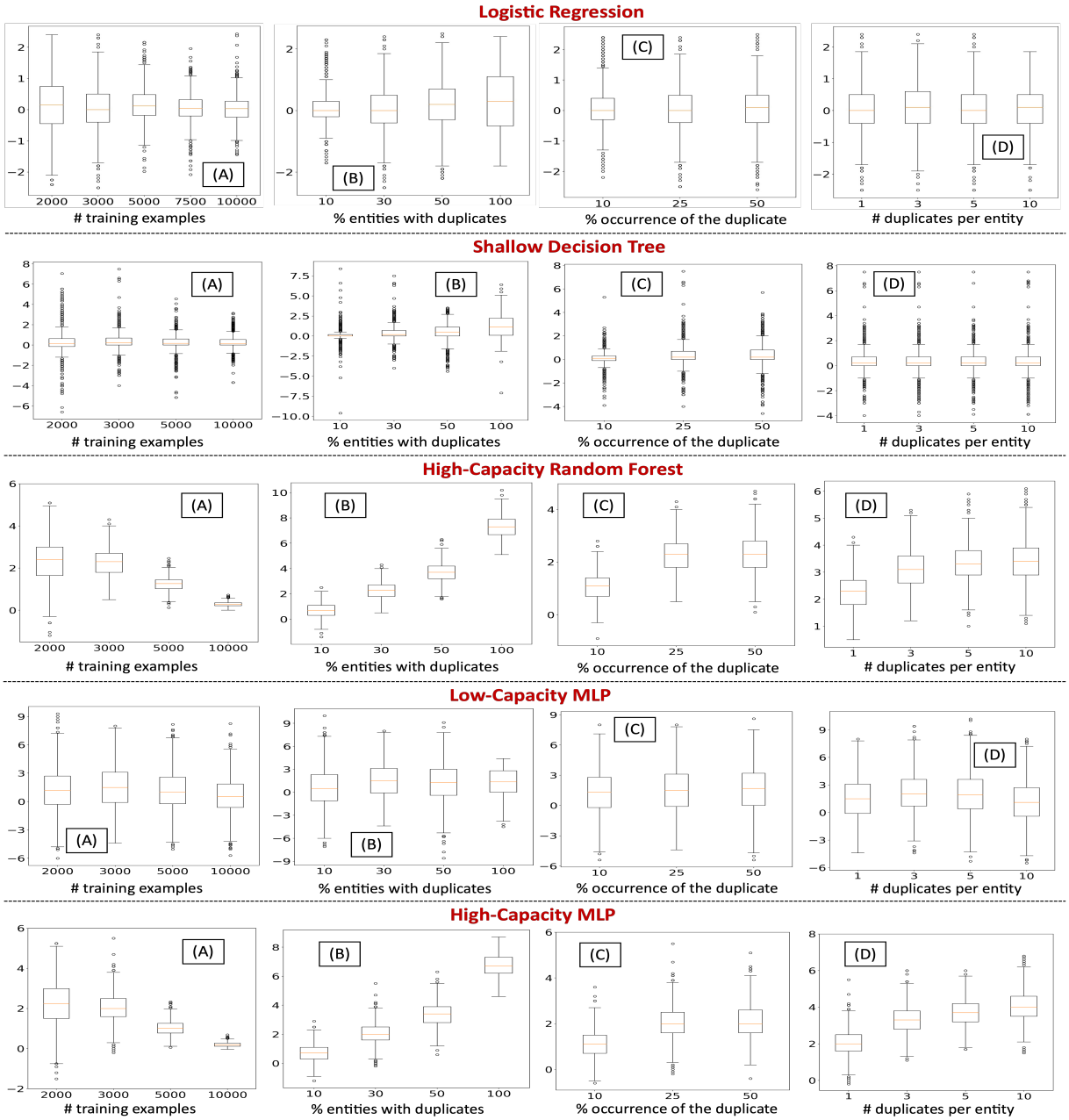


Figure 6.5: `AllA` simulation scenario results for `LR`, `ShallowDT`, `HiCapRF`, `LoCapMLP`, and `HiCapMLP` with *OHE*. $|X| = 3$. (A) Vary $|r|_t$ (# training examples) , while fixing $(|ED|/|E|, occ(D_k), |D_k|) = (30, 25, 1)$. (B) Vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|) = (3000, 25, 1)$. (C) Vary $occ(D_k)$, while fixing $(|r|_t, |ED|/|E|, |D_k|) = (3000, 30, 1)$. (D) Vary $|D_k|$, while fixing $(|ED|/|E|, |r|_t, occ(D_k)) = (30, 3000, 25)$, for all $k \in [1, |ED|]$.

in accuracy with `HiCapRF` *rises with the increase in any of the three duplication controlling parameters,* $|ED|/|E|$, $occ(D_k)$, *and* $|D_k|$. As $|ED|/|E|$, $occ(D_k)$, and $|D_k|$ are each increased 5

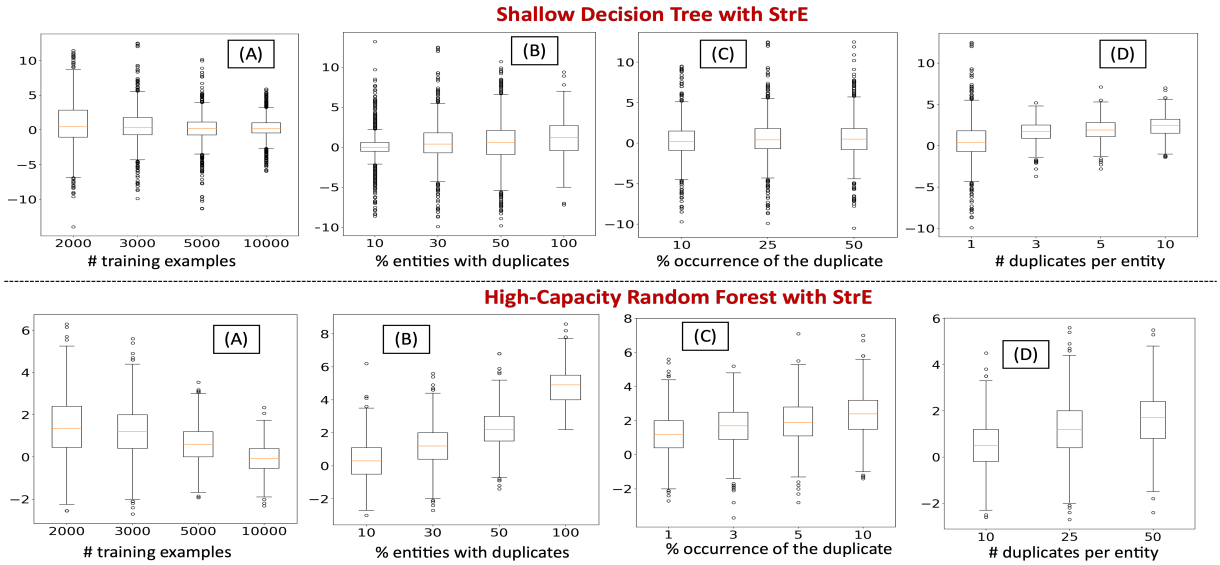


Figure 6.6: `AllA` simulation scenario results for `ShallowDT` and `HiCapRF` with *StrE*. (A) Vary $|r|_t$ (# training examples) , while fixing $(|ED|/|E|, occ(D_k), |D_k|) = (30, 25, 1)$. (B) Vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|) = (3000, 25, 1)$. (C) Vary $occ(D_k)$, while fixing $(|r|_t, |ED|/|E|, |D_k|) = (3000, 30, 1)$. (D) Vary $|D_k|$, while fixing $(|ED|/|E|, |r|_t, occ(D_k)) = (30, 3000, 25)$, for all $k \in [1, |ED|]$.

folds, the magnitude increase in delta drop accuracy with duplicates using *OHE* are $5.3x, 2.1x$, and $1.4x$ respectively. In contrast, the same magnitude increases using *StrE* are $7.3x, 3.4x$, and $1.6x$ respectively. Thus, among the three duplication parameters, $|ED|/|E|$ has the most drastic effect on `HiCapRF`. The effects of the increase in $|D_k|$ are less pronounced because all other parameters including $occ(D_k)$ are kept fixed. Thus, there exist more duplicates for the same occurrence. *Interestingly, we find from Figure 6.4 that StrE is more robust to duplicates than OHE regardless of the parameter being varied, as the delta drop in accuracy with StrE is comparatively lower, although significant in high duplication cases.*

Figure 6.5 presents how a key confounder ($|ED|/|E|$) affects the other studied classifiers. We find that all the high-bias models behave similarly as they show a marginal drop in accuracy even when all the entities are diluted with duplicates. In contrast, `HiCapMLP` exhibits similar behavior as `HiCapRF` when $|ED|/|E|$ is increased. Note that the absolute accuracies of the high-bias approaches are lower than that of high-capacity ones. *Overall, both high-capacity classifiers are more susceptible to the adverse performance impact of duplicates than the high-*

