An Investigation into Optimizers, Hyperparameters and Framework-Specific Settings for Machine Learning-Based Data Cleaning

Daniel van Dijke -
40508135

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Computer Science

School of Computing

October 9, 2024

Authorship Declaration

I, Daniel van Dijke, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed: Daniel van Dijke

Date: 21/04/2024

Matriculation no: 40508135

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.
Daniel van Dijke

Abstract

Data cleaning is fundamental for accurate data analytics and machine learning-based data cleaning frameworks reduce the expensive process of defining explicit rules for automated cleaning or repairing errors manually. However, the classifier these frameworks use to predict correct repairs is heavily influenced by the optimization algorithm selected and other internal parameters relevant to classification. In this investigation, a probabilistic error repair framework, *HoloClean*, was tested on a range of optimization algorithms, optimization hyperparameter values, and three framework-specific settings. These tests were performed on three benchmark datasets that differed in content and error characteristics, to produce findings so recommendations about selecting the best settings for *HoloClean* and similar frameworks can be made.

The Rprop optimizer produced the highest recall and F_1 across all datasets because it was the least sensitive to noise in training data. However, Adam-like optimizers achieved the highest precision when error rate was low due to conservative parameter updates. Regarding hyperparameters, the initial seed value did not affect performance and 10 training epochs was sufficient for convergence. A batch size of 1 was optimal as it allowed for the most parameter updates and a high momentum value of 0.9 was best because it performed well on noisy gradients. If recall and F_1 score are prioritised, a high learning rate of 0.01 and a low weight decay of 0 is optimal, but for better precision, a learning rate of 0.0001 and a high weight decay of 0.01 should be used.

Testing *HoloClean*-specific settings found the optimal domain pruning threshold of candidate repairs varied unpredictably for each dataset, the weak label threshold for including candidate repairs in training classifiers should be set to 0.99 so only the most likely candidates are incorporated, and Naive-Bayes was a better classifier for predicting correct repairs for weak-labelling than Logistic Regression.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Taoxin Peng and my personal tutor Ben Paechter for the advice they have given me for this project.

I would also like to thank my mum and dad for their endless patience and generosity.

Finally, I would like to thank the wonderful D2 regulars for making my time at Napier that much more enjoyable.

1 Introduction

Data cleaning is the process of detecting and correcting corrupt or inaccurate data and is vital if meaningful analysis is to be made from that data. This is because the analysis of large databases is centred around the identification of patterns in data, and errors create noise that leads to less accurate observations about these trends. The use of rule-based cleaning is usually an expensive process requiring a large degree of input from human users, each of whom must be well-informed about qualitative and quantitative constraints imposed on the relevant dataset.

In recent years techniques have been developed that use machine learning to allow for probabilistic data cleaning. These approaches leverage techniques to extract relevant features from the dataset used to train a learning model. Training this model based on these features and identified clean data allows it to determine the most important patterns in the dataset for detecting errors or identifying repairs.

To fit the model so it can effectively predict the probability that a candidate repair value is the correct repair, the internal parameters used to calculate the outputs of the model need to be adjusted so that the loss between the expected output and the actual output is minimised. This is done by generating outputs based on training data that is known to be clean and iteratively adjusting model parameters to reduce the loss for these cases. To help in minimising this loss a variety of algorithms called optimizers have been developed to decide how parameters should be updated. The resultant performance of the model can vary significantly depending on the optimizer used, making it important to test a variety of options, especially given that it is difficult to determine results based only on the theory of optimizers.

The aim of this project therefore was to select a relevant machine learning-based data cleaning framework and to test and compare a range of applicable optimizers, as well as testing a range of hyperparameters used by optimizers, to provide a set of recommendations about how to best apply these settings for software that uses approaches similar to the framework selected. There was also an additional goal to perform research into any framework-specific settings and provide findings and recommendations about these settings where suitable.

Specifically, the research questions that this project aims to address are:

1. How does the performance of error repair by a relevant machine learning-based data cleaning framework vary when tested using a range of op-

timizers, hyperparameter settings and framework-specific settings on benchmark datasets that vary in size, data types, error types and error rates?

2. What are the optimal optimizer, hyperparameters and framework-specific settings to use for the selected framework to best repair errors across a range of datasets, and are these optimal settings consistent across the datasets tested?

3. How does varying any relevant framework-specific settings affect the performance of the selected framework, and what information can be gained from this?

4. Given the evaluation of tests performed on this framework, what broader recommendations can be made about the user of optimizers, hyperparameters and probabilistic data cleaning settings for similar approaches that have been developed?

To address these research questions, a set of project milestones to be completed during this project are outlined. How these milestones were completed and the findings that were made from that process is what will be discussed in this dissertation. These milestones were:

1. To perform a critical review of the existing literature on machine learning-based data cleaning frameworks and a critical review of optimizers and hyperparameters. This includes comparing the theoretical and practical applications of optimizers in contexts that may be similar to those used in probabilistic data cleaning. This section will also provide the relevant background knowledge required for the concepts mentioned in this dissertation to be understood.

2. Based on the literature review, design a testing methodology that will objectively compare optimizers, hyperparameters and framework-specific settings. This includes selecting an applicable framework from those described in the literature and selecting relevant benchmark datasets that vary in characteristics (error rate, number of attributes, functional dependencies etc.) so that conclusions about the performance of different optimizers when can be made. It also requires the selection and implementation of suitable optimizers as well as the identification of the most relevant hyperparameters to test.

3. Present the results produced from testing in an informative manner and describe trends observed in these results. Then provide a discussion that uses relevant background knowledge to explain the results

achieved and the implications this has in terms of what settings would be recommended for use.

4. Provide a set of conclusions about the findings gained from the investigation performed, with emphasis on any broader recommendations that can be made about optimizers and hyperparameters used for similar probabilistic data cleaning frameworks. A description of the limitations of the investigation will then be provided, including suggestions as to how these limitations could be addressed with future work. Finally, a self-appraisal of the personal strengths and weaknesses displayed during this project will be provided.

2 Literature Review

2.1 Introduction

Data Cleaning, also known as data cleansing or data repairing, refers to the detection and elimination of errors identified in a data set (Mueller & Freytag, 2005). In order for businesses to be able to perform effective analyses and gain meaningful insights from the information stored in a database, it must be ensured that the dataset in question is accurate. The more accurate the data, the more reliable the results from analysing that data becomes. It is therefore important that the data entries stored in the dataset are "cleaned" to within a reasonable standard, a process whereby incorrect data entries are either removed or updated to contain acceptable values. In fact, "dirty" data, data that contains erroneous entries are one of the most common problems that individuals dealing with that data have to face, and the act of cleaning this data is a fundamental part of the preprocessing phase of data analytics. (Chu et al., 2016). This includes data used for training machine learning models, which rely on high quality data to be able to generate reliable predictions or insights (I. F. Ilyas & Rekatsinas, 2022).

The data cleaning process consists of two stages, the first is error detection, whereby various types of errors are identified, and the second is error repair, which is the act of editing entries to bring the dataset to a state containing few enough errors that it is useful for the required analysis or application. It is important that as little information is lost during the cleaning stage as is reasonably possible. Cleaning data without the use of automated processes, however, is highly time-consuming.

Automated data cleaning methods that do not incorporate machine learning are typically rule-based. This means that if a data entry causes a violation, such as that it is not of a specified data type, or if it does not fall within the specified range of admissible values, then that entry is either replaced or removed given instructions described by other rules. The main issue with this cleaning method, though, is that these rules often have to be specified by a data expert, which is expensive ((Chu et al., 2016)). Moreover, there are many errors that require complex rule sets to detect, such as a misspelled surname, or a duplicate entry, making it infeasible for humans to specify every rule.

Machine learning addresses this issue by allowing for data cleaning with less user-specified rules. A classifier is trained with clean data until it can identify the features that should exist without requiring an explicit rule set (I. F. Ilyas

& Chu, 2019).

For effective machine learning-based data cleaning, the model used must be able to quickly converge to an optimal set of parameter values so it can make accurate predictions about repairs. For this optimization algorithms, or optimizers, are used to minimise the loss function, the difference between expected and real outputs (Murphy, 2012).

Given different optimizers have varying strengths and limitations, it is necessary to investigate and determine which optimizer and relevant hyperparameters are most suitable to provide effective predictions. However, datasets vary in characteristics both in terms of content and error types, so it may not be that the same optimizers and hyperparameters work for every dataset, or indeed every data cleaning framework.

2.2 Aims

This literature review aims to provide a concise background on machine learning, and a description and critical review of existing machine learning-based data cleaning approaches. It also aims to provide a description of optimizers and critically review how popular optimizers perform in different conditions and how this may apply to machine learning-based data cleaning. Specifically, the aims are to:

- Provide a concise background on data cleaning and metrics used to evaluate data quality.
- Describe machine learning, critically comparing common machine learning classifiers used for tasks such as probabilistic data cleaning.
- Critically review machine learning-based data cleaning approaches described in the literature, comparing how these approaches perform on a variety of benchmark datasets.
- Provide a critical overview of optimizers used in machine learning, highlighting strengths and weaknesses of each that are likely to be relevant to the machine learning-based data cleaning approaches mentioned.

2.3 Defining Errors and Data Quality

There are several types of errors. These can be classified into syntactical, semantic, and coverage anomalies and are outlined by Mueller and Freytag (2005). Syntactical anomalies are errors where data entries are not of the correct format, of which there are three subtypes:

- Lexical errors, when the number of values provided by an entry does not match the number of attributes in the relevant table.
- Domain format errors, when a given value for an attribute doesn't match the format of that attribute. For example, the attribute "EMAIL ADDRESS" may require the inclusion of an "@".
- Irregularities, the non-uniform use of values, units and abbreviations, such as entering product prices, different currencies to specify the price of a product.

Semantic errors are errors that are correct in format but do not convey the correct meaning of the information being represented. This can be further broken down into:

- Contradictions, values that violate dependencies between one or more attributes, such as an entry containing a value in the attribute POST CODE that is not appropriate given the value stated in the attribute CITY.
- Duplicates, two or more entries referring to the same entity.
- Invalid tuples, the hardest error to detect. These are errors that do not violate any of the other errors listed and yet they do not describe the relevant entity correctly.

Coverage errors occur when not all entities that should be represented in the dataset are present or when there errors are not represented to the full extent. There are two subtypes:

- Missing values, when an entry has null values for an attribute where this should not be the case.
- Missing tuples, the absence of a data entry describing an entity that should be represented in the data.

2.4 Evaluation Metrics

To assess how well a data cleaning approach has eliminated errors in a real-world dataset requires a dataset containing errors and a clean dataset to compare to. One way to do this is to test a framework on a benchmark dataset, a dataset containing real-world data for which a clean dataset has already been generated. Another solution is to use or create synthetic datasets that emulate real-world data, but are generated in such a way that they can be

ensured to be free of errors. Errors are then injected in a controlled manner, so it can be known how many there are.

Assessing a framework's performance is typically described by precision, recall and F_1 (James, Witten, Hastie, & Tibshirani, 2013):

- Precision is the ratio of how many errors the model correctly identified, compared to the total number of errors the model identified.
- Recall is the ratio of correctly identified errors compared to the total number of errors in the dataset.
- The F_1 Score is a balanced metric that considers both precision and recall, is computed as:

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In other words, precision describes how many data entries were wrongly identified as erroneous, recall describes how many erroneous data entries were missed and the F_1 Score is a balance between them.

2.5 Machine Learning

According to (Mitchell, 1997)'s definition of machine learning:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997, p. 2).

In the context of data cleaning, the task is either the detection or repair of errors and and the performance measure the difference between expected output and real output. The experience is the clean training data, or ground truth, provided to the classifier. The classifier uses training cases provided to improve specified tasks, by adjusting internal parameters using an optimizer.

There are many types of classifiers used in machine learning, some commonly used are:

- Naive-Bayes: A probabilistic classifier that applies Bayes' theorem with strong independence assumptions between features (Bishop & Nasrabadi, 2006). Despite this being a very simplistic assumption, it

often still provides good results and typically requires far fewer training data. The disadvantage of this classifier however is that it is outperformed by other, more complex models when training data is not limited (Kelleher, Mac Namee, & D'arcy, 2020).

- Support Vector Machines: Models that address the issue of scalability first defining basis function centred on points in the training data and only selecting a subset of these points during training, reducing the order of operations required for nonlinear optimisation. They can be trained quickly, handle multi-dimensional data well, and are relatively resistant to overfitting. The drawback, however, is that the results produced are not easily interpreted, and it is often difficult to understand why a certain prediction was made (Kelleher et al., 2020).
- Decision Trees: Also known as Classification and Regression Trees, Decision Trees are Models which partition the input space into regions based on a sequential decision-making process, where each node in the tree represents a simple yes/no decision about a property of the data, and new input is classified by traversing the branches between these nodes in a top-down manner. However, decision trees can be relatively prone to overfitting and precautions have to be made to ensure that training data is not modelled perfectly (Murphy, 2012). The issue of overfitting can be dealt with to some degree using tree pruning, the removal of subtrees that were likely created by noise in the training data (Kelleher et al., 2020).

These classifiers are used to detect and repair errors based on soft (flexible) confidence margins, as opposed to hard constraints used in functional dependencies and integrity constraints. To train the classifier, the typical case is to construct a factor graph that represents the features present in the dataset. Weightings of hyperedges used in these graphs indicate how often this relation is present in the data. This factor graph is then converted into a format that can be used as input data to the classifier.

2.6 Machine Learning-Based Data Cleaning

The use of machine learning classifiers for data cleaning improves runtime and the range of errors that can be repaired. This section describes and critically compares the most popular machine learning-based data cleaning approaches described in the literature chronologically.

2.6.1 SCARE

SCARE (SCalable Automatic REpairing) is a systematic scalable framework that addresses the issues of relying on constraint-based repairing approaches for error repair (Yakout, Berti-Équille, & Elmagarmid, 2013). *SCARE* uses statistical machine learning techniques such as decision trees and Bayesian networks to capture dependencies and correlations in a dataset and uses that information to predict the correct value. *SCARE* is the first framework to use machine learning for error repair. Previous approaches have used machine learning, but only for data deduplication or the insertion of missing values.

SCARE does not perform error detection, it is assumed possible to identify a subset of clean attributes with the remaining attributes being marked as possibly dirty.

The hypothesis that *SCARE* uses for the error repair is that the more an update to a given cell will make the data follow the underlying data distribution, and the less the cost of the update, the more likely the update is to be correct. The cost of an update for a given value is measured using several metrics such as the Edit distance or the Jaro coefficient. The framework updates as many cells as it can without exceeding a user-specified cost constraint.

To accurately clean data, the challenge of having multiple erroneous attributes also had to be overcome. *SCARE* does this by modelling the probability distributions of the entire subset of attributes known to be dirty given the attributes that are known to be reliable, instead of just a single classification for a dirty attribute. This method means that inter-dependencies between erroneous attributes can be modelled effectively.

SCARE also increases scalability by processing several sections of the dataset in parallel, and models values of each attribute in those sections as a set of correlations to provide "local" predictions. It then combines these local predictions and updates errors if the confidence of the repair meets a given threshold. The final predicted values are typically those that would maximise the associations between the values of all relevant partitions.

Experiments were then performed to evaluate performance. Testing using a dataset on Intel Lab Data was used to assess how well *SCARE* predicted repairs compared to *ERACER*, a system that relies on relational learning. Both *SCARE* and *ERACER* showed high accuracy however *SCARE* does not require the expensive domain expert for the designing of a Bayesian Network

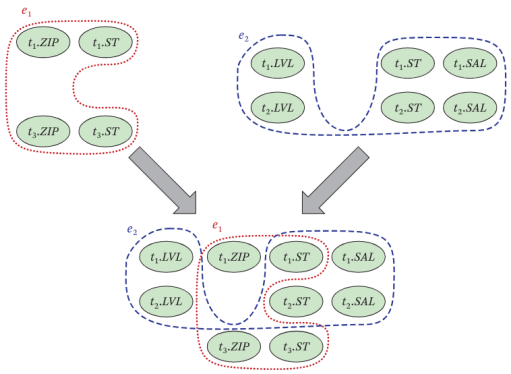

Figure 1: Holistic Error Detection (Chu et al., 2016, p. 152).

that *ERACER* does. Additionally, *SCARE* was tested for scalability on a dataset containing US Census Data. *SCARE* was able to scale linearly with dataset size when using its parallel section processing methodology, completing repair of one million rows in five minutes.

2.6.2 Holistic

Holistic is mostly rule-based but uses machine learning to address the issue of isolated error repair stages (Chu, Ilyas, & Papotti, 2013). This approach, which works with denial constraints, addresses the problem that develops when dealing with only one constraint at a time, the reduced level of information about what the repair for that cell might be. For example, it may be the case that the application of multiple separate denial constraints leads to the identification of an overlapping group of cells as an area that require repairs. However, there may not be enough information to identify the cell containing the erroneous value if the "dirty" groups of cells are only inspected one at a time. However, when information gathered from each error detection stage is combined, then erroneous entries can be better discovered by identifying cell(s) causing the most denial constraints, i.e., by identifying overlap between the "dirty" groups of cells (Fig. 1).

Holistic gathers information about every constraint violation first, and uses this increased level of context to more accurately identify errors and their potential repairs. The overlap between violations is compiled using a factor graph, where hyperedges link cells involved in the same constraint violation. During the repair process the cells that violate the most constraints are repaired first, because those cells are most likely to be erroneous. Once a cell is chosen for repair, each of their violations is processed to get information about how to repair it. After a repair is made, the hypergraph is updated to

remove constraint violations satisfied by the repair made. This process also means a high number of constraint satisfactions occur per repair, leading to less repairs having to be made.

The algorithm *Holistic* uses to select a new value to replace the erroneous entry works in two steps. First, a subset of possible values given the denial constraints imposed on relevant cells is generated. If that subset contains no values, constraints are relaxed incrementally until the subset is no longer empty. Second, the repair with the minimum edit distance is selected from this subset of candidate values. The edit distance is calculated in terms of distance between a candidate value and the original value and the number of cells involved in the repair.

Three datasets containing approximately 100K tuples of real world data each were used to test *Holistic* against other existing data cleaning frameworks. Compared to existing approaches, *Holistic* achieved higher precision and recall, while making fewer repairs.

However there are several drawbacks of this approach. First, it only addresses functional dependency errors and denial constraint errors. Second, it cannot derive denial constraints automatically, so a domain expert is required. Third, the generated hypergraph used to select repairs requires a lot of memory when repairing large databases, which has a significant impact on the runtime.

2.6.3 KATARA

KATARA uses machine learning to repair errors by leveraging knowledge bases and crowdsourcing (Chu et al., 2015). This reduces the need for master data or domain experts to solve complex errors.

KATARA uses an iterative cycle of crowdsourcing and pattern recognition algorithms, first prompting users assumed to be data experts if patterns that it has identified are correct, such as "Does Italy *hasCapital* Madrid?" However forming these prompts correctly still requires some human input. Although an algorithm used makes sure only the most useful questions are asked. The discovery of patterns in the table is performed using a directed graph, where edges represent relations between attributes. A rank-join algorithm is used to determine the top patterns in the table and *SCARE* then uses user-specified upper bounds and pruning methods to allow this algorithm to terminate reasonably quickly (I. Ilyas, Aref, & Elmagarmid, 2004).

Once the patterns in the table have been identified, data are categorised

as either correct or erroneous, based on how the data were validated by both inference from the knowledge bases and input from crowdsourcing. The identified patterns are then use the underlying assumption that the repair that left the new dataset closest to the original dataset was most likely to be correct.

KATARA was tested for performance on several real-world databases, such as information about football players. Two knowledge bases that had been generated by extracting information from Wikipedia were provided to the framework as reference. It was found when the knowledge bases had high coverage of the data in the dataset being tested, precision and recall of repairs was very high (¿0.95), but when coverage was low, performance dropped massively. When *KATARA* was compared to *SCARE* it was found that one approach was not better than the other, instead it was recommended they be used in tandem. *KATARA* is effective when knowledge bases have high coverage, while *SCARE* is effective when there is high data redundancy within the original dataset.

2.6.4 HoloClean

HoloClean (Rekatsinas et al., 2017) uses probabilistic inference to clean data. It uses the same error detection method as *Holistic*, where the user specifies denial constraints. Based on these denial constraints and the errors detected, *HoloClean* separates the dataset into "noisy" and "clean" cells. In the compilation stage, *HoloClean* uses a declarative probabilistic inference framework called *DeepDive* to identify the uncertainty of a value in a noisy cell (Shin et al., 2015).

In the repair phase, *HoloClean* first uses co-occurence statistics to generate a domain of candidate repair values for each "noisy" cell. The framework considers values if they exist in a row in the dataset that have matching attribute values as those that occur in the row containing the erroneous cell. These candidate values must also meet a user specified domain pruning threshold, which is a measure of many times that value co-occurs with the same attribute values as those present in the row of the noisy cell. A high domain pruning threshold leads to high precision. A lower threshold decreases precision but may increase recall.

Next, a classifier such as NaiveBayes is trained based on features identified in the dataset and is used to determine the probability that each value in the domain of potential repair candidates for a given erroneous cell is likely to be the correct repair. If the repair candidate with the highest probability

Dataset (τ)	Metric	HoloClean	Holistic	KATARA	SCARE
Hospital (0.5)	Prec.	**1.0**	0.517	0.983	0.667
	Rec.	**0.713**	0.376	0.235	0.534
	F1	**0.832**	0.435	0.379	0.593
Flights (0.3)	Prec.	**0.887**	0.0	n/a	0.569
	Rec.	**0.669**	0.0	n/a	0.057
	F1	**0.763**	0.0*	n/a	0.104
Food (0.5)	Prec.	**0.769**	0.142	1.0	0.0
	Rec.	**0.798**	0.679	0.310	0.0
	F1	**0.783**	0.235	0.473	0.0+
Physicians (0.7)	Prec.	**0.927**	0.521	0.0	0.0
	Rec.	**0.878**	0.504	0.0	0.0
	F1	**0.897**	0.512	0.0#	0.0+

* Holistic did not perform any correct repairs.
+ SCARE did not terminate after three days.
# KATARA performs no repairs due to format mismatch for zip code.

Figure 2: Performance of HoloClean compared to other data cleaning frameworks (Rekatsinas et al., 2017, p. 1198).

of being the correct repair exceeds a user-specified weak label threshold, this value is assigned as a weak label and replaces the original value in a dataset used for training in the next phase.

In the third phase, a featurized dataset is generated that approximates a factor graph where nodes in the graph represent each cell in the dataset, the values of these cells being those that were produced after the weak labelling process. The hyperedges of the factor graph relate to features between cells, including denial constraints, co-occurence relations and other patterns identified. The weight of each hyperedge has a weight that indicates the importance of this relation in determining cell values.

A custom single-layer neural network, the "tiedLinearLayer", is then trained based on information in the factor graph. Each node in this singular input/output layer uses the same weights although biases may be assigned to each output classes individually. This model produces a logit, a vector containing the log-likelihood of each repair candidate being the correct repair (Demaris, 1992). The softmax activation function is then used to normalize the logit to a vector of probabilities between 0 and 1, and the class (candidate repair value) with the highest probability is assigned as the correct repair. The default optimizer used to update the shared parameters of the tiedLinearLayer is the Adam optimizer (Kingma & Ba, 2014).

HoloClean was compared with *Holistic* for the repair of constraint violations as well as against *KATARA* more generally. *HoloClean* had higher precision, recall and F_1, more than $2\times$ across all four datasets tested (Fig. 2). *HoloClean* does, however, take longer to perform its data cleaning process than *Holistic* and *KATARA*, although not orders of magnitude longer, i.e. hours rather than days(Fig. 3).

Dataset	HoloClean	Holistic	KATARA	SCARE
Hospital	147.97 sec	5.67 sec	2.01 sec	24.67 sec
Flights	70.6 sec	80.4 sec	n/a	13.97 sec
Food	32.8 min	7.6 min	1.7 min	-
Physicians	6.5 hours	2.03 hours	15.5 min	-

Figure 3: Runtime of *HoloClean* compared to similar data cleaning frameworks (Rekatsinas et al., 2017, p. 1198). A dash indicates failure to terminate after 3 days.

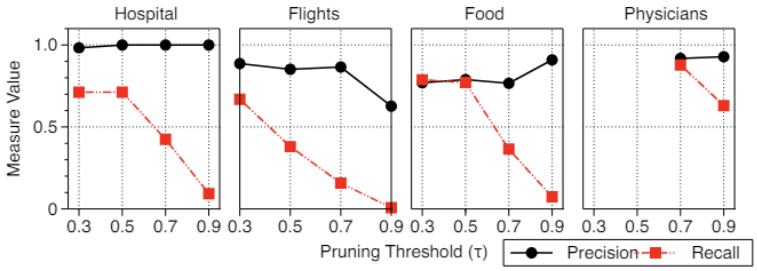

Figure 4: Effect of domain pruning threshold on precision and recall (Rekatsinas et al., 2017).

Rekatsinas et al. (2017) also found that *KATARA* obtains high precision but low recall due to the limited coverage of external datasets whereas *SCARE* had high performance when redundancy in the dataset was high. This is likely because high redundancy means learning models can be trained better. *SCARE* also only uses integrity constraints so that could have also reduced performance in other areas.

The robustness of *HoloClean* was assessed when cells known to be correct were marked as erroneous. It was found that even when 20% of correct cells were marked erroneous, precision only decreased by 7%.

The domain pruning threshold for generating weak labels was also assessed. Figure 4 shows that this threshold produced a tradeoff between recall and precision and that it would require dataset-specific tuning.

2.6.5 Raha

Raha only works on error detection and not repairs and is designed to minimise user interaction requirements. It does so by parsing the dataset and generating feature vectors for each data entry. These feature vectors are a

Approach	Hospital			Flights			Address			Beers			Rayyan			Movies			IT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
dBoost	0.54	0.45	0.49	0.78	0.57	0.66	0.23	0.50	0.31	0.54	0.56	0.55	0.12	0.26	0.16	0.18	0.72	0.29	0.00	0.00	0.00
NADEEF	0.05	0.37	0.09	0.30	0.06	0.09	0.51	0.73	0.60	0.13	0.06	0.08	0.74	0.55	0.63	0.13	0.43	0.20	0.99	0.78	0.87
KATARA	0.06	0.37	0.10	0.07	0.09	0.08	0.25	0.99	0.39	0.08	0.26	0.12	0.02	0.10	0.03	0.01	0.17	0.02	0.11	0.17	0.14
ActiveClean	0.02	0.14	0.03	0.28	**0.94**	0.44	0.14	**1.00**	0.25	0.16	**1.00**	0.28	0.09	**1.00**	0.16	0.02	0.01	0.01	0.20	**1.00**	0.33
Raha	**0.94**	**0.59**	**0.72**	**0.82**	0.81	**0.81**	**0.91**	0.80	**0.85**	**0.99**	0.99	**0.99**	**0.81**	0.78	**0.79**	**0.85**	**0.88**	**0.86**	**0.99**	0.99	**0.99**

Figure 5: Performance of Raha compared to similar frameworks (Mahdavi et al., 2019, p. 11)

binary list that are populated by running a variety of error detection algorithms on the database. These detection algorithms were selected to cover detection for all major error types. Each element in the feature vector relates to one of those algorithms, where a 1 indicates at least one of the algorithms identified that cell as erroneous, and a 0 means the opposite.

Raha then clusters cells together based on similarity between feature vectors are. Cells listed with similar feature vectors are likely to have similar properties, including, if they are errors, to be of the same error type. The user is then prompted to label one representative data entry from each cluster. *Raha* then labels every cell with the same feature vectors as either "dirty" or "clean" depending on what the user specified.

The impact this clustering process makes is that *Raha* only requires a handful of tuples to be labelled, at most 20, and is able to use clusters of cells now considered "dirty" or "clean" based on the labelling and updating cycle as training data for a machine learning model. This model is used to detect erroneous cells that had different feature vectors as the labelled cells. These are typically cells on the edges of generated clusters.

However this framework *Raha* takes longer to run than other frameworks because of how many features that have to be generated. It does, however, outperform other data cleaning frameworks (Fig. 5).