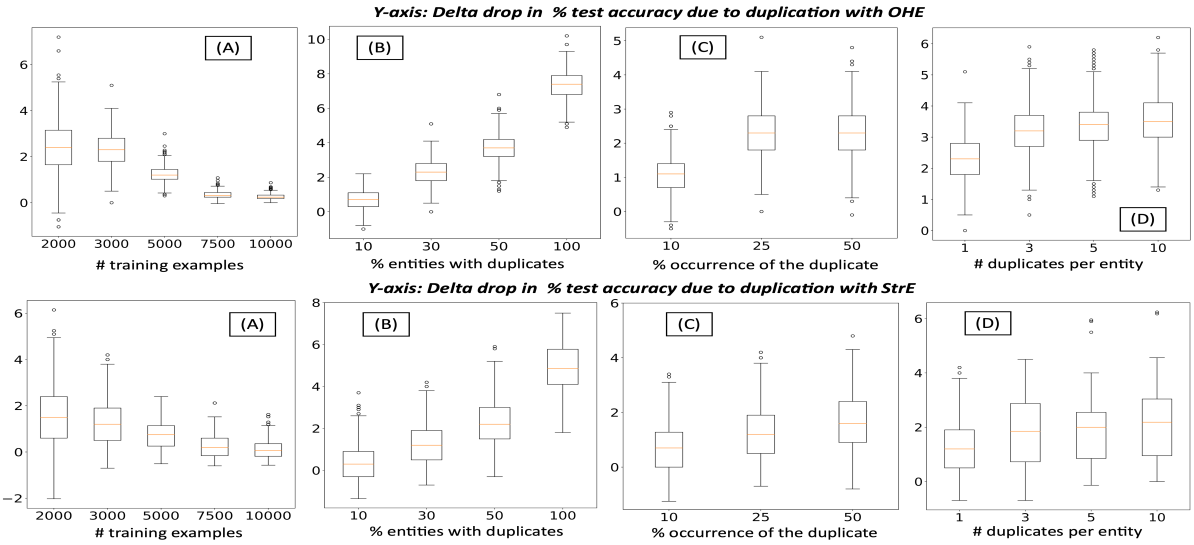


Figure 6.7: `AllA` scenario results for Random Forest with *OHE* and *StrE* where hyper-parameters are tuned with grid search. $|X| = 3$. (A) Vary $|r|_t$ (# training examples) , while fixing $(|ED|/|E|, occ(D_k), |D_k|) = (30, 25, 1)$. (B) Vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|) = (3000, 25, 1)$. (C) Vary $occ(D_k)$, while fixing $(|r|_t, |ED|/|E|, |D_k|) = (3000, 30, 1)$. (D) Vary $|D_k|$, while fixing $(|ED|/|E|, |r|_t, occ(D_k)) = (30, 3000, 25)$, for all $k \in [1, |ED|]$.

bias approaches. We notice the same trend as the other confounders ($occ(D_k)$ and $|D_k|$) are varied. Figure 6.6 shows the results as the confounders are varied for `HiCapRF` with *StrE*. Figure 6.7 shows the same results when the Random Forest parameter are tuned using grid search. We again note that a high-data regime is more robust to duplication than a low-data regime for both encoding schemes.All the high-bias approaches are more robust to duplication than the high-capacity models. Also, duplication confounders can have significant adverse performance effects on high-capacity classifiers

Introducing skewness in the duplication parameters. Until now, we assumed that all entities in ED have identical duplicate set sizes $|D_k|$ and all duplicates in D_k are equally likely to occur. From our labeled data, we find that most entities have small duplicate set sizes and only a few entities have many duplicates. Also, some duplicates in the same set D_k are more likely to occur than others. Thus, we relax these two assumptions and include distributions in $|D_k|$ and $occ(D_k)$ that can better represent the duplication process. We alter our duplication process and approximate $|D_k|$ with a long-tail Zipfian distribution and $occ(D_k)$ with a Needle-and-Thread

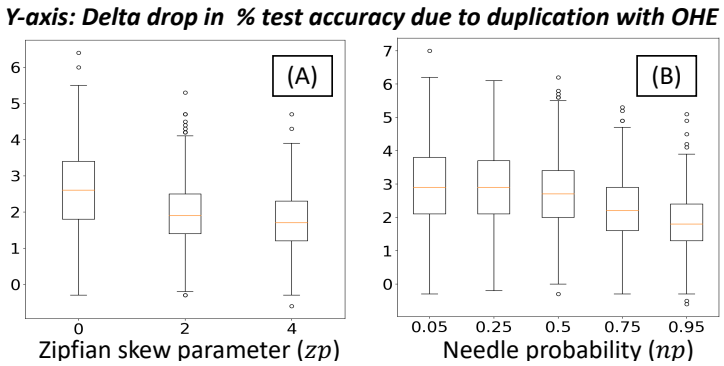


Figure 6.8: Effects of skew parameters on `AllA` simulation scenario for Random Forest with *OHE*. $|A|$ is preset to 3. (A) Vary Zipfian skew parameter zp of $|G_k|$, while fixing $(|r|_t, |ED|/|E|) = (3000, 30)$. (B) Vary needle probability parameter np of $occ(G_k)$, while fixing $(|r|_t, |ED|/|E|, zp) = (3000, 30, 2)$.

distribution, varying the skew amount one at a time. Figure 6.8 presents the results. We find that the delta drop in % accuracy due to duplicates remains significant regardless of the amount of skew in $|D_k|$ and $occ(D_k)$. With the Zipfian skew in $|D_k|$, the delta drop is highest at uniform distribution in $|D_k|$ (no skew setting) and marginally decreases as the skewness parameter is increased. On the other hand, when a needle-and-thread skew in $occ(D_k)$ is present, one duplicate from set D_k has a probability mass np (needle parameter). The remaining 1-np probability mass is uniformly distributed over the rest of the duplicates in D_k. We find that the delta drop due to duplicates decreases while still remaining significant when one duplicate value is more likely to occur than the rest (as np is increased). *Overall, the overarching conclusion from this analysis is that none of our results or takeaways change or get invalidated with this setup.*

Varying properties of duplicates being mapped to "*Others*." We study how duplicates that do not arise in the train set but are present in the test set (say, during deployment) can impact the downstream ML. We modify and repeat our duplication process on just the test set while keeping the train set intact. We introduce just one duplicate in the test set that gets mapped to "*Others*." Figure 6.4 (E-F) presents the results on `HiCapRF` with *OHE* where $|ED|/|E|$ and $occ(D_k)$ are varied. *We find that the delta drop in accuracies with all parameters are even more higher than the corresponding delta drops when both train and test set were duplicated (Figure 6.4*

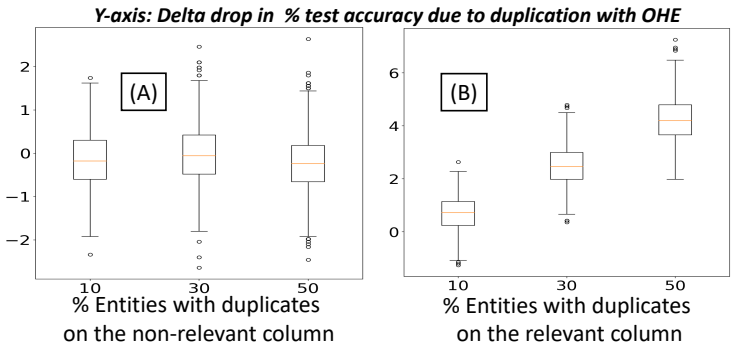


Figure 6.9: `AllA` results on `HiCapRF`. We set $|\mathcal{A}| = 4$ and vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|)$=$(5000, 25, 1)$. Duplicates introduced on the column with (A) non-positive *Relevancy* (noisy column) (B) high *Relevancy* (predictive column).

(B-C)). This simply suggests that the presence of unwarranted duplicates during the test can cause downstream ML to suffer significantly.

Varying column *Relevancy*. So far, we used all the columns in the dataset as part of CPT. Thus, all columns have high *Relevancy*. We now study low vs. high *Relevancy* setting with a slight twist in our simulation. We introduce an additional noisy column in the clean dataset: All except one column participates in CPT. Thus, we have the presence of both high and low *Relevancy* columns in the dataset. We introduce duplicates in both types of columns one at a time. Figure 6.9 presents the results. *We find that duplication on a highly relevant column has a significant adverse impact on HiCapRF performance. In contrast, the impact is negligible when duplicates are introduced over the noisy column. Even increasing the amount of duplication creates no impact with the low relevancy column. We even observe the same trend with HiCapMLP.*

6.5.2 Scenario Hyperplane

Data generating process. We set up distribution with a true hyperplane to separates the classes. (1) We define and fix the normal vector of the hyperplane with weights, $W_i, 1¡=i¡=|\mathcal{A}|$. Each weight W_i with cardinality $|E|$ is randomly sampled from a list of non-zero integers ($[-5,5]n\{0\}$) without replacement. Note that the integer weights are chosen only to make the distance calculation simpler in step (3). The trends of our results do not change even if

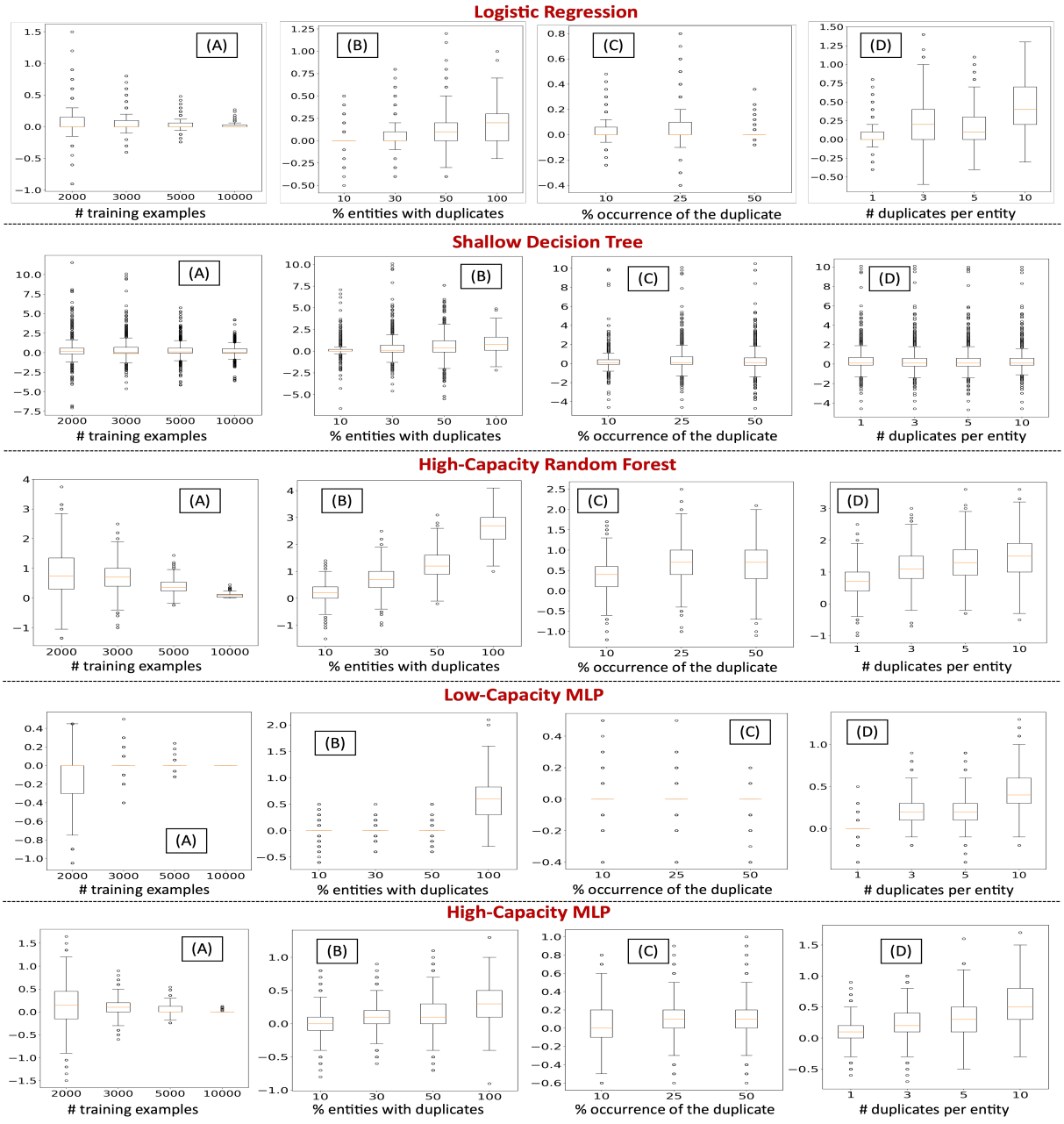


Figure 6.10: `Hyperplane` scenario results for `LR`, `ShallowDT`, `HiCapRF`, `LoCapMLP`, and `HiCapMLP` with *OHE*. $|X| = 3$. (A) Vary $|r|_t$ (# training examples) , while fixing $(|ED|/|E|, occ(D_k), |D_k|) = (30, 25, 1)$. (B) Vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|) = (3000, 25, 1)$. (C) Vary $occ(D_k)$, while fixing $(|r|_t, |ED|/|E|, |D_k|) = (3000, 30, 1)$. (D) Vary $|D_k|$, while fixing $(|ED|/|E|, |r|_t, occ(D_k)) = (30, 3000, 25)$, for all $k \in [1, |ED|]$.

the weights are chosen from real number uniform distribution. (2) Construct $|r|$ tuples of $\mathcal{A}$ by sampling values uniformly randomly from $|E|$. Thus, with fixed weights, the hyperplane over

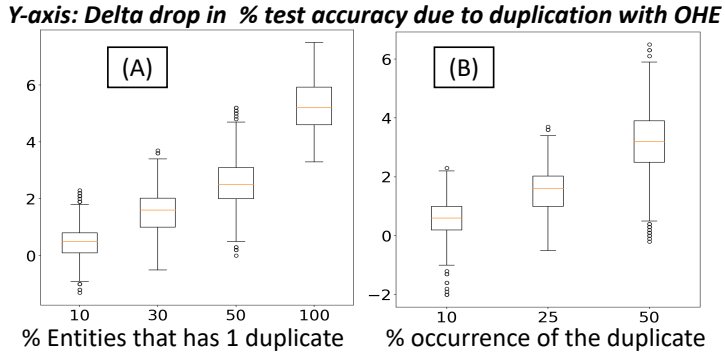


Figure 6.11: `Hyperplane` scenario results on *HiCapRF* with *OHE*. Only test set is diluted with duplicates. (A) Vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|) = (3000, 25, 1)$. (B) Vary $occ(D_k)$, while fixing $(|r|_t, |ED|/|E|, |D_k|) = (3000, 30, 1)$.

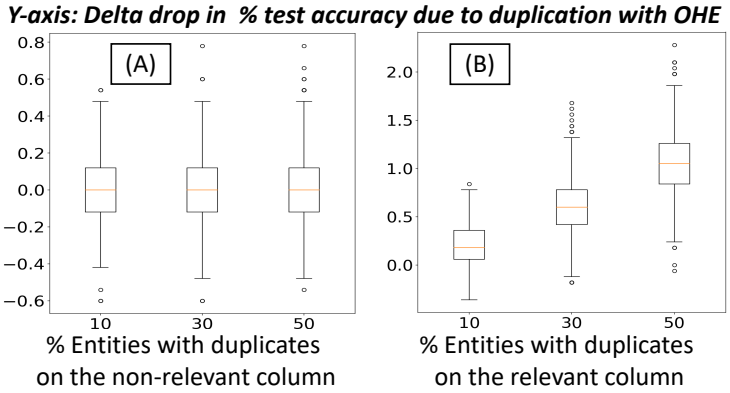


Figure 6.12: `Hyperplane` results on `HiCapRF`. We set $|\mathcal{A}| = 4$ and vary $|ED|/|E|$, while fixing $(|r|_t, occ(D_k), |D_k|) = (5000, 25, 1)$. Duplicates introduced on the column with (A) non-positive *Relevancy* (noisy column) (B) high *Relevancy* (predictive column).

One-hot encoded example feature vectors is given by $\sum_{i=1}^{i=|\mathcal{A}|} W_i \cdot A_i = 0$. (3) Examples for which $\sum_{i=1}^{i=|\mathcal{A}|} W_i \cdot A_i >= 0$ are labeled positive (Y=0) and remaining examples are labeled negative (Y=1). This generates the true dataset where all columns have high *Relevancy*. We introduce duplicates in them by following the same duplication process as Section 6.5.1.

Results. Figure 6.10 shows the delta drop in accuracy due to duplicates with all models using *OHE*. We find that all high-bias approaches are again robust to the presence of duplicates even when all entities are diluted with duplicates. Interestingly, `HiCapMLP` exhibits only marginal impact with duplicates. In contrast, duplicates affect `HiCapRF` significantly, especially in a high-duplication regime. We explain this interesting behavior in Section 6.5.3. We vary other

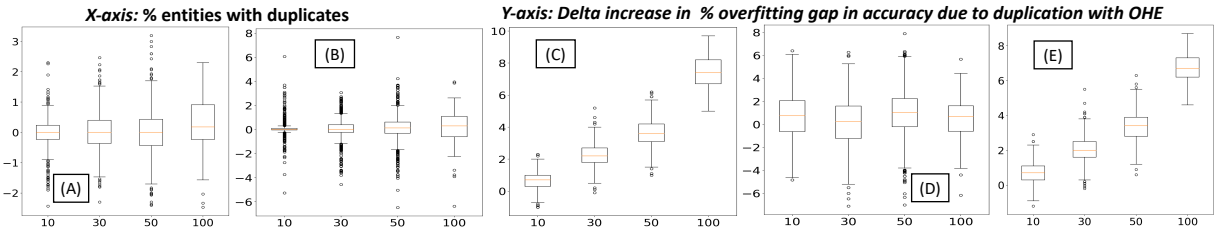


Figure 6.13: `AllA` setting results on (A) `LR` (B) `ShallowDT` (C) `HiCapRF` (D) `LoCapMLP` (E) `HiCapMLP` (with the same setup as Figure 6.4(B)).

confounders such as the other duplication parameters, the fraction of entities being mapped to "*Others,*" and column *Relevancy*. We confirm the same trends that we saw with all models in `AllA` scenario, except with `HiCapMLP` which behaves similar to `LR` than `HiCapRF`. We present the results in-depth below.

Varying the data regime and the parameters that control the amount of duplication. Figure 6.10 presents the delta drop in classification accuracy due to duplicates with all models using *OHE*. We again note that as the number of available training examples is increased, the delta drop in accuracy due to duplicates decreases for `HiCapRF`. Raising the other duplication parameters such as $|ED|/|E|, occ(D_k), |D_k|$ also increases the adverse performance impact of duplicates on `HiCapRF`. Interestingly, we find that `HiCapMLP` exhibits only a marginal amount of overfitting on the `Hyperplane` simulation scenario. Thus, we do not see any impact due to duplicates on `HiCapMLP` and also on all the high-bias approaches.

Varying properties of duplicates being mapped to *"Others"* and column *Relevancy.* We now repeat the simulation scenario presented in Section 6.5 but with `Hyperplane` setting, i.e. the true distribution is given by a hyperplane that separates the classes. Figure 6.11 presents the results when only the test set is diluted with duplicates. We again note that the presence of duplicates in the test set impacts *HiCapRF* significantly. Figure 6.12 presents the `Hyperplane` simulation results when we have the presence of both high and low relevancy columns in the dataset. We again find that the duplication on a noisy column has a marginal impact on `HiCapRF`, while duplicates the on relevant column affect it significantly.

6.5.3 Explanations

We now intuitively explain the general behavior of the ML classifiers in presence of duplicates on the two simulation settings with *OHE*. We check the generalization ability of the ML models with the overfitting gap. Figure 6.13 presents the overfitting gap results of all classifiers with *OHE* on the `AllA` scenario. We find that the delta drop in accuracy (Figure 6.4) closely follows the increase in the overfitting gap due to duplicates with both high-capacity models, `HiCapRF` and `HiCapMLP`. That is, the increase in overfitting or variance with duplicates explains the accuracy drop we see. *Thus, duplicates can negatively impact the generalization capability of the high-capacity models, which are prone to overfitting.* However, as the number of training examples rises, the amount of overfitting subsides. This explains our trends in the high-data regime.