2.6.6 Baran

Baran uses machine learning as a component of its overall cleaning process and continues the work undertaken with *Raha*, forming an end-to-end pipeline of error detection and repair (Mahdavi & Abedjan, 2020, 2021). This paradigm iteratively prompts the user to label a row with erroneous values from the dataset and updates the correction models incrementally. It leverages all three data error contexts: the value of data errors, the vicinity of data errors in the dataset and the domain of data errors.

Baran main contribution over existing frameworks includes is that it reduces

System	Hospital	Flights	Address	Beers	Rayyan	IT	Tax
KATARA	234	116	5739	180	134	2031	15992
SCARE	76	123	11853	363	216	8717	55495
Holistic	15	10	69	9	8	3	247
HoloClean	148	39	17582	96	112	885	25778
Baran	23	22	11073	114	26	247	11936

Figure 6: Runtime (seconds) of *Baran* compared to Benchmarks (Mahdavi & Abedjan, 2020, p. 1957)

System	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
KATARA	0.98	0.24	0.39	0.00	0.00	0.00	0.79	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.01	0.02	0.59	0.01	0.02
SCARE	0.67	0.53	0.59	0.57	0.06	0.11	0.10	0.10	0.10	0.16	0.07	0.10	0.00	0.00	0.00	0.20	0.10	0.13	0.01	0.01	0.01
Holistic	0.52	0.38	0.44	0.21	0.01	0.02	0.41	0.31	0.35	0.49	0.01	0.02	0.85	0.07	0.13	1.00	0.78	0.88	0.96	0.26	0.41
HoloClean	1.00	0.71	0.83	0.89	0.67	0.76	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.01	0.01	0.11	0.11	0.11
Baran	0.88	0.86	0.87	1.00	1.00	**1.00**	0.67	0.32	**0.43**	0.91	0.89	**0.90**	0.76	0.40	0.52	0.98	0.98	**0.98**	0.84	0.78	0.81
Baran (with TL)	0.94	0.88	**0.91**	1.00	1.00	1.00	0.67	0.32	0.43	0.94	0.87	0.90	0.80	0.44	**0.57**	0.98	0.98	0.98	0.95	0.73	**0.83**

Figure 7: Performance of *Baran* Compared to similar data cleaning approaches (Mahdavi & Abedjan, 2020, p. 1957). P = precision, R = recall, F = F1 Score.

the expensive user labelling process by leveraging transfer learning. This is the process of learning from external sources first, providing context for training classifiers working on similar (but not the same) data. In this case the source is the Wikipedia Page Revision History, a list of the edits made to correct mistakes on Wikipedia. This helps to solve simple grammatical mistakes such as the use of "Holland" instead of "The Netherlands." Pairing this strategy with labelled user input means *Baran* doesn't require any user-specified rules or statistical parameters which are steps required by *SCARE*, *Holistic*, *HoloClean*, and *KATARA*.

Baran was tested against comparable frameworks on 7 benchmark datasets containing 4 error types. The other frameworks were provided with as many integrity constraints and knowledge bases (in the case of *KATARA*) as reasonably possible, while *Baran* was allowed to prompt the user up to 20 times.

Runtime for *Baran* was one of the lowest across all datasets (Fig. 6), and the F_1 was dominant over other approaches (Fig. 7), even without transfer learning. *Baran* also maintained a high F_1 when error rate increased because it leveraged trustworthy contexts of the data more effectively.

Algorithm	E	Tax			Hospital		
		P	R	F1	P	R	F1
Horizon	E1	**0.81**	0.74	**0.76**	**0.93**	**0.77**	**0.84**
	E2	0.8	**0.27**	**0.38**	0.88	0.61	0.71
Holistic	E1	0.16	**0.87**	0.25	0.04	0.28	0.06
	E2	0.22	0.19	0.19	0.48	0.04	0.03
SAMP	E1	0.08	0.34	0.09	0.20	0.29	0.20
	E2	0.12	0.08	0	0.24	0.43	0.29
Unified	E1	0.12	0.01	0.01	0.82	0.66	0.73
	E2	**1.0**	0	0	0.61	0.7	0.65
Min	E1	0.32	0.7	0.43	0.42	0.6	0.49
	E2	0.29	0.26	0.27	**0.99**	**0.76**	**0.85**

Algorithm	Parking			DataX		
	P	R	F1	P	R	F1
Horizon	**0.98**	**0.56**	**0.7**	0.93	**0.93**	**0.93**
Holistic	0.43	0.12	0.18	1.0	0.08	0.14
SAMP	0.14	0.02	0	0.2	0.32	0.24
Unified	0.42	0.1	0.16	0.71	0.06	0.11
Min	0.1	0.04	0.05	**0.95**	0.66	0.77

Figure 8: Performance of *Horizon* compared to other benchmark frameworks. (Rezig et al., 2021, p. 2552)

2.6.7 Horizon

Horizon (Rezig et al., 2021) addresses scalability issues past approaches had using a linear-time cleaning algorithm allowing for scaling up to millions of records. However *Horizon* is only designed to deal with functional dependencies. The repair method used is to select repairs that preserve the most frequent patterns found in the original data. Like other frameworks, a feature graph is generated but *Horizon* selects repairs that preserve relations (graph edges) with the highest weights. The scalability of this approach roots from a topological sorting method that is applied to the feature graph, allowing for linear-time graph traversal.

When tested for performance against other rule-based frameworks, *Horizon* has the highest F_1 on almost every dataset used (Fig. 8).

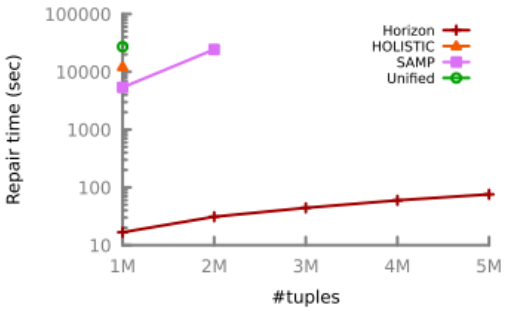

Figure 9: Repair time of *Horizon* compared to similar data cleaning approaches (Rezig et al., 2021, p. 2253)

Horizon was also tested against non-rule-based approaches such as *HoloClean* and *Baran*. It had a lower F_1 when data redundancy was lower or included more error types than just functional dependency errors. However it was also shown the runtime was 3 orders of magnitude faster than other frameworks (Fig. 9).

2.6.8 Summary of Machine Learning-Based Data Cleaning

Table 1 summarises the strengths and limitations of each data cleaning approach mentioned in this section.

Framework	Advantages	Disadvantages
SCARE	• Can repair detected errors accurately compared to past frameworks. • Scales linearly for large datasets. • Especially effective when the dataset has a high degree of redundancy.	• Does not cover error detection, only error repair. • Does not take pre-specified integrity constraint rules into account, so cannot use that information to improve speed and accuracy of repair.
Holistic	• Improves performance by taking all constraint violations into account before making repairs. • High precision and recall for a rule-based approach. • Repairs datasets with less edits than other rule-based software.	• Requires manual rule insertion. • Only deals with functional dependency errors and denial constraint errors. • The feature graph it uses leads to higher memory usage for a rule-based approach leading to scalability issues.
KATARA	• Can utilise knowledge bases and crowdsourcing for improved accuracy, reducing the need for master data. • Good performance when knowledge bases have high coverage.	• Effectiveness heavily reliant on knowledge base coverage. • Requires an external dataset to perform repairs.
HoloClean	• Handles both error detection and error repairs. • Can leverage both rule-based cleaning and machine learning techniques. • User can edit confidence thresholds for repairs. • Outperforms KATARA, SCARE and Holistic under most conditions. • Robust performance, even with a high degree of errors.	• Longer runtime compared to other approaches. • Still relies on integrity constraints as specified by the user. • Does not have the option of prompting the user to label pairs useful to the repair process • Uses similar methodology as *Holistic* for error detection, leading to same scalability issues.
Raha	• Requires very little user interaction. •Does not require external ground truth dataset or knowledge base. Outperforms other frameworks developed previously (e.g. HoloClean).	• Average runtime is much longer than other mentioned approaches due to generation of feature vectors. • Can only perform error detection, not repairs.
Baran	• Can effectively use external, similar datasets to improve classifier accuracy. • Leverages the trustworthy contexts of data errors more effectively than other approaches. • More accurate repairs even without being provided the external data. • Low runtime on benchmark data compared to other approaches.	• Requires a low level of user prompting to make accurate repairs. • Not designed to deal with error detection, only error repair.
Horizon	• Scales to millions of records, using approximately linear time complexity in respect to the number of records. •Approximately three orders of magnitude faster than other approaches. •Very effective at repairing functional dependency errors.	•Designed only to deal with functional dependency errors. •Less effective for datasets with low data redundancy or multiple types of errors.

Table 1: Comparison of Machine Learning-Based Data Cleaning Frameworks

2.7 Optimizers

Optimizers, are used to to find values for the set of weights and biases used in the classifier, or learning model, to minimize a cost or loss function which calculates the difference between the expected output and the actual output of the classifier (Murphy, 2022). These optimizers use the gradients of the loss function calculated for each weight during backpropagation to iteratively update model parameters such that the loss is reduced and, as a result, the classifier produces more accurate predictions.

By mapping the partial second derivatives of the loss function of a parameter a symmetric scalar field known as the Hessian Matrix can be produced that the optimizer must traverse in order to reach the minimum loss possible (Bishop & Nasrabadi, 2006). If the Hessian Matrix is known, then the shortest path to the global minimum can be calculated easily. However, to generate the Hessian Matrix is computationally expensive. Instead, most optimizers use the first-order derivative, or gradient, of the loss function instead to help the model converge to a low level of loss. However there are nonlinear optimization algorithms, such as Limited-memory BFGS (LBFGS), that can take the properties of the Hessian Matrix into account when updating parameters (Bishop & Nabney, 2008).

Optimizers are typically reliant on several hyperparameters that impact how appropriately the parameters of the learning model can be fitted to the prediction problem. These hyperparameters have to be tuned to gain optimal performance, some of the most common of which are:

1. Learning Rate, or step size, the rate at which updates to the weights are made. If learning rate is high the model can converge quickly however it risks overshooting the loss minima, whereas if it is too low then convergence not occur within the time allocated for training. The learning rate can be applied using various methods, although commonly it is set as a small constant or reduced by a constant γ every time a specified threshold is reached, a technique known as step decay (Goodfellow, Bengio, & Courville, 2016; Murphy, 2022).
2. Weight Decay, a form of regularisation that applies a penalty based on the magnitude of a parameter (Murphy, 2022). If the magnitude of the parameter is high it means the fitted model is more complex, increasing the likelihood of overfitting. The greater the parameter's magnitude, the higher the weight decay.
3. Momentum. Used in optimizers such as Stochastic Gradient Descent

(SGD) and RMSprop, this parameter addresses issues including slow learning, small consistent gradients, or noisy gradients (Goodfellow et al., 2016; Polyak, 1964). It accelerates learning by taking previous gradients into account when updating a new value. It adds a fraction (usually 0.9) of the previous update vector, or velocity, to the current update, which has the effect of increasing the learning rate in the general direction of traversal across the Hessian Matrix, speeding up convergence. If tuned too sensitively however, there is a risk of overshooting minima and updating weights with the wrong direction.

4. Number of Epochs. An Epoch is one complete iteration of the training set used, although the number of updates varies based on batch size. If the number of epochs increases, then the increased number of weight updates could lead to better convergence but this is at the expense of runtime.

5. Batch Size, or sometimes mini-batch size is the size of the subset of the training data used to calculate the gradient of the loss function before parameters are updated. If batch size is large then a better estimate of the loss function is made but this takes more time and if the number of epochs is fixed then fewer updates to the weights will be performed, assuming that the training cases in each mini-batch do not overlap. However samples can also be chosen randomly for each mini-batch (Murphy, 2022), which could prevent overfitting but also means it cannot be guaranteed the model is trained on all samples, leading to less coverage.

6. Seed, which determines the initial parameters of the model. Given a sufficient number of epochs and an adequate optimizer, the probability that the seed will impact the end results significantly is low, however studies on deep learning architectures found that there are likely still outliers (Picard, 2021).

Hyperparameters need tuning for their specific use cases to find optimal settings that allow for a suitable tradeoff between speed and accuracy when fitting a model. This can either be done manually or by using automatic hyperparameter optimization algorithms. Two of the most common automated hyperparameter tuning algorithms include grid search and random search, of which random search has been found to provide appropriate hyperparameter configurations with much less computational power (Bergstra & Bengio, 2012). Other algorithms such as greedy sequential methods have also been applied to this problem as computational power has become more readily available in the last decade (Bergstra, Bardenet, Bengio, & Kégl,

2011). However, if there is not enough available time to tune all hyperparameters, the learning rate is the most important to tune first as it affects the capacity that the model will have to match the complexity of its given task the most (Goodfellow et al., 2016).

Optimizers have different strengths and limitations in practical applications, given the factors they take into account when updating weights.

2.7.1 Stochastic Gradient Descent

The simplest optimizer, Stochastic Gradient Descent (SGD) computes the average gradient of the loss for all training cases in the mini-batch. The weight update algorithm for SGD, as detailed by Goodfellow et al. (2016) is:

Algorithm 1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ε_k.
Require: Initial parameter θ.
  while stopping criterion not met **do**
    Sample a mini-batch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
    Apply update: $\theta \leftarrow \theta - \varepsilon_k \mathbf{g}$
  end while

Because only a subset of the training data is used the loss gradient calculated will differ from loss gradient of the entire training set. This introduces stochasticity and can help to escape local minima because a stationary point with respect to the loss of one parameter may not be a stationary point for the whole training set (Bishop & Nabney, 2008). This makes the learning rate a crucial part of SGD because the stochasticity introduced from mini-batch sampling does not disappear even as the model converges to a minimum. The learning rate is often decayed over time to ensure this issue does not become more influential as the gradient plateaus (Goodfellow et al., 2016). SGD also takes momentum into account to overcome the problem of getting stuck in local minima.

Given only a small batch size is used for each iteration, SGD has been found to have fast performance when dealing with large training sets (Bottou, 2010). Using a small batch size requires more update iterations but this is insignificant compared to the time required to assess the entire data set. One issue

with SGD, however, is that the stochasticity introduced from mini-batching can lead to oscillation around an ideal minimum (Sun, Cao, Zhu, & Zhao, 2019). This oscillation can partially be dealt with by increasing the size of the mini-batch to include multiple samples (Robbins & Monro, 1951).