Now with the `Hyperplane` setting, `LR` exhibits no amount of extra overfitting with duplicates. This is because the VC dimension of the `LR` is linear in the number of features. As the dimensionality of the feature space expands with duplicates, the VC dimension of `LR` expands. We get an expanded logistic hypothesis space with duplication that is a superset of the true logistic hypothesis space. Thus, a larger hypothesis space can potentially lead to more variance unless the true concept is simple enough to recover in an expanded feature space. We check the weights of the hyperplane learned with `LR` in presence of duplicates where a higher weight indicates higher importance. We find that the absolute weights of duplicate features are often close to zero. *This suggests that the `LR` can learn the true concept by completely ignoring the extra dimensions. Thus, the variance does not rise.* Also with `MLP`, each neuron functioning as a hyperplane classifier can easily recover the true hyperplane. Thus, even `HiCapMLP` doesn't overfit as much on the `Hyperplane` setting. In contrast, `HiCapRF` with *OHE* makes many binary splits on the clean data to recover the hyperplane, causing the tree to fully grow to the restricted height. Chances of further overfitting with duplicates are reduced with a limited height. Thus, we less significant drop in accuracy with duplicates on `HiCapRF` in the `Hyperplane` setting compared to `AllA`. *This*

also explains why a set-based split with StrE is more robust than binary splits with OHE as it allows to pack more category splits within the same tree height.

6.6 Towards Making Deduplication Automation Accurate

Category duplicates can often impact downstream ML accuracy substantially. However, what is missing in the existing open source AutoML tools is an automated deduplication workflow. The present art of deduplicating categories is still largely manual and relies on ad hoc rule-based approaches. This may not suffice in an AutoML environment, where given arbitrary data, it is difficult to decide the number of rules to use, which rules to apply, and with what threshold to apply. Thus, we take a step towards accurately automating category deduplication tasks using our labeled dataset (Section 6.3). Moreover, this enables us to assess the accuracy of automation.

6.6.1 Task

We simplify the *Categorical* deduplication problem with the following task: Given category set C, identify all pairs of categories that refer to the same real-world entity or are duplicates. In this work, we cast this task as a binary-class classification problem. We enumerate all possible pairs of categories from C. We then use rule-based and learning-based approaches (built using our labeled data) to assess their accuracies on the task. Admittedly, the time complexity of the enumeration is quadratic in the size of the category set, which can be large. However, in this work, we focus only on the feasibility study. *We showcase that our labeled data is valuable to deliver learning-based solutions that are more accurate than rule-based solutions.* We leave the scalability study with blocking mechanisms to reduce the quadratic complexity of enumeration to future work.

6.6.2 Rule-based Baselines

We build 3 rule-based baselines to validate if a set of rules can sufficiently identify the duplicates. We call the category pair a duplicate if the lowercase of the two categories is identical (*B1 baseline*), their Levenshtein edit distance is 1 (*B2 baseline*). *B3 baseline* uses a set of 6 rules (duplicate if any applies) to capture misspellings, synonyms, capitalization, ordering, and morphological variations. We provide two rules below. (1) To detect synonyms, we leverage a *doc2vec* model, pre-trained on English Wikipedia articles to obtain a feature representation for each of the two categories [25]. We then obtain a list of the top 100 words that are closest in terms of cosine distance for each of the two categories. The pair is marked duplicate if any of the categories belong to the top 100 closest words of the other's *doc2vec* representation. (2) We check misspellings with Jaccard distance threshold of 0.9.

6.6.3 ML-based approach using our data

We leverage the two classical learning-based approaches: Logistic Regression and Random Forest. We use two kinds of features for the prediction problem. (1) 8 category-level features to get aggregate stats about both categories such as absolute occurrence count, occurrence count relative to examples, the total number of characters, words, uppercases, lowercases, and special characters present. (2) 14 similarity features to estimate the "distance" between the categories such as Cosine, Hamming, Levenshtein, and Jaccard distance, and % common uni/bi/tri-grams.

6.6.4 Empirical Methodology

We consider 33 raw CSV files from our labeled data where there exists at least one *Categorical* column with duplicates. Overall, we get 52 *Categorical* columns with duplicates (*ColswithDups*) and 533 columns without any duplicates (*ColswithoutDups*). We form one train, validation, and test set by splitting each of *ColswithDups* and *ColswithoutDups* in 60:20:20 ratio.

Table 6.14: Lift in precision, recall, F1-score, and binary-class classification accuracy on our benchmark labeled test dataset (averaged across 5 random splits) with rule-based and ML-based approaches, relative to the random baseline. LogReg and RForest are Logistic Regression and Random Forest respectively.

	Dataset	train, validation, and test splits are kept intact						train, validation, and test splits are rebalanced					
		Majority Baseline	Rule-based			Our ML Models		Majority Baseline	Rule-based			Our ML Models	
			B1	B2	B3	LogReg	RForest		B1	B2	B3	LogReg	RForest
Duplicates	Precision	0	+100	+9.2	+0.6	+36	+49	0	+100	+64.2	+71.6	+96.9	+95.4
	Recall	0	+8.5	+12.5	+31.4	+47.8	+48.9	0	+8.5	+12.5	+31.4	+76.9	+69.7
	F1 Score	0	+15	+4.1	+1.1	+28.5	+42.9	0	+15	+19.8	+43.1	+84.8	+80
Non-Duplicates	Precision	99.9	+0	+0	+0	+0	+0	80	+1.4	+1.8	+5.2	+14.7	+13
	Recall	100	+0	+0	-2.2	+0	+0	100	+0	-2	-2.2	-0.7	-0.8
	F1 Score	99.9	+0	+0	-1.1	+0	+0	88.9	+0.9	+0.3	+2.1	+8	+7.1
Overall	Accuracy	99.9	+0	+0	-2.1	+0	+0	80	+1.7	+0.9	+4.5	+14.9	+13.3

We then enumerate all possible pairs of categories occurring in the column, with a max of 0.5M pairs per column for the tractability sake. Since the splits are done at a column-level, the test partition has category pairs of the columns not seen before. To control for randomness in results from the choice of splits, we perform 5 random (train/validation/ test) splits of our dataset and report the averaged results.

With the above methodology, we kept all the splits of the data intact, which makes them heavily imbalanced towards non-duplicate pairs. However, in practice, one may prune out unlikely matches (say obvious non-duplicate pairs with a blocking heuristic), instead of enumerating all possible category pairs. Thus, to showcase a real-world representative deployment scenario, we rebalance all the splits of the data. We downsample non-duplicate pairs randomly such that the ratio 20:80 duplicate:non-duplicate pairs is maintained for every column. Again, we report the averaged accuracy results from 5 random splits of our data.

6.6.5 Results

Table 6.14 shows the average lift in precision, recall, F1-score, and accuracy on our test dataset with rule-based and the ML-based approaches, relative to the majority baseline (predict

all category pairs as non-duplicate). We find that both the ML approaches are substantially more accurate than the three rule-based baselines, regardless of how data is balanced, e.g., a lift of 10.4% in binary class accuracy with Logistic model compared to $B3$ baseline when data is rebalanced. Overall, these results suggest that our labeled dataset is valuable to provide not only an objective way of measuring the task accuracy but also automating the task more accurately than the baselines. Although duplicates are identified with high precision and recall with our ML model, there remains an accuracy gap for improvement. We discuss how our approach can be further improved in future work in Section 6.7.2.

6.7 Discussion

6.7.1 Public Release

We release a public repository on GitHub with our entire benchmark suite [36]. This includes our labeled dataset of entities in the string *Categorical* columns annotated with their category duplicates, along with their raw CSV files. We also release the downstream benchmark suite with raw and deduped versions of all datasets, synthetic benchmarks, and code to run them.

6.7.2 Takeaways

We find that the presence of duplicates can potentially impact downstream ML accuracy significantly. The amount of impact can be characterized by multiple confounders that interact in non-trivial ways. It is not always possible to disentangle the impact on downstream ML with each confounder individually. However, our analysis with downstream benchmark suite and simulation studies can provide insights into when cleaning effort would be more or less beneficial. The current practice among ML practitioners and AutoML platform developers to handle *Categorical* duplicates is largely ad hoc rule-based and completely oblivious to many confounders. We first

give general guidelines and actionable insights to help them prioritise their deduplication effort and also potentially design better end-to-end automation pipelines respectively. We then lay out the critical open questions in this direction for researchers.

1. For ML practitioners and AutoML platform developers.

a. Make ML workflows less susceptible to the adverse performance impact of category duplicates. `LR` is less prone to overfitting than `RF` and `MLP` when duplicates arise. Also, *StrE* is relatively more robust than *OHE* when using `RF`. Moreover, *SimE* inherently exploits the presence of similarly valued duplicates in the *Categoricals*. This makes it significantly more robust from duplicates compared to *OHE* and *StrE*. Moreover, unseen duplicates that arise during the deployment phase can degrade ML performance with *OHE* or *StrE*. Overall, *Similarity* encoding and/or a Logistic Regression can be utilized by ML practitioners and AutoML developers if they desire to guard their pipelines against any adverse drop in ML performance from likely duplicates. Moreover, the impact of duplicates get mitigated in a higher-data regime compared to a low-data regime. Thus, whenever possible, one can consider getting more train data to offset their impact by trading off runtime.

b. Track the overfitting gap of ML models. Deduplication can reduce the overfitting gap caused by duplicates on downstream ML. Thus, cleaning duplicates may not be worthwhile if the overfitting gap is already low on the raw dataset. Monitoring and presenting it as an auxiliary metric to the AutoML user can provide them with more confidence about the downstream performance.

c. Simulate duplications in your data with the synthetic suite. Our synthetic simulation suite provides an empirical methodological infrastructure for understanding the category deduplication effect in presence of different confounders. Given an arbitrary dataset, it can create semi-synthetic variants of category duplicates by modeling them with various observed properties from our labeled data. Also, the impact becomes more interpretable when confounders are disentangled, e.g., quantifying the impact of "*Others*" in a deployment setting with *OHE*.

6.8 Conclusion

Our empirical analysis using downstream benchmark suite and simulation studies shows that duplicates impact high-capacity classifiers much more than Logistic Regression. Also, we find that Similarity Encoding is significantly more robust than One-hot and String Encoding. Thus, Category Deduplication, a data prep commonly performed may not always be needed. Our takeaways from this work provide actionable insights and guidelines to help ML practitioners prioritise their deduplication effort. Also, AutoML developers can use these analyses to design robust end-to-end automation pipelines. Moreover, our work opens up major open questions for research that require contributions from the community. Also, our feasibility study shows that our labeled data is useful to deliver ML-based solutions that are more accurate than rule-based baselines.

Chapter 6 contains material from "An Empirical Study on (Non-)Importance of Cleaning Categorical Duplicates before ML" by Vraj Shah, Thomas Parashos, and Arun Kumar, which has been submitted for publication of the material. The dissertation author was the primary investigator and author of this material.

Chapter 7

Related Work

7.1 Related Work for Hamlet++

Database Dependencies and ML. Optimizing ML over joins of multiple tables was studied in [99, 130, 125, 98], but their goal was primarily to reduce runtimes without affecting ML accuracy. ML over joins was also studied in [168] but their focus was on devising a new ML algorithm. In contrast, our work studied the more fundamental question of whether KFK joins can be avoided safely for ML classifiers. We first demonstrated the feasibility of avoiding joins safely in [100] for linear models. In this work, we revisit that idea for high-capacity classifiers and also empirically verify mechanisms to make foreign key features more practical. Embedded multi-valued dependencies (EMVDs) are database dependencies that are more general than functional dependencies [41]. The implication of EMVDs for probabilistic conditional independence in Bayesian networks was originally described by [114] and further explored by [162]. However, their use of EMVDs still requires computations over all features in the data instance. In contrast, avoiding joins safely omits entire sets of features for complex ML models *without performing any computations* on the foreign features. There is a large body of work on statistical relational learning (SRL) to handle joins that cause duplicates in the fact table [66]. But as mentioned

before, our work focuses on the regular IID setting for which SRL might be an overkill.

Feature Selection. The ML and data mining communities have long studied feature selection methods [72]. Our goal is *not* to design new feature selection methods nor is it compare existing ones. Rather, we study if KFKDs/FDs in the schema let us to avoid entire tables a priori for some popular high-capacity classifiers, i.e., "short-circuiting" feature selection using database schema information to reduce the burden of data sourcing. The trade-off between feature redundancy and relevancy is well-studied [72, 169, 92]. The conventional wisdom is that even a feature that is redundant might be highly relevant and thus, unavoidable in the mix [72]. Our work shows that, perhaps surprisingly, even highly relevant foreign features can be safely discarded in many practical classification tasks for many high-capacity classifiers. There is prior work on exploiting FDs in feature selection; [146] infers approximate FDs using the dataset instance and exploits them during feature selection, FOCUS [42] is an approach to bias the input and reduce the number of features, while [54] proposes a measure called consistency to aid in feature subset search. Our work is orthogonal to these algorithms because they all still require computations over all features, while avoiding joins safely *omits foreign features without even looking at them* and obviously, without performing any computations on them. To the best of our knowledge, no feature selection method exhibits such a dramatic capability. Gini and information gain are known to be biased towards large-domain features in decision trees [56]. Different approaches have been studied to resolve this issue [78]. Our work is orthogonal because we study how KFKDs/FDs enable us to ignore foreign features a priori safely. Even with the gain ratio score that is known to mitigate the bias towards large-domain features, our main findings stand. Unsupervised dimensionality reduction methods such as random hashing and PCA are also popular [75]. Our foreign key domain compression techniques for decision trees are inspired by such methods.

Data Integration. Integrating data and features from various sources for ML often requires applying and adapting data integration techniques [104, 58], e.g., integrating features

from different data types in recommendation systems [84], sensor fusion [89], dimensionality reduction during feature fusion [65], and controlling data quality during data fusion [60]. Avoiding joins safely can be seen as one schema-based mechanism to reduce the integration burden by predicting a priori if a source table is unlikely to improve accuracy. It is an open challenge to devise similar mechanisms for other types of data sources, say, using other schema constraints, ontology information, and sampling. There is also a growing interest in making data discovery and other forms of metadata management easier [63, 77]. Our work can be seen as a mechanism to verify the potential utility of some of the discovered data sources using their metadata. We hope our work spurs more research in this direction of exploiting ideas from data integration and data discovery to reduce the data sourcing burden for ML tasks.

7.2 Related Work for ML Data Prep Zoo

Data Prep and Cleaning. TFDV [45] is a tool for managing ML-related data in TensorFlow Extended. It uses conservative rule-based heuristics to infer ML schema from column statistics. Our ML-based approach raises accuracy of ML schema inference substantially. DataLinter is a rule-based tool to inspect a data file and flag possible data quality issues to the user [81]. It still requires users to perform data transformations manually, which makes it orthogonal to our focus. There is growing work on reducing data cleaning effort using ML properties (e.g., [96]). Our work is part of this growing direction but our work specifically targets data prep tasks and casts them as applied ML tasks.

AutoML Platforms. Existing AutoML platforms such as Einstein [13] and AutoWeka [95] focus mainly on model selection, not ML-based ML data prep. Thus, the models we produce can enhance such platforms. OpenML [155] is an open-source platform for ML users to share and compare models, data, and analysis workflows. Our focus on *creating* high-quality labeled datasets for semi-automating ML data prep tasks is thus complementary. Our artifacts can be

contributed to OpenML for spurring more research on end-to-end AutoML platforms. We could also get more analysis workflows from OpenML to enhance our work in the future.

7.3 Related Work for ML Feature Type Inference

AutoML Platforms. Several AutoML tools such as AutoWeka [145] and Auto-sklearn [64] have an automated search process for model selection, allowing users to spend no effort for algorithm selection or hyper-parameter search. However, these AutoML systems do not automate the ML feature type inference task. Several tools perform automatic data transformation steps and generate a set of useful features given a dataset [87, 86]. However, Deep Feature Synthesis algorithm [86] assumes that the ML feature types are provided explicitly as input, while ExploreKit [87] operates on the syntactic types. Thus, such automatic feature engineering tools can benefit by leveraging the ML models trained on our labeled data.

Other end-to-end AutoML platforms such as Einstein AutoML [13], AutoML Tables [9], and AutoGluon [62] do automate the type inference task. We believe that the standardization of the task and our benchmark labeled dataset is valuable to objectively compare and improve their AutoML platforms. The ML models trained on our labeled dataset can be integrated into such AutoML platforms to improve their type inference accuracy. In addition, other ML platforms such as Airbnb's Zipline [3], Uber's Michelangelo [20], Facebook's FBLearner Flow [6], and commercial platforms such as H20.AI [11] and DataRobot [5] are complementary to our focus and they can also benefit by adopting models trained on our data.

ML Data Prep and Cleaning. Auto-Type [167] is a semantic type detection tool that synthesizes type detection logic for semantic types such as *EAN Code*, *Swift Code*, etc. But it too is complementary and not directly usable for AutoML just like Sherlock. DataLinter is a rule-based tool that inspects a data file and raises potential data quality issues as warnings to the user [81]. However, ML feature type inference must be done manually. Many works

study program synthesis-based approaches [70, 71, 83, 76] and/or visual interfaces [18] to reduce manual data transformation grunt work in data prep. There is also much work on reducing data validation and cleaning effort (e.g., [96, 97, 129]). Our work further this general direction on reducing manual effort but it is complementary to all these prior works: our paper is the first to formalize and benchmark ML feature type inference in AutoML platforms.

Database Schema Inference. DB schema inference has been explored in some prior work. Google's BigQuery does syntactic schema detection when loading data from external data warehouses [14]. [44] infers a schema from JSON datasets by performing *map* and *reduce* operations using pre-defined rules. But DB schema inference task is syntactic. For instance, the attribute type with integer values has to be identified as an integer. In contrast, with ML type inference the attributes with type integer can be *Categorical.*

Benchmarks. OpenML AutoML Benchmark focuses on understanding the automation of model selection and hyper-parameter search components of the ML workflow [67]. However, they do not cover any data prep steps. CleanML benchmark focuses on studying the effect of data cleaning operations on downstream models [103]. However, they do not handle the feature type inference task. Thus, both benchmarks are orthogonal to our work.

Data/Model Repositories. OpenML [155] is an open-source collaborative repository for ML practitioners and researchers to share their models, datasets, and workflows for reuse and discussion. Our labeled datasets can be made available to the OpenML community to invite more contributions for augmenting the current labeled dataset and for building more sophisticated models. Hence, our work is complementary to OpenML.

7.4 Related Work for Category Deduplication

Empirically Studying the Impact on ML. CleanML [103] analyses the impact of many data cleaning steps on downstream ML classification tasks. However, they do not cover *Cate-*

gorical deduplication step. Open source AutoML benchmark [67] performs a comparative study of many AutoML tools in terms of the overall classification accuracy with their model selection routines. However, understanding any data prep step from downstream ML accuracy standpoint is not the focus of their work. A previous work [135] performed an objective benchmarking of a specific data prep step for ML, namely the feature type inference task. We build upon their open-sourced datasets but we study a completely different problem.

AutoML Platforms. Several AutoML libraries allow users to perform automated ML algorithm and hyper-parameter search without covering any data prep tasks [64, 145, 112]. Other AutoML platforms that offer (or claim to offer) end-to-end ML support such as Salesforce Einstein TransmogrifAI [17], Google AutoML Tables [10], and Amazon AutoGluon [62] do automate many data prep tasks. However, none of them handles *Categorical* duplicates. Instead, the users are asked to explicitly clean and remove inconsistencies in *Categorical* columns before using their platforms [8]. Our labeled data can lead to more contributions from the community to automate the deduplication task. Moreover, we believe that our downstream benchmark, our analysis with simulation studies, and takeaways are all valuable to improve their AutoML platforms.