2.7.2 Averaged Stochastic Gradient Descent

A close variant of SGD, Averaged Stochastic Gradient Descent (ASGD) performs parameter updates using the same methods as SGD however the final parameters output by this optimizer take into account the parameters set in previous iterations. ASGD typically returns a weighted average of the parameters after a specified number of iterations, with new updates typically having a higher weight than older updates (Polyak & Juditsky, 1992). This mitigates the problem of SGD oscillating around the point of minimum loss due to the stochasticity created from small batch sizes. However, the increased complexity does cause the limitation of having to tune the time step determining when parameter averaging begins, and what the weighting of the parameters used in new updates are compared to the older parameters.

2.7.3 Adagrad

The adaptive subgradient method, or Adagrad (Duchi, Hazan, & Singer, 2011), is similar to SGD only it adapts the learning rate for each parameter. The new equation for the parameter update rule therefore becomes:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\alpha}{\varepsilon + \sqrt{s_i^{(k)}}} g_i^{(k)}$$

Where x_i is the weight being updated, k is the iteration, α is the learning rate, ϵ is a small term to prevent division by zero and $s^{(k)}$ is a vector of which the ith entry is the sum of squares of all the past gradients used for that parameter during training (Kochenderfer & Wheeler, 2019). At each training iteration, the square of the specified partial derivative of the parameter is added to this vector. Weights that have the largest loss gradient have a large decrease in the learning rate, whereas weights with small gradients only have a small decrease in the learning rate. As a result, there is greater movement in directions of the parameter space that have more smoothly sloped gradients (Goodfellow et al., 2016). This is a desirable property when gradients are sparse because good progress can be made in a given direction with few updates. However, there are drawbacks. The first is that because Adagrad adapts the learning rate, it can cause it to decrease to a negligible

amount leading to premature convergence due to the accumulation of the squared gradients from earlier iterations. As a result, empirical studies show that Adagrad can perform well for some learning models but not all of them (Goodfellow et al., 2016; Kochenderfer & Wheeler, 2019).

2.7.4 RMSprop

RMSprop (Root Mean Square Propagation)(Tieleman, 2012) is an optimizer that extends Adagrad by negating the issue of the monotonically decreasing learning rate. It adds an extra term to the update equation that maintains a decaying average of the squared gradients introduced by Adagrad (Kochenderfer & Wheeler, 2019). This decay is typically set to just less than 1.0. If the hyperparameters are tuned correctly, RMSprop has been shown to converge at an appropriate rate when used in applied machine learning contexts (Shi & Li, 2021). The downside of this optimizer, however, is that because it also incorporates momentum alongside the additional gradient decay term, RMSprop is sensitive to many hyperparameters and may require a significant degree of hyperparameter tuning to find a successful optimum. Additionally, theoretical studies have not been able to determine the conditions necessary for RMSprop to be guaranteed to converge inenvironments that do not have one global minimum (The Hessian Matrix is symmetrical, therefore two global minima can exist) (Zou, Shen, Jie, Zhang, & Liu, 2019).

2.7.5 Adadelta

Adadelta is very similar to RMSprop, except that by introducing an exponentially decaying average of the squared gradient updates, the learning rate parameter can be removed entirely(Zeiler, 2012). This means it does not require manual tuning in the same way that RMSprop does. This makes it more robust when working with noisy gradients and it works well on a variety of learning models (Zeiler, 2012). It also has the same benefits of RMSprop in being able to deal with the learning rate reducing too quickly that Adagrad suffers from, however like RMSprop because the learning rate does not decrease with time it cannot be guaranteed that this optimizer will converge to a solution (Murphy, 2022). The inability to fine-tune certain hyperparameters may also be a downside under certain conditions. For example, if the rate of convergence is not high enough, the learning rate cannot be adjusted manually to account for this.

2.7.6 Adam

Adaptive movement estimation (Adam) (Kingma & Ba, 2014), is an optimizer that builds on those mentioned previously. It combines the properties of momentum used in SGD with the moving average of the squared differentials used in RMSprop. However, there are a few differences. The first is that Adam incorporates momentum directly as an estimate of the first-order moment, an exponentially decaying average of the gradients (Goodfellow et al., 2016). The update rule for the first order moment is:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Where m_t is the first order moment, t is the time step, β_1 is the exponential decay rate and g_t is the gradient at time t (Murphy, 2022). Second, Adam uses biases to correct estimates of the first-order and second-order moments. The second-order moment refers to the update of the decaying average of the squared gradients:

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) g_t^2$$

Where s_t is the second-order moment, β_2 is a decay factor and g_t^2 refers to the square of the gradient at time step t (Murphy, 2022). The values for the two bias corrections, β_1 and β_2, are typically set to a default of 0.9 and 0.999 respectively (Kingma & Ba, 2014), and it is considered that besides the learning rate, the hyperparameters used are generally quite robust (Goodfellow et al., 2016).

Adam is one of the most commonly used optimizers in machine learning, especially for deep learning because its properties make it adaptable to many contexts (Norouzi & Ebrahimi, 2019). However, it shares limitations with RMSprop in that it has been shown that convergence does not occur under certain conditions although there are variants of Adam designed to address this issue(Reddi, Kale, & Kumar, 2019). Theoretically, optimizers that Adam has all of the properties of, such as SGD with momentum, should never outperform Adam. One comparison study showed it is actually inadequate hyperparameter tuning that leads to any empirical data showing otherwise (Choi et al., 2019).

The inclusion of several optimization techniques does increase the computational resources Adam requires given information about past gradients must be stored for each parameter.

2.7.7 Adamax

Adamax is a variant of Adam which, rather than calculating the second-order moment it calculates the infinity norm (Kingma & Ba, 2014). The infinity norm u_t is the maximum of the absolute value of the current gradient $|g_t|$ of a parameter and the previous infinity norm u_{t-1} after having been affected by a decay rate β_2:

$$u_t = \max(\beta_2 \cdot u_{t-1}, |g_t|)$$

The effect of using the infinity norm instead of the second-order moment is that Adamax performs well on sparser gradients and it typically converges more stably than Adam. However, this convergence is generally also slower (Yi, Ahn, & Ji, 2020). Like Adam and other adaptive subgradient methods, it has been applied to a broad range of machine learning tasks (Stock et al., 2022; Thapa et al., 2020), despite sharing the same convergence limitations as Adam and RMSprop.

2.7.8 AdamW

AdamW is a variant of Adam that decouples the weight decay component from the optimization steps (Loshchilov & Hutter, 2017). Adam first applies weight decay to the gradient of the parameter and then performs subsequent update steps such as calculating first-order and second-order moments. By contrast, AdamW applies the weight decay directly to the parameter at the end of the update process. According to Loshchilov and Hutter (2017), this change means that learning rate and weight decay hyperparameters are no longer dependent on each other when tuning hyperparameters and that AdamW outperforms Adam when it comes to generalization, the ability to correctly classify data different to the training data (Bishop & Nabney, 2008). The reduced errors in generalization have recently been proven, both theoretically and experimentally (Zhou, Xie, Lin, & Yan, 2024). However, because it is still similar to Adam it suffers many of the same limitations. For example, studies have shown that while it should outperform SGD and other optimizers it covers the functionality of in terms of minimising the loss, Adam-like optimizers are still worse at generalization than SGD (Chen et al., 2018; Keskar & Socher, 2017). They achieve faster convergence during training but plateau faster on test data (Zhou et al., 2020).

2.7.9 Rprop

Resilient Propagation (Rprop) differs significantly from both Adam and SGD (Riedmiller & Braun, 1993). The learning rate is the only significant hyper-parameter used. It focuses only on the sign (positive or negative) of the gradient. If the sign of the gradient is the same as the sign of the gradient during the last iteration, then a greater update is made, and these step sizes increase until a specified maximum step size. However, if the gradient sign differs from the sign in the last iteration a minimum has been overshot. So the learning rate is reduced, leading to smaller parameter updates, down to a specified lower bound, allowing the model to converge back to this minimum stably.

According to Igel and Hüsken (2003), Rprop is fast, accurate and robust with respect to hyperparameter settings. They also state Rprop is very suitable when training data contains noise. It is also simple to implement due to the lack of hyperparameters requiring less tuning. It is also more resilient to the problem of negligible parameter updates being made on smooth gradients, as is the case for SGD with low momentum and Adagrad.

However, Rprop does not consider the loss when reversing an update due to a change in gradient sign. This means that if a parameter update decreases the overall loss but the gradient of that parameter has now changed, Rprop will perform an update in the reverse direction, potentially increasing the loss again (Igel & Hüsken, 2000). Rprop is also usually not considered for mini-batch learning because changes to the gradient are not averaged correctly across training cases (Hinton, Srivastava, & Swersky, 2012).

2.7.10 LBFGS

Limited-Memory BFGS (LBFGS) is a quasi-Newton second-order optimizer. It uses information regarding second derivatives of the loss by generating estimates of the Hessian Matrix in a manner that is not computationally limiting (D. C. Liu & Nocedal, 1989). However, this is still with much higher memory requirements than first-order optimizers. Unlike full BFGS where information about the gradient at each time step is used to estimate the Hessian, LBFGS only stores information from the last M time steps to perform the estimation. Therefore memory complexity is $O(MD)$ where D is the number of parameters in the model (Murphy, 2022). Empirical evidence has shown that even small values of M, between 3 and 20, can lead to satisfactory results (Wright, 2006).

Studies have also shown that LBFGS has lower convergence times compared

to SGD-like optimizers and theoretical work shows that LBFGS will converge with a probability of 1 assuming there are bounds on the eigenvalues of the Hessian Matrix, meaning flat or overly steep regions of the loss landscape are avoided (Mokhtari & Ribeiro, 2015). But LBFGS is not suited for mini-batch processes, making it less suited to large-scale machine learning tasks (Berahas, Nocedal, & Takác, 2016; Bollapragada, Nocedal, Mudigere, Shi, & Tang, 2018). It also requires a closure method to be included in the optimization process so the overall loss can be stored after each update, making implementation harder.

2.7.11 Summary of Optimizers

A summary of the strengths and limitations of each optimizer has been outlined in Table 2.

Table 2: Comparison of Optimization Algorithms

Optimizer	Strengths	Limitations
SGD	• Applicable to mini-batching • Generalizes better than adaptive subgradient methods • Fast performance, scales well to large datasets • Easy to implement • Can use momentum to escape local minima	• Prone to oscillation, prevent convergence • Requires tuning of learning rate
ASGD	• Can prevent oscillation around local minima • Shares same generalization properties as SGD.	• Requires tuning of point at which to start averaging parameters.
Adagrad	• Performs well on sparse gradients • Converges faster than SGD on smooth gradients	• Learning rate can be adapted too quickly, preventing convergence • Applicable to less models than SGD
RMSprop	• Addresses learning rate issue Adagrad suffers from • Shown to converge well in applied contexts	• Sensitive to many hyperparameters that require tuning • Not proven to converge in environments that do not have one global minimum

Adadelta	• Does not require tuning of learning rate • More robust than RMSprop on noisy gradients • Works well on a variety of learning models • Deals with Adagrad issue of learning rate decreasing too quickly	• Not guaranteed to converge because learning rate does not decrease with time • Lack of hyperparameters means manual training cannot occur for specific applications
Adam	• Applicable to many machine learning contexts • At least same functionality of SGD if tuned correctly	• Convergence does not occur under certain conditions • Increased computational resources compared to SGD • Worse at generalization than SGD • Can plateau on test data
Adamax	• Applicable to many machine learning contexts • More stable convergence than Adam • can handle sparse gradients better than Adam	• Converges more slowly than Adam • Suffers from same tied learning rate and weight decay issues that Adam has
AdamW	• Learning rate and weight decay not tied when tuning, unlike Adam • Outperforms Adam in Generalization	• Worse at generalization than SGD • Fast progress in training but plateaus on test data
Rprop	• Only requires tuning of learning rate • Fast, accurate and less sensitive to hyperparameters • Resilient to smooth gradients unlike SGD • Suitable for noisy gradients • Simple to implement	• Does not consider loss when updating parameters, so could reverse a beneficial update • Typically not applicably to mini-batching

LBFGS	• Can estimate Hessian Matrix to converge more quickly • Uses less memory than other second-order optimizers •Has been proven to converge under non-complex conditions	• Not suited for mini-batching • Does not scale well compared to first-order optimizers • Requires implementation of closure method • Requires more memory than first-order optimizers

2.8 Summary of Literature Review

Machine learning can help to reduce the resources required for data cleaning by reducing the user input needed to detect and repair errors. Optimizers help to perform this process more accurately. This literature review has provided a concise background on data cleaning, the errors involved and the evaluation metrics used in determining data quality. It has also provided a description of machine learning, including the strengths and limitations of common classifiers such as Naive-Bayes, Logistic Regression and SVM.

A critical review of machine learning-based cleaning frameworks has also been performed. It was found that more modern frameworks such as *Horizon* and *Baran* achieve better performance than other approaches for the goals they aimed to achieve. However, many frameworks reviewed address different aspects of probabilistic data cleaning, such as runtime or specific error types. Due to this, it is difficult to state that certain frameworks are better for all cases. Instead, most developers aim to build on their frameworks to address more aspects of data cleaning instead of specialising in improving performance for one goal. For example, *Baran* works to include error detection for the *Raha* framework rather than improving error detection. Only frameworks that are direct developments over previous versions can be said to be better overall, such as *HoloClean* over *Holistic*. Generally speaking though for frameworks reviewed, those developed more recently are both faster and more accurate than older frameworks.

Finally, a critical review of optimizers used in machine learning was performed. The main finding was that each optimizer has its strengths and limitations. Both theoretical and empirical studies have found applications where all optimizers are applicable. SGD and ASGD are simple and have good generalization properties but can struggle to converge quickly or when minima are complex. Adaptive subgradient methods typically converge faster and more stably than SGD but do not generalize as well and are more com-

putationally expensive. Rprop works separately from these other optimizers and is simple to implement, but is less suited to mini-batching and is limited by not considering the steepness of the loss gradient when making updates. Second-order optimizers such as LBFGS can estimate the Hessian Matrix, allowing for faster convergence, especially in complex loss landscapes. However, this comes at the cost of much higher memory requirements, so first-order methods are typically still favoured in machine learning, especially when scalability is preferred over guarantees about loss minimisation.