Data Prep and Cleaning for ML. There exists numerous data prep tools such as rule-based tools [81], exploratory data analysis-based libraries [116], visual interfaces [18], and program synthesis-based tools [76, 83] to reduce user's manual grunt work effort and allow them to productively prepare their data for ML. In addition, Auto-insight generator tools [160, 57, 55] allow users to discover interesting statistical properties of a given dataset. Our work's insights can complement all these tools to reduce human time and effort and make their analysis more interpretable. Some works have studied human-in-the-loop data cleaning to improve ML accuracy and reduce user effort [96, 97]. However, they do not support a cleaning operation when *Categorical* duplicates arise. Our labeled data can spur more follow-up works in this general direction of automating and improving data prep for ML.

Entity Matching (EM). EM, the task of identifying whether records from two tables refer to the same real-world entity has received much attention with rule-based [141, 113, 142], learning-based [93, 110, 105, 174], semi-supervised [88], unsupervised [165], and active-learning [108] approaches. Our focus is on analyzing the impact of category duplicates and category deduplication on downstream ML. In the service of this goal, we offer new hand-labeled data, benchmarks, and simulations. We do not propose new techniques for EM or even category deduplication. Thus, prior work on EM is complementary to ours in terms of utility for AutoML platforms.

EM solutions [93, 174, 105] operate at a tuple-level since they have access to the entire feature vectors of the two tables. Note that tuple-level duplicates do not necessarily imply duplication in *Categorical* strings. Likewise, duplication in a *Categorical* column does not necessarily lead to row-level duplicates. Thus, problem of EM is orthogonal to category deduplication. Admittedly, it is possible to view category deduplication as an extension of row-level deduplication but doing so is non-trivial. *Regardless, our focus is to study only the impact of category deduplication on ML and not how to perform deduplication.* We leave it to future work to automate this, including potentially extending existing row-level deduplication works. Moreover, a *Categorical* column assumes values from a finite closed domain. Thus, generic open domain addresses, person names, or even textual descriptions that are used in public EM datasets [93, 94] are not *Categorical* and not enough to study this task.

Chapter 8

Conclusion and Future Work

In this dissertation, we take a step towards simplifying three critical ML data prep tasks over tabular data: joining tables with key-foreign key dependencies between them, ML feature type inference, and category deduplication. We use DB schema management along with ML techniques to simplify, objectively quantify, and automate them. This not only improve the productivity of the data scientists, but also makes it easier to validate (Auto)ML tools. Our objective benchmark on both data prep tasks and downstream tasks exposes several shortcomings of the existing solutions and the common ML practice. This can help ML practitioners glean actionable insights to help them prioritise their data prep effort. Also, this can potentially lead to better design of end-to-end automation pipelines. In this Chapter, we provide an overview of several exciting research directions at the intersection of ML theory, data management, ML system design, and human computer interaction for future work.

8.1 Future Work Related to Hamlet++

While our analysis intuitively explains the behavior of decision trees and RBF-SVMs, there are many open questions for research. Is it possible to quantify the probability of wrong partitioning with a decision tree as a function of the data properties? Is it possible to quantify the

probability of mismatched examples being picked by RBF-SVM? Why does the theory of VC dimension predict the opposite of the observed behavior with these models? How do we quantify their generalization if memorization is allowable and what forms of memorization are allowed? Answering these questions would provide deeper insights into the effects of KFKDs/FDs on such classifiers. It could also yield more formal mechanisms to characterize when avoiding joins is feasible beyond just looking at tuple ratios.

There are database dependencies more general than FDs: embedded multi-valued dependencies and join dependencies [140]. How do these dependencies among features affect ML models? There are also conditional FDs, which satisfy FD-like constraints among subsets of rows [140]; how do such data properties affect ML models? Finally, Armstrong's axioms imply that foreign features can be divided into arbitrary subsets before being avoided; this opens up a new trade-off space between avoiding $\mathbf{X}_R$ and using it $\mathbf{X}_R$. How do we quantify this trade-off and exploit it? Answering these questions would open up new connections between data management and ML theory and potentially enable new functionalities for ML analytics systems. Other interesting avenues include understanding the effects of other database dependencies on ML, including regression and clustering models, and designing an automated "advisor" for data sourcing for ML tasks, especially when there are heterogeneous data types and sources.

8.2 Future Work Related to ML Data Prep Zoo

In this dissertation, we focused on benchmarking and automating two critical ML data prep tasks: ML Feature Type Inference and Category Deduplication. However, there still remains other tasks open for benchmarking such as Embedded Number Extraction and Value Standardization, which we discussed in Chapter 4. Our philosophy of using ML to automate data prep can directly be applied here as well. We see two major avenues of future work.

1. Creating large labeled dataset for data prep tasks is the major research challenge.

Although, manual annotations deliver us high-quality datasets, it is not scalable for other tasks. Also, weak supervision currently do not support complex prediction outputs. Crowdsourcing is the most feasible option for many data prep tasks such as Embedded Number Extraction unlike ML Feature Type Inference task, which we found to be too technically nuanced for lay crowd workers. Designing appropriate mechanisms and interfaces to support such data prep through crowdsoucing is an open question. How a task is designed by a requester is very critical for quality as it has been shown in number of literatures [164, 82]. Setting up task design requires addressing several open questions such as how are instructions given to workers, how the human computation process is set up, financial incentives provided to them, how tasks are assigned and how worker's responses are aggregated. Data augmentation in slices where examples are under-represented can potentially be tried to synthesize labeled data. Also, there remains many open questions with regard to employing weak supervision: how much is weak supervision helpful? what sort of weak supervision rules can be created? How to create new interfaces for weak supervision?

2. An another important research question to answer is *how much of ML data prep can we get away with?* Answering this question would require characterizing the downstream performance in terms of appropriate confounders. For instance, we found in Chapter 6 that for many confounder settings such as Logistic regression and Similarity Encoding, Category deduplication task can be completely avoided. Likewise, for Embedded Number Extraction task, n-grams provide an alternative representation for numbers with the vector representation of n-gram always larger than the integer representation. So, is the step even needed for a high-capacity classifier? Answering this question requires characterizing this formally from a learning theory standpoint.

8.3 Future Work Related to ML Feature Type Inference

We see three main avenues of improvement for researchers wanting to improve accuracy: better features, better models, and/or getting more labeled data.

First, designing features that can perfectly capture human-level reasoning is an open research question. We found that descriptive stats and attribute names are most useful for prediction, while raw attribute values have only marginal utility. Thus, one can consider designing better featurization routines for them. Second, capturing more semantic knowledge of attributes with an alternative neural architecture is another open problem. Finally, based on our analysis in Section 5.4.4, one potential way to increase the accuracy is to create more labeled data in categories of examples where ML models get confused, e.g., for *List* type. Weak supervision and denoising with Snorkel [122] and/or Snuba [159] is one potential mechanism to amplify labeled datasets and teach the ML models to learn better.

8.4 Future Work Related to Category Deduplication

We discuss three major open questions for research that require contributions from the community.

a. Design accurate methods for deduplication. While our ML models for deduplication outperforms rule-based baselines, there remains a large scope of improvement. There are mainly two ways to enhance our performance results. (1) Capturing semantic-level characteristics of the categories with either designing features or with deep learning models like Siamese neural network [111]. (2) Data amplification through augmentation in slices where examples are under-represented. Table 6.5 can provide guidelines on how to synthesize duplicates. Existing weak supervision environments can augment them further [123, 159].

b. Define new benchmark tasks. In an AutoML production setting with millions of features, one cannot possibly perform deduplication over all columns. Given a cleaning budget in

terms of accuracy and runtime, how to guide an AutoML platform to prioritize which column to clean? One can consider designing a coarse-grained classifier to help identify them, but how does a column-level featurization look like? Our labeled data can be leveraged and amplified at a column-level to assess the accuracy of such approaches.

c. Theoretical quantification. Our empirical study suggests that duplicates can increase variance since the hypothesis space of the model can grow. This opens up several research questions at the intersection of ML theory and data management: Is it possible to establish bounds on the increase in variance using VC-dimension theory [157]? Can we set up a decision rule to formally characterize when deduplication would be needed?

Bibliography

[1] Personal communications: Facebook friend recommendation system; LogicBlox retail analytics; MakeMyTrip customer analytics.

[2] Nature News. https://www.nature.com/articles/d41586-017-02190-5.

[3] Airbnb Zipline, https://conferences.oreilly.com/strata/strata-ny-2018/public/schedule/detail/68114, Accessed April 11, 2022.

[4] AWS Glue, https://aws.amazon.com/glue, Accessed April 11, 2022.

[5] DataRobot, https://www.datarobot.com, Accessed April 11, 2022.

[6] Facebook's FBLearner Flow, https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/, Accessed April 11, 2022.

[7] Google AutoML Tables cleaning duplicates user guidelines, {https://cloud.google.com/automl-tables/docs/data-best-practices\#make_sure_your_categorical_features_are_accurate_and_clean}, Accessed April 11, 2022.

[8] Google AutoML Tables data prep user guidelines, https://cloud.google.com/automl-tables/docs/data-best-practices, Accessed April 11, 2022.

[9] Google AutoML Tables, https://cloud.google.com/automl-tables, Accessed April 11, 2022.

[10] Google Cloud AutoML, https://cloud.google.com/automl/, Accessed April 11, 2022.

[11] H2o.AI, https://www.h2o.ai/, Accessed April 11, 2022.

[12] Kaggle: The State of Data Science and Machine Learning, https://ailab-ua.github.io/courses/resources/the_state_of_data_science_machine_learning_-_kaggle_2017_survey.pdf, Accessed April 11, 2022.

[13] Salesforce Einstein AutoML, https://www.salesforce.com/video/1776007, Accessed April 11, 2022.

[14] Schema Detection BigQuery, https://cloud.google.com/bigquery/docs/schema-detect, Accessed April 11, 2022.

[15] Similarity Encoder Library, https://github.com/dirty-cat/dirty_cat, Accessed April 11, 2022.

[16] TPC Benchmarks, https://www.tpc.org/information/benchmarks5.asp, Accessed April 11, 2022.

[17] TransmogrifAI: Automated Machine Learning for Structured Data, https://transmogrif.ai/, Accessed April 11, 2022.

[18] Trifacta: Data Wrangling Tools & Software, https://www.trifacta.com/, Accessed April 11, 2022.

[19] TSM Habitat Dataset. https://data.ca.gov/dataset/tsm-habitat-rapid-assessment-survey-2016-ds28271, Accessed April 11, 2022.

[20] Uber Michelangelo, https://eng.uber.com/michelangelo/, Accessed April 11, 2022.

[21] https://datacatalog.hsls.pitt.edu/dataset/77, Accessed April 11, 2022.

[22] https://data.cityofchicago.org/Transportation/Relocated-Vehicles/5k2z-suxx, Accessed April 11, 2022.

[23] https://everydaydata.co/2017/02/07/hacker-news-part-one.html, Accessed April 11, 2022.

[24] https://github.com/fivethirtyeight/data/tree/master/region-survey, Accessed April 11, 2022.

[25] https://github.com/jhlau/doc2vec, Accessed April 11, 2022.

[26] https://maxcandocia.com/article/2018/Oct/22/trick-or-treating-ages/, Accessed April 11, 2022.

[27] https://osmihelp.org/research, Accessed April 11, 2022.

[28] https://transparentcalifornia.com/salaries/san-francisco/, Accessed April 11, 2022.

[29] https://www.asdcode.de/2021/01/it-salary-survey-december-2020.html, Accessed April 11, 2022.

[30] https://www.kaggle.com/definitelyliliput/rawscores, Accessed April 11, 2022.

[31] https://www.kaggle.com/mlomuscio/wifi-study, Accessed April 11, 2022.

[32] https://www.kaggle.com/pushpaltayal/etailing-customer-survey-in-india, Accessed April 11, 2022.

[33] Utility Dataset. https://data.ny.gov/Energy-Environment/Utility-Company-Customer-Service-Response-Index-CS/w3b5-8aqf, Accessed April 11, 2022.

[34] ACM SIGMOD Blog: Automation of Data Prep, ML, and Data Science: New Cure or Snake Oil?, https://wp.sigmod.org/?p=3270, Accessed June 2, 2022.

[35] Microsoft Excel, https://www.microsoft.com/en-us/microsoft-365/excel, Accessed June 2, 2022.

[36] The ML Data Prep Zoo Repository, Accessed March 15, 2018. https://github.com/pvn25/ML-Data-Prep-Zoo.

[37] Github Repository for ML Feature Type Inference, https://github.com/pvn25/ML-Data-Prep-Zoo/tree/master/MLFeatureTypeInference, Accessed March 22, 2021.

[38] URL Standards, https://url.spec.whatwg.org, Accessed March 22, 2021.

[39] Building Violations Dataset. https://data.cityofchicago.org/Buildings/Vacant-and-Abandoned-Buildings-Violations/kc9i-wq85, Accessed September 5, 2021.

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

[41] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[42] Hussein Almuallim and Thomas G. Dietterich. Efficient Algorithms for Identifying Relevant Features. Technical report, 1992.

[43] Richard C. Angell, George E. Freund, and Peter Willett. Automatic spelling correction using a trigram similarity measure. *Inf. Process. Manag.*, 19(4):255–261, 1983.

[44] Mohamed-Amine Baazizi, Houssem Ben Lahmar, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Schema Inference for Massive JSON Datasets. In *Extending Database Technology (EDBT)*, 2017.

[45] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy,

Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1387–1395. ACM, 2017.

[46] Paul Suganthan G. C., Adel Ardalan, AnHai Doan, and Aditya Akella. Smurf: Self-service string matching using random forests. *Proc. VLDB Endow.*, 12(3):278–291, 2018.

[47] Patricio Cerda, Gaël Varoquaux, and Balázs Kégl. Similarity encoding for learning with dirty categorical variables. *Machine Learning*, 107(8):1477–1494, 2018.

[48] Vincent Chen, Sen Wu, Alexander J Ratner, Jen Weng, and Christopher Ré. Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices. In *Advances in neural information processing systems*, pages 9397–9407, 2019.

[49] Peter Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012.

[50] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. Notes from the ai frontier: Insights from hundreds of use cases. *McKinsey Global Institute*, page 28, 2018.

[51] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Slice Finder: Automated Data Slicing for Model Validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1550–1553. IEEE, 2019.

[52] Anamaria Crisan and Brittany Fiore-Gartland. Fits and starts: Enterprise use of automl and the role of humans in the loop. In Yoshifumi Kitamura, Aaron Quigley, Katherine Isbister, Takeo Igarashi, Pernille Bjørn, and Steven Mark Drucker, editors, *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, pages 601:1–601:15. ACM, 2021.

[53] CrowdFlower. 2017 data science survey. `https://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport.pdf`.

[54] Manoranjan Dash, Huan Liu, and Hiroshi Motoda. Consistency based feature selection. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, PAKDK, pages 98–109, London, UK, UK, 2000. Springer-Verlag.

[55] Çagatay Demiralp, Peter J. Haas, Srinivasan Parthasarathy, and Tejaswini Pedapati. Foresight: Recommending visual insights. *Proc. VLDB Endow.*, 10(12):1937–1940, 2017.

[56] Houtao Deng, George Runger, and Eugene Tuv. Bias of importance measures for multi-valued attributes and solutions. In *Proceedings of the 21st International Conference on*

Artificial Neural Networks - Volume Part II, ICANN'11, pages 293–300, Berlin, Heidelberg, 2011. Springer-Verlag.

[57] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 317–332. ACM, 2019.

[58] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[59] Pedro Domingos. A Unified Bias-Variance Decomposition and its Applications. In *Proceedings of 17th International Conference on Machine Learning*, 2000.

[60] Xin Luna Dong and Divesh Srivastava. Big data integration. *PVLDB*, 6(11):1188–1189, 2013.

[61] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[62] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *CoRR*, abs/2003.06505, 2020.

[63] Raul Castro Fernandez, Ziawasch Abedjan, Samuel Madden, and Michael Stonebraker. Towards Large-scale Data Discovery: Position Paper. In *Proceedings of the Third International Workshop on Exploratory Search in Databases and the Web*, ExploreDB '16, pages 3–5, New York, NY, USA, 2016. ACM.

[64] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2962–2970, 2015.

[65] Yun Fu, Liangliang Cao, Guodong Guo, and Thomas S. Huang. Multiple feature fusion by subspace learning. In *Proceedings of the 2008 International Conference on Content-based Image and Video Retrieval*, CIVR '08, pages 127–134, New York, NY, USA, 2008. ACM.

[66] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning)*. The MIT Press, 2007.

[67] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[68] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[69] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. 2020.

[70] Sumit Gulwani. Automating String Processing in Spreadsheets Using Input-Output Examples. In *ACM Sigplan Notices*, volume 46, pages 317–330. ACM, 2011.

[71] Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, 2012.

[72] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A. Zadeh. *Feature Extraction: Foundations and Applications*. New York: Springer-Verlag, 2001.

[73] John T. Hancock and Taghi M. Khoshgoftaar. Survey on categorical data for neural networks. *J. Big Data*, 7(1):28, 2020.

[74] Trevor Hastie, Jerome H. Friedman, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2001.

[75] Trevor Hastie, R Tibshirani, and J Friedman. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer-Verlag, 2001.

[76] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. Transform-Data-by-Example (TDE): An Extensible Search Engine for Data Transformations. *Proceedings of the VLDB Endowment*, 11(10):1165–1177, 2018.

[77] Joseph M Hellerstein, Vikram Sreekanti, Joseph E Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, and Shirshanka Das. Ground: A Data Context Service. In *CIDR*, 2017.

[78] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS*, 15(3):651–674, 2006.

[79] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1500–1508, 2019.

[80] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer, 2019.

[81] Nick Hynes, D Sculley, and Michael Terry. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. In *NIPS MLSys Workshop*, 2017.

[82] Ayush Jain, Akash Das Sarma, Aditya Parameswaran, and Jennifer Widom. Understanding workers, developing effective tasks, and enhancing marketplace dynamics: a study of a large crowdsourcing marketplace. *Proceedings of the VLDB Endowment*, 10(7):829–840, 2017.

[83] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and Hosagrahar V Jagadish. Foofah: A Programming-By-Example System for Synthesizing Data Transformation Programs. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1607–1610, 2017.

[84] Yushi Jing, David Liu, Dmitry Kislyuk, Andrew Zhai, Jiajing Xu, Jeff Donahue, and Sarah Tavel. Visual search at pinterest. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1889–1898, New York, NY, USA, 2015. ACM.

[85] Jingbo Shang, Rajesh Gupta, Lucila Ohno-Machado, Arun Kumar, Giorgio Quer. Towards Intelligent Sharing and Search for AI Models and Datasets. `https://shangjingbo1226.github.io/2020-11-01-nsf-c-accel`.

[86] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[87] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. ExploreKit: Automatic Feature Generation and Selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[88] Mayank Kejriwal and Daniel P. Miranker. Semi-supervised instance matching using boosted classifiers. In Fabien Gandon, Marta Sabou, Harald Sack, Claudia d'Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann, editors, *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, volume 9088 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2015.

[89] Bahador Khaleghi, Alaa Khamis, Fakhreddine O. Karray, and Saiedeh N. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, January 2013.

[90] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data Scientists in Software Teams: State of the Art and Challenges. *IEEE Transactions on Software Engineering*, PP(99), 2017.

[91] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations (ICLR)*, 2015.

[92] Daphne Koller and Mehran Sahami. Toward Optimal Feature Selection. In *ICML*, 1995.