Each optimizer mentioned relies on the hyperparameters that determine the degree of effect of their internal mechanics. High momentum can overcome noise but also overshoot minima and high batch size allows for better estimations of the loss gradient but longer runtimes. A high learning rate speeds up convergence but risks adjusting parameters too far. A high weight decay prevents overfitting but can prevent the model from including necessary but complex features.

3 Methodology

3.1 Introduction

The focus of the research undertaken in this study is to select a relevant and accessible framework that uses probabilistic machine learning for the purposes of error repair and to test this framework on a range of benchmark datasets. These tests should compare a range of suitable optimizers, a range of suitable hyperparameter settings and, where possible, a range of framework-specific settings to provide a set of results that give insight into what optimizers, hyperparameters and other settings create optimal results across the datasets tested.

In this methodology, several processes will be described that were used to address the project aims scientifically so other researchers may replicate the results of this investigation. These are:

- The criteria used to select the machine learning-based data cleaning framework for testing.
- The criteria used to select the datasets used during the testing phase along with how these datasets were implemented such that interesting findings were produced. This includes a description of how the performance of the selected framework can be assessed correctly.
- A description and justification of the testing strategy used so it can be assumed with confidence that the trends identified from the results are caused by the designated independent variable for that test.
- A description of the optimizers, hyperparameters and framework-specific settings tested, with justifications as to why investigating those settings would provide relevant findings to the domain of probabilistic data cleaning.

3.2 Framework Selection

The data cleaning framework selected for testing and evaluating optimizers and hyperparameters was required to fulfil certain requirements for the research aims to be addressed sufficiently. Specifically, the framework would have to be:

1. Open-source, frameworks for which either the codebase or an executable version could be found online.

2. Functional. This means that the software should be able to perform its tasks as described in the literature. During the selection process, several frameworks were discarded because either no available code releases could be built and run successfully or because of errors within the software itself. As was the case for one relevant and available framework called *CPClean* (Karlaš et al., 2020), the code lacked the necessary information regarding external package dependencies.

3. Described in an academic setting. The framework would require a description of the machine learning techniques used and the features likely relevant to the testing process. This was to ensure that the background knowledge provided about that framework could be used in helping to understand why certain test results may have been achieved and to give a better context as to how these results may have changed had the tests been different. Section 2.6 covers a range of frameworks that satisfy this criterion.

4. Editable. To test different optimizers, it had to be apparent where these optimizers had been implemented in the codebase these optimizers had been implemented so they and their relevant hyperparameters could be tested.

5. Recent. Ideally, the selected framework would still be in support and use up-to-date machine learning software. Unfortunately, many frameworks did not satisfy the first three selection criteria so this was not possible. Instead, frameworks developed since 2015 were considered as long as the probabilistic error repair methods used were similar to those used in current frameworks.

6. Relevant to the aims. A relevant framework should use a machine learning classifier(s) for probabilistic error detection, probabilistic error repair, or a combination of detection and repair.

7. Fast. Certain frameworks such as *Raha* had unreasonably long runtimes making the extensive testing required for this investigation infeasible.

Given this set of criteria, the framework selected as most suitable for the testing process was *HoloClean*, a more detailed description of which is provided in Section 2.6.4. Originally described in 2017, the source code for this framework was updated considerably in 2019, including use of different framework-specific parameters. Most notably, NaiveBayes has been implemented as the classifier used in the weak-labelling stage instead of Logistic Regression. It was also identified that the only optimizers that had been implemented in the training of the tiedLinearLayer model used to determine

the correct repair value from a domain of candidate values so far were SGD and Adam. *HoloClean* also used the Pytorch package[1] to implement these optimizers, a package that is straightforward to work with. These two factors made it an especially suitable framework to select as it was clear the use of different optimizers had not been tested extensively and that adding new optimizers would be relatively straightforward. This helped to centre the focus of the investigation on the testing rather than the implementation.

3.3 Dataset Selection

The datasets selected for use in testing the selected cleaning approaches also had to fulfil certain criteria, namely:

1. The datasets should be realistic, preferably being a real-world database in which the data represent real entities. Artificial datasets designed to be realistic were also considered.
2. A clean copy of the dataset should be available to verify how well errors were detected and cleaned. In certain cases, only the cleaned dataset was available so errors had to be injected synthetically.
3. The dataset should ideally have been used as a benchmark for other data cleaning frameworks. This means they should also be publicly available and known to researchers so the tests in this investigation can be verified. It also makes it possible for other frameworks to assess objectively if optimal settings discovered for *HoloClean* are the same for their framework.
4. The dataset size should be large enough to provide an accurate gauge of performance and contain enough redundancy for *HoloClean* to achieve meaningful results, but small enough for the runtime and memory requirements not to become a limiting factor during testing.
5. The dataset should vary in characteristics to meet the aim of testing the settings of *HoloClean* on a range of datasets, meaning that the number of entries, number of attributes, data types, error types, error rate and data content should be varied.

Based on these criteria, three databases were selected, a description of each is provided in Table 3 below.

Hospital is a benchmark dataset containing an extract of real data about US medical appointments (Chu et al., 2013; Dallachiesa et al., 2013). Both a

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

Parameter	*Hospital*	*Flights*	*Adult*
Tuples	1000	2376	1100
Attributes	19	6	11
Error Rate	0.03	0.30	0.10
Denial Constraints	15	4	24

Table 3: Datasets selected for use in testing.

clean and dirty dataset existed for this dataset. The dirty dataset had errors injected synthetically by the developers of *HoloClean*. Specifically, typos were introduced by changing a random letter in ∼3% of cells (Rekatsinas et al., 2017).

Flights describes real data about departure and arrival times of flight journeys from different flight-tracking sources (Li, Dong, Lyons, Meng, & Srivastava, 2015). The dirty dataset had a high error rate of ∼30%. It contained typos, missing values and dependency violations. These dependency violations were evident because entries referring to the same flight stated different departure and arrival times.

Because of the limited benchmarks that worked for *HoloClean*, *Adult* was selected. It is not a benchmark dataset but was included in the *HoloClean* codebase and a manual search showed it is an extract of real data describing the demographics and salaries of American adults stemming from US Census data[2] [3] data. No dirty dataset was provided for *Adult* so an error injection method was developed that was similar to what had been done for the *Hospital* dataset[4].

This newly developed error injector introduced typos that caused constraint violations. For each cell, there was a 10% chance the letter "x" would replace a randomly assigned index of the string contained in that cell. 10% may seem like a high degree of errors but given that *Adult* has 11 attributes, there was still plenty of remaining data redundancy. This meant co-occurrence statistics could still be leveraged easily by the *HoloClean* framework.

All three clean datasets can be found on the *HoloClean* Github Page[5]. The dirty datasets created for this project can be found in the supplementary

[2] `https://www.kaggle.com/datasets/ddmasterdon/income-adult`
[3] `http://www.census.gov/ftp/pub/DES/www/welcome.html`
[4] `https://github.com/40508135/HoloClean/blob/master/errorinjection.py`
[5] `https://www.kaggle.com/datasets/ddmasterdon/income-adult`

material and on Github[6]. Extracts of these dirty datasets can also be found in Appendix D.

Each dataset also required a set of denial constraints on which *HoloClean* is reliant to detect errors (Appendix C). For *Hospital* these constraints were included in the *HoloClean* codebase. However, it should be noted the original paper (Rekatsinas et al., 2017) only used 9 denial constraints for *Hopsital* instead of the 15 constraints included in the codebase. Given this investigation aims to assess how effectively repairs are made instead of focusing on error detection, providing the most denial constraints for a given dataset seemed justified. In addition, *HoloClean* was designed with the assumption all relevant and known denial constraints would be provided, so it only makes sense to include all 15. The *HoloClean* framework has also been updated since the original paper was written so direct comparisons to the results in Figure 2 would not be possible anyway. Preliminary testing using the default *HoloClean* settings with the original 9 denial constraints produced different results (Precision = 0.29, Recall = 0.81, $F_1 = 0.45$).

Denial constraints for *Flights* were not included in the code base for *HoloClean* however they were described in the (Mahdavi & Abedjan, 2020) paper, so these were added manually based on that description.

The test data provided within the *HoloClean* codebase provided two denial constraints for *Adult* however these were deemed insufficient in providing enough coverage for the errors that had been injected. Errors had been injected into all 11 attributes but the two denial constraints only linked the attributes "Relationship" and "Sex". This made it impossible for the error detector to identify erroneous cells in other attributes as no constraint violations could occur for those columns. Therefore, based on a manual review of the attributes used in *Adult*, 24 denial constraints were generated. This was a difficult process because *HoloClean* lacked the relevant documentation on entering constraints other than a brief description provided by Rekatsinas et al. (2017), so only a limited number of constraint types were identified as being possible to enter. These were of the form

"if t1.w == t2.w and t1.x != t2.x..."

"if t1.w == t2.w and t1.x == t2.x..."

"if t1.w == y and t1.x == z..."

"if t1.w == y and t1.x != z..."

[6] `https://github.com/40508135/HoloClean`

where t1 and t2 are tuples (rows), w and x are attributes, and y and z are the contents of a cell. This led to the creation of lengthy logical statements to circumvent the limited formats and describe other types of constraints. These 24 denial constraints were added incrementally until there was enough coverage of attributes for reasonable detection and repair performance to be achieved.

Given the manual entry of most denial constraints and that the datasets used for testing are extracts from larger datasets, it cannot be guaranteed that the denial constraints will be correct for all datasets or that these are all the constraints that exist within the datasets. However, the constraints provided gave enough coverage of the erroneous cells that *HoloClean* was adequately able to split the dataset into clean and dirty portions. This meant the learning models used in the repair process were trained to an appropriate standard, as was deduced from performance metrics.

Other benchmark datasets such as *Rayyan* and *Taxes* were implemented (Mahdavi & Abedjan, 2020). However, the error types present in these datasets and the lack of applicable denial constraints meant *HoloClean* could not detect any errors. As a result, these datasets were not included in testing.

3.4 Testing Strategy

Given the aims of this investigation are to test a range of hyperparameters on various optimizers to evaluate their performance for each dataset, this is a similar strategy to conventional hyperparameter tuning strategies outlined in Section 2.7. Conventional strategies involve either automated tuning processes or a manual search. To test different settings such that the impact of one specific hyperparameter could be measured effectively, it became clear only one setting should be varied for each test and all others should be kept constant. One common method for hyperparameter tuning is random search. However, random search would not make it possible to measure the impact of any one hyperparameter as other settings are not kept constant. Instead, it appeared likely that a method similar to grid search where fixed values for each hyperparameter are tested would be more appropriate.

Grid search is very exhaustive and requires significant development to automate testing. It also suffers from the curse of dimensionality meaning even a small grid search of 5 points per setting tested would be infeasible. Specifically, the time complexity of a grid search would be O(D*H*P*F*N) where D is the number of datasets, H is the number of hyperparameters tested, P is

the number of points chosen for each search, F is the number of framework-specific settings tested and N is the number of optimizers used. For this investigation that would lead to $3*6*5*3*9 = 2430$ tests. *HoloClean* takes around 5 minutes to terminate one run on the device used in this investigation which would lead to $2430 * 5/60 = 202.5$ hours of total testing. In light of the time frame dedicated to this project, an exhaustive grid search was ruled out. Instead, a manual search using a smaller number of tests was performed. A different range of values was selected for each setting that was assessed. It was ensured these ranges covered the standard values used for each setting, including the default set by the Pytorch package.

The constants selected to be used during the testing process were based on the default parameters provided by the *Hospital* dataset repair example[7] included in the framework, with only a few minor changes. It was assumed based on these default hyperparameter settings that some tuning had occurred already, making these values appropriate benchmarks. However, the momentum hyperparameter used in SGD and RMSprop was changed from a low value of 0 to a high value of 0.9. It was discovered early on during the testing phase that this improved the results for these optimizers considerably, so it was kept as a new constant for subsequent testing. Second, the learning rate for Adadelta was set to 1.0 because it uses this hyperparameter differently than other optimizers. A value of 1.0 effectively removes the impact of the learning rate, as is the intention for Adadelta. Other hyperparameter tunings were discovered to be optimal in general cases however as this was at a later stage of testing the constants were not adjusted. The hyperparameters used as constants during testing are shown in Table 4. Any deviations from these constants including the independent variable for that test will be described in the relevant sections.

For each test in this investigation, the Precision, Recall and F_1 Score were reported as these were considered the most important performance metrics output by *HoloClean*. Other metrics like accuracy after each epoch of model training was observed during testing are less relevant to results.

3.5 Testing Optimizers

10 different optimizers that update the parameters of the tiedLinearLayer model used for selection of correct repairs in the *HoloClean* framework were selected and implemented during this investigation. These were implemented

[7] `https://github.com/HoloClean/holoclean/blob/master/examples/holoclean_repair_example.py`

Parameter	Setting
domain_thresh_1	0
domain_thresh_2	0
weak_label_thresh	0.99
max_domain	1000
cor_strength	0.6
nb_cor_strength	0.8
epochs	10
weight_decay	0.01
learning_rate	Adadelta = 1.0, Others = 0.001
threads	1
batch_size	1
feature_norm	False
weight_norm	False
momentum	0.9
Seed	45

Table 4: Parameters Kept Constant When Comparing Optimizers.

using Pytorch version 1.6.0. Any hyperparameters used by these optimizers that are not included in the constants mentioned (Table 4) were left as their default settings. These are described on the Pytorch website for each specific version and optimizer[8]. The optimizers selected were:

1. SGD. This optimizer had already been implemented in the *HoloClean* framework so it seemed appropriate to test. The default batch size of 1 also meant SGD was a suitable option, as it was designed for this case. It was the simplest optimizer tested and requires the least computational power. This means if SGD achieved results as high as other, more resource-intensive optimizers, it could be reasoned that SGD is the best choice for this framework.

2. ASGD. This optimizer was tested to check if any poor results generated via the implementation of SGD could be mitigated by reducing any oscillation around the local minima that may have occurred.

3. Adagrad. This optimizer can be used to test if there are any sparse gradients for the weights used by the single-layer model, which would cause comparatively high results for this optimizer.