[93] Pradap Venkatramanan Konda. *Magellan: Toward building entity matching management systems*. The University of Wisconsin-Madison, 2018.

[94] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3(1):484–493, 2010.

[95] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *J. Mach. Learn. Res.*, 18:25:1–25:5, 2017.

[96] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Jiannan Wang, and Eugene Wu. ActiveClean: An Interactive Data Cleaning Framework For Modern Machine Learning. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 2117–2120. ACM, 2016.

[97] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. BoostClean: Automated Error Detection and Repair for Machine Learning. *CoRR*, abs/1711.01299, 2017.

[98] Arun Kumar, Mona Jalal, Boqun Yan, Jeffrey Naughton, and Jignesh M. Patel. Demonstration of Santoku: Optimizing Machine Learning over Normalized Data. *PVLDB*, 8(12):1864–1867, 2015.

[99] Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. Learning generalized linear models over normalized data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1969–1984, New York, NY, USA, 2015. ACM.

[100] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. To join or not to join?: Thinking twice about joins before feature selection. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 19–34. ACM, 2016.

[101] Angela Lee, Doris Xin, Doris Lee, and Aditya G. Parameswaran. Demystifying a dark art: Understanding real-world machine learning model development. *CoRR*, abs/2005.01520, 2020.

[102] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. Auto-fuzzyjoin: Auto-program fuzzy similarity joins without labeled examples. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1064–1076. ACM, 2021.

[103] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *arXiv preprint arXiv:1904.09483*, 2019.

[104] Yifeng Li and Alioune Ngom. Data Integration in Machine Learning. In *IEEE International Conference on Bioinformatics and Biomedicine (BTBM)*, 2015.

[105] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.*, 14(1):50–60, 2020.

[106] David Maier. *The theory of relational databases*, volume 11. Computer science press Rockville, 1983.

[107] Wes McKinney. pandas: a Foundational Python Library for Data Analysis and Statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[108] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. A comprehensive benchmark framework for active learning methods in entity matching. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1133–1147. ACM, 2020.

[109] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[110] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 19–34. ACM, 2018.

[111] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. Learning text similarity with siamese recurrent networks. In Phil Blunsom, Kyunghyun Cho, Shay B. Cohen, Edward Grefenstette, Karl Moritz Hermann, Laura Rimell, Jason Weston, and Scott Wen-tau Yih, editors, *Proceedings of the 1st Workshop on Representation Learning for NLP, Rep4NLP@ACL 2016, Berlin, Germany, August 11, 2016*, pages 148–157. Association for Computational Linguistics, 2016.

[112] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In Tobias Friedrich, Frank Neumann, and Andrew M. Sutton, editors, *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 485–492. ACM, 2016.

[113] Fatemah Panahi, Wentao Wu, AnHai Doan, and Jeffrey F. Naughton. Towards interactive debugging of rule-based entity matching. In Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß, editors, *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, pages 354–365. OpenProceedings.org, 2017.

[114] Judea Pearl and Thomas Verma. The Logic of Representing Dependencies by Directed Graphs. In *AAAI*, 1987.

[115] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[116] Jinglin Peng, Weiyuan Wu, Brandon Lockhart, Song Bian, Jing Nathan Yan, Linghao Xu, Zhixuan Chi, Jeffrey M. Rzeszotarski, and Jiannan Wang. Dataprep.eda: Task-centric exploratory data analysis for statistical modeling in python. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China*, 2021.

[117] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1723–1726, New York, NY, USA, 2017. ACM.

[118] Foster Provost and Pedro Domingos. Tree Induction for Probability-Based Ranking. *Machine Learning*, 52(3):199–215, 2003.

[119] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[120] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., 2003.

[121] Juan Ramos. Using TF-IDF to Determine Word Relevance in Document Queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. New Jersey, USA, 2003.

[122] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

[123] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282, 2017.

[124] Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. Overton: A Data System for Monitoring and Improving Machine-Learned Products. *arXiv preprint arXiv:1909.05372*, 2019.

[125] Steffen Rendle. Scaling Factorization Machines to Relational Data. *PVLDB*, 6(5):337–348, 2013.

[126] Robert Ricci, Eric Eide, and CloudLab Team. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *; login:: the magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[127] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[128] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 253–260, New York, NY, USA, 2002. ACM.

[129] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating Large-Scale Data Quality Verification. *Proceedings of the VLDB Endowment*, 11(12):1781–1794, 2018.

[130] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 3–18, New York, NY, USA, 2016. ACM.

[131] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Machine Learning: The High Interest Credit Card of Technical Debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[132] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2503–2511, 2015.

[133] Vraj Shah and Arun Kumar. The ML data prep zoo: Towards semi-automatic data preparation for ML. In Sebastian Schelter, Neoklis Polyzotis, Stephan Seufert, and Manasi Vartak, editors, *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning, DEEM@SIGMOD 2019, Amsterdam, The Netherlands, June 30, 2019*, pages 11:1–11:4. ACM, 2019.

[134] Vraj Shah, Arun Kumar, and Xiaojin Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *Proc. VLDB Endow.*, 11(3):366–379, 2017.

[135] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. Towards benchmarking feature type inference for automl platforms. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1584–1596, 2021.

[136] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. Towards Benchmarking Feature Type Inference for AutoML Platforms, Accessed March 22, 2021. `https://adalabucsd.github.io/papers/TR_2021_SortingHat.pdf`.

[137] Vraj Shah, Kevin Yang, and Arun Kumar. Improving Feature Type Inference Accuracy of TFDV with SortingHat, Accessed March 22, 2021. `https://adalabucsd.github.io/papers/TR_2020_TFDV.pdf`.

[138] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

[139] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[140] Abraham Silberschatz, Henry Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5 edition, 2006.

[141] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Generating concise entity matching rules. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1635–1638. ACM, 2017.

[142] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Synthesizing entity matching rules by examples. *Proc. VLDB Endow.*, 11(2):189–202, 2017.

[143] Survey. 2021 state of data science and machine learning. `https://www.kaggle.com/kaggle-survey-2021`, Accessed April 11, 2022.

[144] Ki Hyun Tae and Steven Euijong Whang. Slice Tuner: A Selective Data Collection Framework for Accurate and Fair Machine Learning Models. *arXiv preprint arXiv:2003.04549*, 2020.

[145] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[146] Ozge Uncu and I.B. Turksen. A Novel Feature Selection Approach: Combining Feature Wrappers and Filters. *Information Sciences*, 177(2), 2007.

[147] `https://aws.amazon.com/redshift/features/redshift-ml/`. Aws redshift ml, Accessed April 11, 2022.

[148] `https://aws.amazon.com/sagemaker/`. Aws sagemaker, Accessed April 11, 2022.

[149] `https://github.com/fabulousdj/autogluon`. Amazon autogluon integration with ml feature type inference models, Accessed April 11, 2022.

[150] `https://pypi.org/project/sortinghatinf`. Ml feature type inference usage on openml, Accessed April 11, 2022.

[151] `https://www.informatica.com/products/data-catalog/enterprise-data-prep.html`. Informatica enterprise data prep, Accessed April 11, 2022.

[152] `https://www.kaggle.com/surveys/2017`. 2017 kaggle survey on data science, Accessed February 15, 2018.

[153] `https://www.salesforce.com/video/1776007`. Salesforce einstein automl, Accessed May 30, 2022.

[154] `https://www.tableau.com/learn/get-started/prep`. Tableau prep, Accessed April 11, 2022.

[155] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. OpenML: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.

[156] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

[157] Vladimir Naumovich Vapnik. *The Nature of Statistical Learning Theory, Second Edition*. Statistics for Engineering and Information Science. Springer, 2000.

[158] Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. *Proc. VLDB Endow.*, 12(3):223–236, 2018.

[159] Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. *Proc. VLDB Endow.*, 12(3):223–236, 2018.

[160] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *Proc. VLDB Endow.*, 8(13):2182–2193, 2015.

[161] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009. ACM.

[162] S. K. M. Wong, , Cory J Butz, and Yang Xiang. A Method for Implementing a Probabilistic Model as a Relational Database. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, 1995.

[163] Alex Woodie. 2020 state of data science: Moving from hype toward maturity. `https://www.datanami.com/2020/07/06/data-prep-still-dominates-data-scientists-time-survey-finds`.

[164] Meng-Han Wu and Alexander James Quinn. Confusing the crowd: Task instruction quality on amazon mechanical turk. In *Fifth AAAI Conference on Human Computation and Crowdsourcing*, 2017.

[165] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. Zeroer: Entity resolution using zero labeled examples. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1149–1164. ACM, 2020.

[166] Doris Xin, Eva Yiwei Wu, Doris Jung Lin Lee, Niloufar Salehi, and Aditya G. Parameswaran. Whither automl? understanding the role of automation in machine learning workflows. In Yoshifumi Kitamura, Aaron Quigley, Katherine Isbister, Takeo Igarashi, Pernille Bjørn, and Steven Mark Drucker, editors, *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, pages 83:1–83:16. ACM, 2021.

[167] Cong Yan and Yeye He. Synthesizing Type-Detection Logic for Rich Semantic Data Types Using Open-Source Code. In *Proceedings of the 2018 International Conference on Management of Data*, pages 35–50, 2018.

[168] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Crossmine: Efficient classification across multiple database relations. In *Proceedings of the 2004 European Conference on Constraint-Based Mining and Inductive Databases*, pages 172–195, Berlin, Heidelberg, 2005. Springer-Verlag.

[169] Lei Yu and Huan Liu. Efficient Feature Selection via Analysis of Relevance and Redundancy. *Journal of Machine Learning Research*, 5, December 2004.

[170] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[171] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding Deep Learning Requires Rethinking Generalization. In *International Conference on Learning Representations (ICLR)*, 2017.

[172] Xiang Zhang and Yann LeCun. Text Understanding from Scratch. *CoRR*, abs/1502.01710, 2015.

[173] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657, 2015.

[174] Chen Zhao and Yeye He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 2413–2424. ACM, 2019.