4. RMSprop. Included to rule out any poor performance results using

[8] `https://pytorch.org/docs/1.6.0/optim.html`

Adagrad being caused by the previous optimizers' tendency to reduce the learning rate too quickly.

5. Adadelta. This optimizer can be robust on noisy gradients and because training data may contain errors because of how *HoloClean* splits the dataset into training and test cases, the performance results of this optimizer may provide insights into this case.

6. Adam. Adam is the default optimizer used by *HoloClean* so it served as a relevant benchmark for test results.

7. Adamax. Adamax has been shown to perform well on many machine learning tasks, therefore it was included in testing. Tests could also show if the Adam optimizer is limited by the stability of convergence that it lacks compared to Adamax.

8. AdamW. AdamW has been shown to generalize better than Adam so this optimizer was tested to check if this factor was the reason for limited performance in the Adam optimizer. The decoupling between learning rate and weight decay for AdamW was also interesting to test because this reduced need to tune multiple combinations of hyperparameters could prove beneficial.

9. Rprop. This optimizer is not designed for mini-batching, however, it was included because batch size was one of the hyperparameters tested so it may outperform other optimizers when larger batch sizes are used. Its resilience to noisy gradients also means that if it outperforms other optimizers, it may be due to this factor.

10. LBFGS. This was to test if using a second-order optimizer led to higher performance results, despite increased computational requirements. If LBFGS outperformed all the other optimizers significantly, it could indicate that the nature of the Hessian Matrix is complex and that convergence with first-order optimizers is difficult. This optimizer required implementing a closure() method to first clear the gradients of each parameter and then compute the loss and return it after each iteration. The edited file with the closure() is included in the supplementary material and is on Github[9].

All 10 optimizers were tested across the 3 datasets, as well as *Flights* with only the first 500 rows and *Flights* with an increased error rate, where errors were injected using the same method as performed for *Adult*. *Adult* was also tested with 200 rows and only 100 rows. These extra tests were performed to assess

[9] `https://github.com/40508135/HoloClean`

how the performance of the optimizers would be affected by either reduced redundancy in the dataset or increased noise in the training data.

3.6 Testing of Hyperparameters

Because an exhaustive test of every combination of hyperparameter settings against every optimizer test was not feasible, each hyperparameter was tested individually with other settings kept constant. This provides less information about the optimal combination of hyperparameters but still provides a lot of insight into the effect any one hyperparameter has on performance. Each hyperparameter was tested on either a range of optimizers, a range of datasets or, as was typically the case, a combination of the two.

3.6.1 Seed

A range of initial seed values were tested to check for any variance in performance on three of the more successful optimizers on *Hospital*, and for Adam on *Adult*. Using the same seed should always lead to the same results, so this test was also used to check if this deterministic property held for *HoloClean*. If it did it meant only one run would have to be performed for each case in subsequent tests.

3.6.2 Epochs

By default, the number of epochs used was 10. A range of epochs between 10 and 50 was used to assess performance of the AdamW optimizer on *Hospital*, to test if increasing the number of epochs increased performance or if convergence had already occurred in the first 10 epochs. If it did then it could be indicate this settings is appropriate and that it may not be the limiting factor causing variance in results.

3.6.3 Batch Size

Batch Size was tested across all three datasets using the AdamW optimizer and Adam and Rprop were also tested on *Hospital* on a range of batch sizes, from 1 through 32. A batch size of 32 is still not close to full-batch optimization, but it was considered likely to be high enough that similar properties would become true. *HoloClean* processes each training case in order and without repetition for each epoch. This means that the number of parameter updates made in each epoch is equal to the number of training cases divided by the batch size. This test could then indicate an ideal trade-

off between updates made and how accurately the gradient of the loss can be estimated.

3.6.4 Momentum

The effect of the momentum value was assessed on *Hospital* using SGD as the optimizer on a range of values between 0 and 0.99. Because only SGD and RMSprop are affected by this hyperparameter only one dataset and optimizer were tested because the results of this experiment are not as insightful as testing hyperparameters used by most optimizers. However, it was still included to assess the default settings and to gain information about the loss gradient. Higher performance for increased momentum values could also bolster any claims about the level of noise in the training inputs.

3.6.5 Learning Rate

The hyperparameter was tested on all three datasets using the Rprop optimizer and was also tested on *Hospital* using AdamW. Values between 0.00001 and 0.1 were used to determine how quickly optimizers could converge without overshooting the minima. Additionally, Adadelta was tested with a range of values between the default of 0.001 and 1.5. While Adadelta doesn't need a learning rate, the Pytorch implementation of this optimizer still allows for tuning where the learning rate acts as a constant on the step size that has been calculated previously. The use of 1.0 means that the learning rate has no additional effect so the testing was to see if this could be improved.

3.6.6 Weight Decay

This hyperparameter was tested with values between 0.00001 and 0.1 using Adam and AdamW on all three datasets, and SGD was tested on *Adult*. This was to find the optimal tuning and assess if the default of 0.01 was too high. By contrast, the default used in Pytorch is 0.001. Based on results from these tests most of the optimizers were reassessed on *Hospital* using a weight decay of 0 to check if conclusions made when testing the optimizers using the default constants still held. The *Flights* dataset containing extra errors that had been injected synthetically was also reassessed with a weight decay of 0 to test if the differences in performance between optimizers shown to exist when assessing *Hospital* with a low weight decay also existed in a context that displayed similar differences between optimizers.

3.7 Testing of Framework-Specific Settings

Besides the optimizer-related parameters, several settings unique to *HoloClean* were also tested. The same constants used for testing the optimizers and hyperparameters were also used for these tests.

The first was the domain pruning threshold. Rekatsinas et al. (2017) showed that values used for this parameter formed a compromise between precision and recall, with higher values typically having higher precision but lower recall (Figure 4). Because the framework has been updated since 2017 the default parameters were assessed across the three datasets to test if similar trends existed. Rprop was also tested on *Flights* and *Adult* to check if the results were the same when tested on more than one optimizer.

The weak label threshold was also tested. Lowering this threshold would make it more likely that a candidate repair value would replace the original value in the featurized dataset. However, if it was too low candidates that are unlikely to be the correct repair would be used to train the tiedLinearLayer model. The effect of this setting using Adam on *Hospital* was tested with a range of values between 0.3 and 0.99. No further testing was performed as the results were pretty conclusive.

Finally, it was identified that both the Logistic Regression and NaiveBayes classifiers had been implemented with *HoloClean* to predict how likely a candidate repair value was correct, based on which weak labels are determined. However, NaiveBayes was set as standard and using the Logistic classifier was not made available as an option to the user. It was therefore deemed an interesting test to compare these classifiers for this aspect of *HoloClean* and was done so using Rprop across the three datasets.

It should be noted that the use of Rprop in many of the tests despite the recommendations against its use in mini-batching is because of the performance it achieved on all three datasets, which is described in the results. One of the aims of the investigation was to find the optimal settings, even if findings are made that do not necessarily conform to general advice provided in the literature.

4 Results

4.1 Comparison of Optimizers

		Adadelta	Adagrad	Adam	AdamW	Adamax	ASGD	LFBGS	RMSprop	Rprop	SGD
Hospital	Precision	0.81	0.02	**1.0**	0.95	**1.0**	**1.0**	0.03	**1.0**	0.96	**1.0**
	Recall	0.19	0.02	0.46	0.8	0.43	0.39	0.07	0.35	**0.83**	0.43
	F_1	0.31	0.02	0.63	0.87	0.6	0.56	0.04	0.52	**0.89**	0.61
Flights	Precision	0.74	0.46	0.73	0.74	**0.78**	0.65	0.43	0.77	**0.78**	0.75
	Recall	0.35	0.22	0.34	0.35	0.37	0.31	0.24	0.36	**0.44**	0.35
	F_1	0.47	0.29	0.47	0.48	0.5	0.42	0.31	0.49	**0.56**	0.48
Adult	Precision	0.81	0.86	0.78	0.62	0.78	0.75	0.06	0.61	0.51	0.78
	Recall	0.19	0.05	0.25	0.31	0.25	0.27	0	**0.31**	**0.31**	0.17
	F_1	0.31	0.1	0.38	0.41	0.42	**0.45**	0.01	0.41	0.39	0.38

Table 5: Comparing Optimisers.

As seen in Table 5, there is a significant disparity in the results achieved between the optimizers used for the custom tiedLinearLayer model, across all three datasets. Adam, Adamax, ASGD and SGD generally achieve the highest precision, up to 100% on *Hospital*. However it was Rprop that achieved the highest recall across all three datasets, and the highest F_1 on *Hospital* and *Flights*. On *Hospital*, the default optimizers of Adam and SGD were outperformed by Rprop and AdamW by $\sim$25% in F_1, a large improvement. In general, many optimizers performed well on both *Flights* and *Adult*. Adagrad and the second-order method of LFBGS achieved very poor results on all three datasets.

From Table 6 it can be seen that there are no large differences between the performance of selected optimizers between the full *Flights* dataset and only the first 500 rows. This is despite there being less data available to train the tiedLinearLayer model. However, Table 7 showed that when extra errors were injected synthetically to the full *Flights* dataset, Rprop had a higher recall and F_1 score than any of the other optimizers, by a factor of 10%.

Table 8 shows that when tested on *Adult* with very few rows, greater disparities between optimizers arise, with RMSprop having the highest F_1 when tested on both 200 and 100 rows. However, the optimizers mentioned to have

	Adam	AdamW	RMSprop	Rprop	SGD
Precision	0.64	0.64	**0.65**	0.57	0.64
Recall	0.35	0.35	0.35	**0.36**	0.34
F_1	0.45	0.45	**0.46**	0.44	0.45

Table 6: Performance of Optimizers on First 500 Rows of *Flights*.

	Adadelta	Adam	AdamW	RMSprop	Rprop	SGD
Precision	0.8	0.81	**0.82**	0.8	**0.82**	0.81
Recall	0.31	0.31	0.34	0.31	**0.41**	0.31
F_1	0.45	0.45	0.48	0.45	**0.55**	0.45

Table 7: Performance of Optimizers on *Flights* with Increased Error Rate.

No. Rows		Adadelta	Adagrad	Adam	Adamax	AdamW	ASGD	RMSprop	Rprop	SGD
200	Precision	0.75	0.42	**0.8**	0.78	0.75	0.83	0.64	0.68	0.78
	Recall	0.11	0.02	0.17	0.15	0.29	0.09	**0.33**	0.31	0.17
	F_1	0.2	0.04	0.27	0.25	0.42	0.17	**0.44**	0.42	0.28
100	Precision	0.89	0.44	0.84	**0.92**	0.78	0.8	0.64	0.81	0.89
	Recall	0.07	0.04	0.14	0.11	0.28	0.07	**0.36**	0.25	0.15
	F_1	0.13	0.07	0.24	0.19	0.41	0.13	**0.46**	0.39	0.26

Table 8: Performance of Optimizers on *Adult* with Reduced Rows.

high precision metrics in *Hospital* retain this same property when tested on this smaller dataset.

4.2 Hyperparameter Results

4.2.1 Seed

	Seed	0	20	45	75	1000	10000
Adam	Precision	**1**	**1**	**1**	**1**	**1**	**1**
	Recall	**0.46**	**0.46**	**0.46**	**0.46**	**0.46**	**0.46**
	F_1	**0.63**	**0.63**	**0.63**	**0.63**	**0.63**	**0.63**
AdamW	Precision	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
	Recall	0.81	0.8	0.81	0.81	0.81	**0.82**
	F_1	0.87	0.87	0.87	0.87	0.87	**0.88**
Rprop	Precision	0.95	**0.96**	**0.96**	**0.96**	0.93	**0.96**
	Recall	0.79	0.78	**0.83**	0.82	0.77	0.76
	F_1	0.87	0.86	**0.89**	**0.89**	0.84	0.85

Table 9: Effect of Seed on *Hospital* Performance.

Shown in Table 9 and Table 10, there was little difference in Performance for the three optimizers tested and no difference for the three datasets that Adam was tested on. However, Rprop appeared to be affected more than adaptive subgradient methods, up to a 7% decrease in recall.

	Seed	0	20	45	75	1000	10000
Hospital	Precision	**1**	**1**	**1**	**1**	**1**	**1**
	Recall	**0.46**	**0.46**	**0.46**	**0.46**	**0.46**	**0.46**
	F_1	**0.63**	**0.63**	**0.63**	**0.63**	**0.63**	**0.63**
Flights	Precision	**0.73**	**0.73**	**0.73**	**0.73**	**0.73**	**0.73**
	Recall	**0.34**	**0.34**	**0.34**	**0.34**	**0.34**	**0.34**
	F_1	**0.47**	**0.47**	**0.47**	**0.47**	**0.47**	0.46
Adult	Precision	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**
	Recall	**0.25**	**0.25**	**0.25**	**0.25**	**0.25**	**0.25**
	F_1	**0.38**	**0.38**	**0.38**	**0.38**	**0.38**	**0.38**

Table 10: Effect of Seed on Performance using Adam.

4.2.2 Epochs

No. of Epochs	10	20	30	40	50
Precision	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
Recall	0.8	**0.81**	**0.82**	**0.82**	**0.82**
F_1	0.87	**0.88**	**0.88**	**0.88**	**0.88**

Table 11: Effect of Number of Epochs on *Hospital* Performance using AdamW Optimizer.

Table 11 shows that for the AdamW optimizer using *Hospital*, increasing the number of epochs higher than the default setting of 10 has little impact on the optimizer's performance. This indicates that convergence has already occurred within 10 epochs.

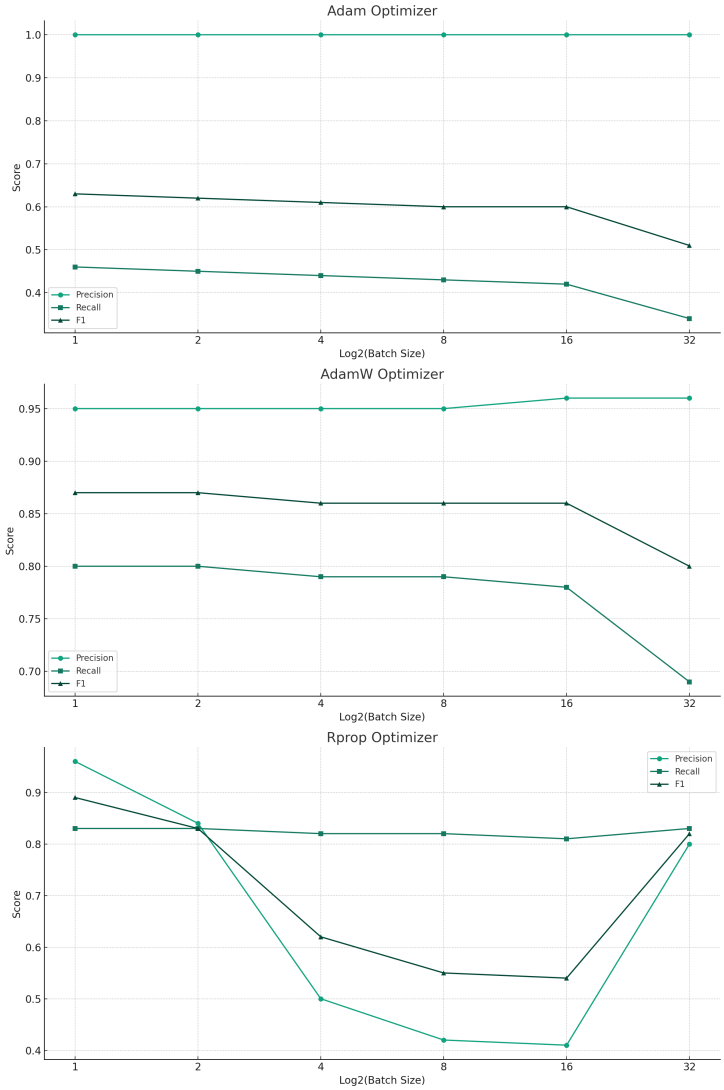

Figure 10: Effect of Batch Size on *Hospital* Performance (Table 24 in Appendix B).

4.2.3 Batch Size

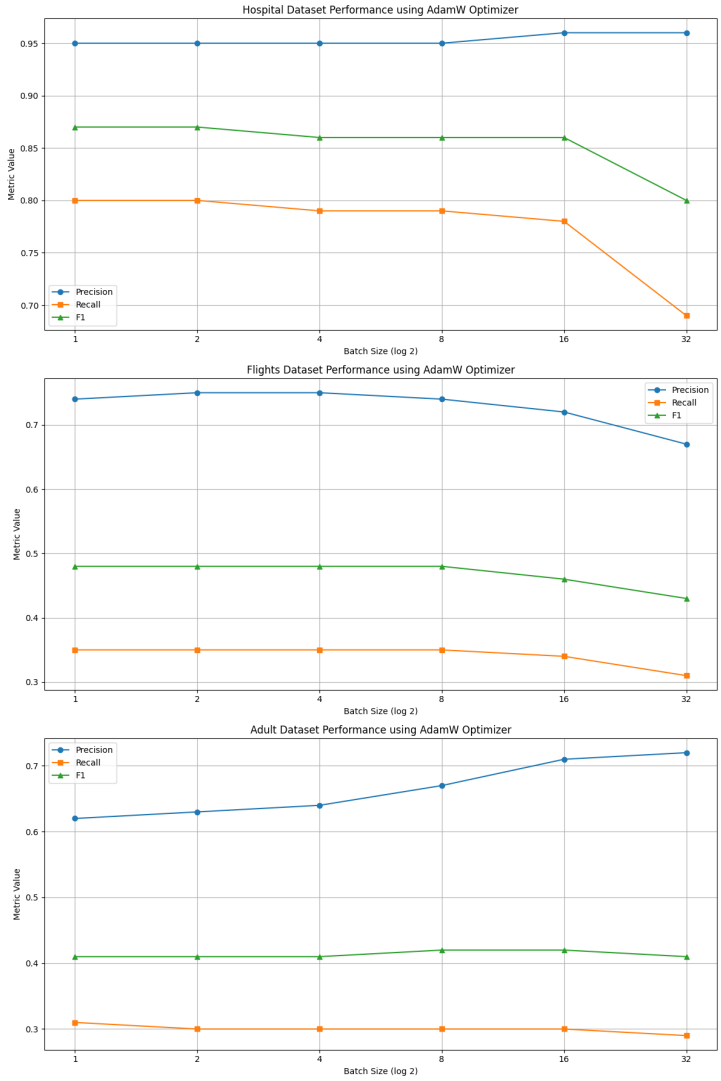

Figure 11: Effect of Batch Size on Performance using AdamW Optimizer (Table 25 in Appendix B).

Figure 10 shows that for the *Hospital* dataset, increasing the batch size decreased the recall and F_1 of the adaptive subgradient methods although precision increased slightly. Figure 10 shows this same trend occurred for AdamW across all three datasets. By comparison, Rprop performance first decreased significantly when batch size was increased before returning to a high level at a batch size of 32, though not as high as a batch size of 1.

4.2.4 Momentum

Momentum	0.99	0.9	0.6	0.2	0.1
Precision	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
Recall	**0.44**	0.43	0.43	0.4	0.4
F_1	**0.61**	**0.61**	0.6	0.58	0.57

Table 12: Impact of Momentum on *Hospital* Performance using SGD as Optimizer.

Testing SGD on *Hospital* showed that momentum made a small difference on recall and F_1. A high value of 0.99 performed the best, much higher than 0 which is both the *HoloClean* and Pytorch default.

4.2.5 Learning Rate

		0.00001	0.0001	0.001	0.01	0.1
Hospital	Precision	0.06	0.93	**0.96**	0.46	0.11
	Recall	0.08	0.36	**0.83**	0.81	0.7
	F_1	0.07	0.52	**0.89**	0.59	0.2
Flights	Precision	0.77	**0.78**	**0.78**	0.77	0.77
	Recall	0.42	**0.44**	**0.44**	**0.44**	**0.44**
	F_1	0.54	**0.56**	**0.56**	**0.56**	**0.56**
Adult	Precision	0.26	**0.71**	0.51	0.56	0.29
	Recall	0.02	0.27	**0.31**	**0.31**	0.23
	F_1	0.03	0.39	0.39	**0.4**	0.26

Table 13: Effect of Learning Rate on Performance using Rprop Optimizer.

	0.00001	0.0001	0.001	0.01	0.1
Precision	0.05	**0.97**	0.95	0.95	0.95
Recall	0.06	0.71	0.8	0.82	**0.85**
F_1	0.05	0.82	0.87	0.88	**0.9**

Table 14: Effect of Learning Rate on *Hospital* Performance using AdamW Optimizer.

When testing Rprop across all three datasets (Table 13) and AdamW on *Hospital* (Table 14), the two optimizers showed a small variation in optimal learning rates. Rprop achieved the highest performance using the default

Learning Rate		0.001	0.5	1.0	1.5
Hospital	Precision	0.02	**0.81**	**0.81**	**0.81**
	Recall	0.04	**0.19**	**0.19**	**0.19**
	F_1	0.03	**0.31**	**0.31**	**0.31**
Flights	Precision	0.44	**0.74**	**0.74**	**0.74**
	Recall	0.24	**0.35**	**0.35**	**0.35**
	F_1	0.31	**0.47**	**0.47**	**0.47**
Adult	Precision	0.35	**0.81**	**0.81**	**0.81**
	Recall	0.01	**0.19**	**0.19**	**0.19**
	F_1	0.01	**0.31**	**0.31**	**0.31**

Table 15: Effect of Learning Rate on Adadelta Optimizer.

learning rate of 0.001. AdamW however, achieved better performance with a very high learning rate of 0.1, notably 5% higher recall than when using the default. Both optimizers displayed poor performance when using a very low learning rate of 0.00001. Precision however was highest for both optimizers when set to lower values of either 0.0001 or 0.001.

Fine-tuning of the learning rate of the Adadelta optimizer (Table 15) showed no difference in performance, other than a large disparity between the *Holo-Clean* default of 0.001 and the recommended value of 1.0.

4.2.6 Weight Decay

	Weight Decay	0.00001	0.0001	0.001	0.01	0.1
Hospital	Precision	0.96	0.96	0.96	**1**	0
	Recall	**0.8**	0.79	0.79	0.46	0
	F_1	**0.87**	**0.87**	**0.87**	0.63	0
Flights	Precision	0.67	0.68	0.68	**0.73**	**0.73**
	Recall	0.32	0.32	0.32	**0.34**	**0.34**
	F_1	0.43	0.44	0.44	0.46	**0.47**
Adult	Precision	0.66	0.69	0.73	**0.78**	0
	Recall	**0.3**	0.29	0.29	0.25	0
	F_1	**0.41**	**0.41**	**0.41**	0.38	0

Table 16: Effect of Weight Decay on Performance using Adam Optimizer.

Table 16 shows that a lower weight decay than the default of 0.01 leads to an increase in recall, and therefore an increase of F_1 by 25% when using Adam

	Weight Decay	0.00001	0.0001	0.001	0.01	0.1
AdamW	Precision	0.61	0.61	0.62	0.62	**0.7**
	Recall	**0.31**	**0.31**	**0.31**	**0.31**	0.29
	F_1	**0.41**	**0.41**	**0.41**	**0.41**	**0.41**
SGD	Precision	0.73	0.73	0.73	**0.78**	0
	Recall	**0.29**	**0.29**	**0.29**	0.25	0
	F_1	0.41	**0.42**	0.41	0.38	0

Table 17: Effect of Weight Decay on *Adult* Performance.

		Adadelta	Adagrad	Adam	Adamax	AdamW	ASGD	RMSprop	Rprop	SGD
Hospital	Precision	0.62	0.04	0.95	0.97	0.95	**0.99**	0.81	0.96	0.96
	Recall	0.31	0.06	0.8	0.68	0.8	0.53	**0.83**	**0.83**	0.78
	F_1	0.41	0.05	0.87	0.8	0.87	0.69	0.82	**0.89**	0.86

Table 18: Comparing Optimizers, Weight Decay = 0

	Adadelta	Adam	AdamW	RMSprop	Rprop	SGD
Precision	0.81	**0.82**	**0.82**	0.8	**0.82**	0.81
Recall	0.33	0.34	0.34	0.37	**0.41**	0.33
F_1	0.47	0.48	0.48	0.5	**0.55**	0.47

Table 19: Performance of Selected Optimizers on *Flights* with Increased Error Rate, Weight Decay = 0.

on *Hospital*, with a smaller improvement seen when tested on *Adult*. This small improvement can be seen when testing SGD on *Adult* (Table 17). On the other hand, a higher weight decay led to higher precision for both Adam and SGD.

Testing of AdamW *Adult* showed that precision increased with weight decay however there was no difference in recall for *Adult*.

Table 18 shows that when the optimizers are compared on *Hospital* again, only with low weight decay, the gap between optimizers observed using the default parameters is reduced. For example, the performance between Adam and AdamW disappears. However, comparing the best-performing optimizers with a weight decay of 0 on *Flights* with extra errors injected synthetically showed the same patterns as when using default settings. Rprop still outperforms the adaptive subgradient optimizers by at least 5% for F_1.

4.3 Results of *HoloClean*-Specific Settings

4.3.1 Domain Pruning Threshold

Figure 13 showed that the highest F_1 was achieved with a low pruning threshold of 0 *Flights* but with higher values on *Hospital* and *Adult*. There did not appear to be a clear trend between this parameter and performance. Results using the Rprop optimizer on *Flights* and *Adult* achieved the best performance with domain pruning values of 0.1 and 0.7 respectively, the same as when using Adam, suggesting the optimal values for a given dataset may be the same across different optimizers. Adam also showed dips in F_1 on *Hospital* and *Adult* at a domain pruning threshold of 0.1, after which performance recovered.

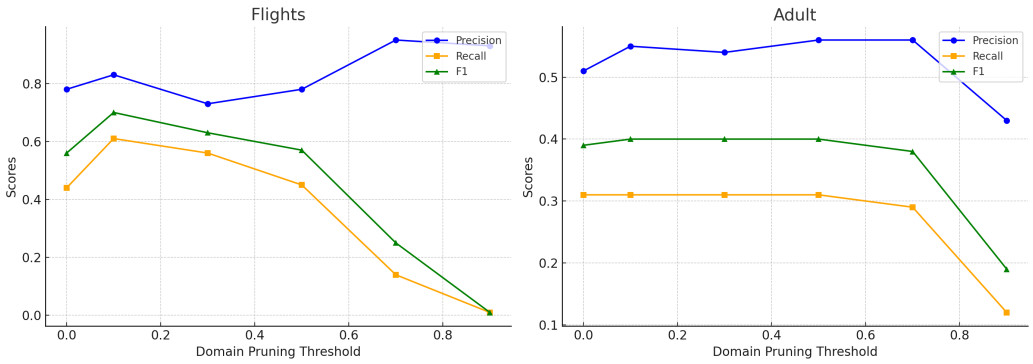

Figure 12: Effect of Domain Pruning Threshold 1 on Performance using Rprop Optimizer (Table 23 in Appendix B).

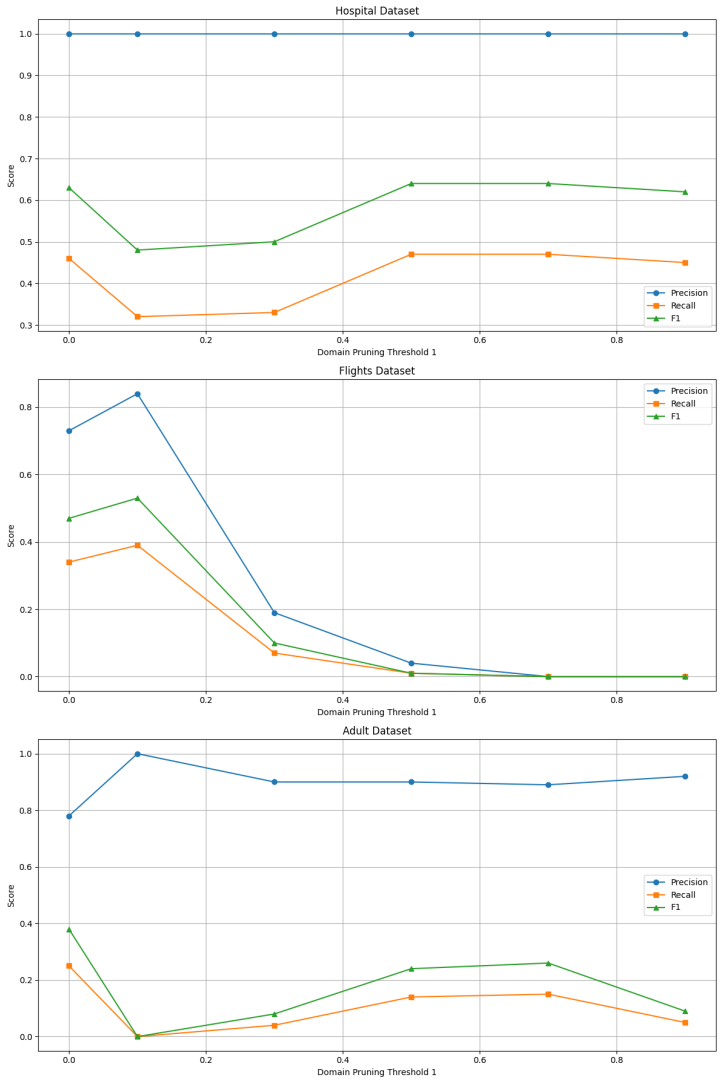

Figure 13: Effect of Domain Pruning Threshold 1 on Performance using Adam Optimizer (Table 22 in Appendix B).

4.3.2 Weak Label Threshold

Testing of this setting on *Hospital* using Adam showed that a high value of at least 0.9 led to the highest performance, in line with the default settings. Reducing this value reduces precision by over 50% while recall is not affected.

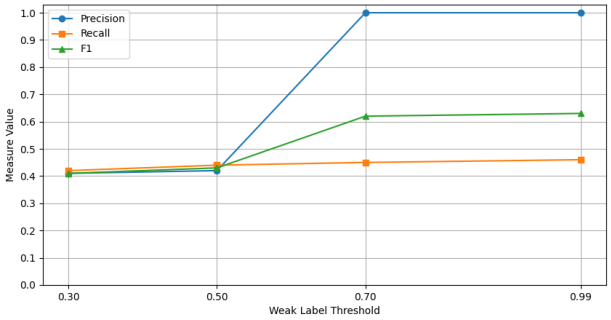

Figure 14: Effect of Weak Label Threshold on *Hospital* Performance using Adam.

		Naive-Bayes	Logistic
Hospital	Precision	**0.96**	**0.96**
	Recall	**0.83**	0.72
	F_1	**0.89**	0.82
Flights	Precision	**0.78**	0.57
	Recall	**0.44**	0.28
	F_1	**0.56**	0.38
Adult	Precision	**0.51**	0.49
	Recall	**0.31**	**0.31**
	F_1	**0.39**	0.38

Table 20: Comparison of Naive-Bayes and Logistic Classifiers using Rprop Optimizer.

4.3.3 Naive-Bayes vs Logistic

Table 20 shows that when using the best performing optimizer, Rprop, using Naive-Bayes for the purposes of generating weak-labels leads to higher recall on *Hospital* and both higher precision and recall on *Flights* than using the Logistic Regression classifier.

5 Discussion

5.1 Discussion of Optimizers

It is clear from the results of this investigation that testing different optimizers on a probabilistic data cleaning framework such as *HoloClean* is vital because, amongst all of the experiments, this had the highest effect on performance. On the low end, Adagrad performed very poorly, perhaps due to this optimizer being prone to reducing the learning rate too quickly, preventing convergence to a better minimum. RMSprop is designed to address this limitation and it performed much more reasonably, suggesting that this is likely the case. LFBGS also led to very poor results, even though theoretically it should have better performance than first-order optimizers. Because it is not reasonable for a second-order to perform worse than first-order optimizers, it was concluded that this optimizer was not implemented correctly or is not compatible with *HoloClean*. However, no errors or reasons for incompatibility could be identified.

SGD outperformed ASGD on all three datasets, which is atypical given how similar the optimizers are. One likely cause for this backward expectation in performance may be that the default Pytorch hyperparameters were inappropriate. ASGD uses a term to dictate when measuring the parameters of the learning model to include in the returned average. If this was set too early then model parameters when the model had not converged closely enough to the minimum could have been included in the final result.

The most notable result was Rprop performing the best despite not being designed for mini-batching. It may be that the landscape of the Hessian Matrix of the loss for a single-layer neural network is complex or different to conventional networks, therefore adaptive subgradient methods are not as effective. It may also be that the method *HoloClean* uses to split data into training and test data led to erroneous entries being used to train the model and therefore noisy gradients, which Rprop may have been able to handle more effectively. The fact that Rprop outperformed other optimizers for *Flights* with an increased error rate backs up this reasoning. It could also potentially be that other hyperparameters were not tuned correctly. Rprop only uses the learning rate meaning it is not affected as much by incorrectly tuned hyperparameters. This last suggestion would imply that when testing other hyperparameters, Rprop would ultimately be outperformed by other optimizers but this was never the case making it unlikely to be the reason. Regardless, its high performance is an interesting finding given it is not recommended for a batch size of 1.

It was also notable that simply by changing the optimizer from Adam to AdamW or Rprop, the framework yielded better results on *Hospital* than even those outlined in the original *HoloClean* paper, by 7% on the F_1. It could be reasoned that the addition of extra denial constraints or more recent updates made to the framework was the cause of these improved results. But this can be ruled out because the repair example on *Hospital* using Adam had worse results than those from 2017. *HoloClean* also switched from Logistic to Naive-Bayes for weak labelling but this can also be ruled out as the cause of performance increases because both classifiers were tested in this investigation. Neither classifier resulted in better performance with the defaults. This makes it very likely this performance improvement was caused simply by changing the optimizer. The performance achieved by the Rprop optimizer is also the same as those produced by the *Baran* framework of 2020, which has a much higher computational overhead, further illustrating the importance of selecting the correct optimizer.

Curiously the results for *Flights* were never as good as those reported by (Rekatsinas et al., 2017) at any stage in the investigation, even with default settings. So it might be that updates made to *HoloClean* decreased performance for this dataset. Identifying the cause of this issue would require an extensive investigation so it was not performed. The dataset is still effective for demonstrating the effect of the settings tested.

5.2 Discussion of Hyperparameters

The impact of hyperparameters was varied. Some showed clear trends, others did not. First, varying the initial seed had no impact on results further highlighting that it is the choice of optimizer having the largest effect on performance. It rules out the idea that any variance noticed is that the initial model parameters benefit one optimizer more than others. The overlap of the same settings and several runs using the same seed also demonstrated the deterministic outcomes of using the same seed. Based on the literature reviewed in Section 2.7, it is likely still that there are seed values that produce outlier performances, but identifying these would be very time-consuming and not particularly informative.

Testing the number of epochs clearly showed that convergence occurred within 10 epochs meaning that for practical purposes it should never have to be increased in a data cleaning framework similar to *HoloClean* where first-order optimizers are used. Observation of the accuracy achieved after each epoch during testing also confirmed that maximum accuracy had already been achieved at epoch 10.

Increasing the batch size was generally worse. This makes sense considering that with the *HoloClean* implementation, increasing the batch size means fewer iterations occur when the number of epochs is fixed. This creates a lower likelihood of the model converging to a minimum within the 10 epochs. Although the sudden spike in performance from Rprop at a high batch size of 32 is interesting. It could just be a coincidence but considering that Rprop is designed to perform full-batch optimization, which a high batch size is closer to, it may be that had higher values been tested this trend would have continued. However, these tests were not performed because the proportional relation between batch size and update iterations would have quickly negated any beneficial properties from further increasing batch size.

The increased performance when increasing the rate of momentum for SGD likely indicates that either the Hessian Matrix of the loss function is indeed quite complex or that the gradients were very noisy. This would align with general research of optimizers and also gives further evidence that this is why Rprop outperformed other optimizers. So a high value of 0.9 is recommended for use.

Results regarding the learning rate were also in line with general theory regarding hyperparameters. It is considered the most important hyperparameter to tune and results in this investigation showed this. The poor performance for a learning rate of 0.00001 was likely due to the optimizers not converging within 10 epochs. Analysis of the accuracy after each epoch explains why AdamW performed better than Rprop at very high learning rates. It showed that the accuracy of Rprop decreased after around 7 epochs, indicating it overshot the minimum loss. This makes sense because AdamW typically makes more conservative updates than Rprop. This means it likely converges more slowly and stably, reducing the risk of overshooting minima. More importantly, the highest precision on both optimizers was achieved when using learning rates lower than 0.001 but higher recall is achieved using learning rates higher than this. A low learning rate reduces the likelihood that the model will make large, incorrect adjustments that take less common features of errors into account, leading to fewer false positives. However, this same principle of not adjusting to take less common features into account means the model will have recall. As a result, the learning rate becomes a pragmatic trade-off between precision and recall based on what the user requires.

Weight Decay showed the inverse of the trend seen with learning rate. The high precision with weight decay can be attributed to a higher penalty being placed on weights with a greater magnitude. Higher weights are often

indicative of greater model complexity, which means more features from the factor graph can be taken into consideration and so the model can fit more closely to the data. This has the effect of candidate repairs that have less typical features still being classified as the correct repair and therefore higher recall. However, the high weight magnitudes also mean features that are less relevant are more likely to be considered, increasing the chances of false positives and therefore lowering precision. It again becomes a hyperparameter for the user to tune based on whether precision or recall is preferred. Typically precision is preferred because the error can still be flagged to the user for manual repair. But if the user prefers to clean data more quickly at the risk of incorrect repairs being made a low weight decay like 0.0001 is recommended.

AdamW not being affected as much by weight decay is to be expected, as by only incorporating the weight decay at the end up of the parameter update process, it places less emphasis on the importance of this hyperparameter, so it is recommended to use this optimizer over Adam as it records similar results but less tuning is required.

Comparing the optimizers on *Hospital* again but with a weight decay of 0 shows that a high default setting of 0.01 is likely what caused a lot of the variance when comparing the optimizers using the constants stated in the methodology. However, the differences in performance on *Flights* with an increased error rate shows that while weight decay plays a significant role, the choice of optimizer is still the most important. Even with low weight decay, Rprop's resilience to noisy gradients allowed it to achieve higher recall and F_1 than Adam-like optimizers when tested with this increased error rate.

5.3 Discussion of Framework-Specific Settings

In Figure 2 in Section 2.6.4, it was stated that a domain pruning threshold of 0.5 was used for *Hospital* and 0.3 was used for *Flights*. 0.5 was still found to be the best value for Adam on *Hospital* but 0.3 performed very poorly on *Flights* for both optimizers, suggesting that the update made to the framework affected the optimal values for this parameter. The results also indicate that while the best domain pruning threshold was the same for both optimizers tested, a lack of a clear optimum across the three datasets means this setting should be tuned on a case-by-case basis. The results also did not match the original explanation that a low pruning threshold led to a higher recall because more candidates are considered. Additionally, for both *Hospital* and *Flights*, there was an uncharacteristic dip in recall at a pruning

threshold of 0.1 using Adam. It is difficult to explain why performance would recover after this dip, although it may be the case that the particular set of values that were accepted into the domain of candidate repairs had a high impact on how the weak-label classifier and tiedLinearLayer model were trained. In any case, this only highlights the necessity to test multiple values of this parameter when cleaning a dataset.

A high weak label threshold leading to a much higher performance is to be expected. The risk that a lower value introduces of adding erroneous weak labels to the featurized dataset that the repair model will then be trained on far outweighs the risk of not including a correct repair as a feature. So the default value of 0.9 is still recommended.

Naive-Bayes outperforming Logistic Regression on all metrics is also an interesting finding. Naive-Bayes assumes independence between features in the given class, leading to a simpler model that is more immune to overfitting (Murphy, 2022). This could mean that the combination of characteristics used to identify the correct repair for weak labelling does not have to be considered strongly, instead it is only what those characteristics are that matter. It might also be that the number of features and attributes is high, which Naive-Bayes typically handles better than Logistic Regression (Bishop, 2006). However, for a multi-class problem such as this, it is hard to determine which classifier will perform better as there are examples of both classifiers performing well in real-world applications (Farid, Zhang, Rahman, Hossain, & Strachan, 2014; Rennie, 2001; Sultana & Jilani, 2018). This indicates that implementing and testing multiple classifiers is a worthwhile investigation for any users of *HoloClean* or similar frameworks.

6 Conclusions

This investigation aimed to determine how the performance of the error repair process used in a machine learning-based data cleaning framework, *HoloClean* in this case, varied across a range of optimizers, hyperparameter settings and framework-specific settings. It was also intended to determine the optimal settings to maximise the performance of *HoloClean*. Relating to theoretical and empirical background knowledge, reasons could then be given about how changes to framework-specific settings impacted the performance. The last aim was to provide broader recommendations based on these findings about the use of optimizers and hyperparameters for similar probabilistic data cleaning frameworks.

Using three benchmark datasets that varied in size, content and the type and frequency of erroneous datasets, 10 optimizers, 6 optimization algorithm hyperparameters and 3 framework-specific settings were tested to address these aims. A well-defined set of constants was used across all tests to ensure that the investigation was scientifically rigorous, hence improving the confidence with which conclusions made about the impact of different settings can be stated. Care was taken to select a relevant framework, including an extensive literature review of existing approaches. Justification for the selection of the optimizers and hyperparameters tested in this investigation was also provided, and literature surrounding the theoretical and empirical background of these settings was reviewed. Using the knowledge gained from existing literature, the results from testing were discussed with comparisons to the literature on data cleaning frameworks as well as references to the expected effects varying optimizers would have.

To answer the aims concerning optimizers, the performance of the error repair process used by *HoloClean* varies most significantly based on which optimizer was selected for use in the tiedLinearLayer model. The optimizer Rprop is recommended because it achieved the highest F_1 across all three datasets tested, especially when the dataset has a high error rate. Using Rprop increased performance comparable to more recent and computationally intensive machine learning-based repair frameworks such as *Baran*. This increase in performance was likely due to the Rprop optimizer being more robust to noise present in the cases used to train the tiedLinearLayer model. Like the AdamW optimizer that also achieved consistently high results, Rprop is also more resistant to how optimizer hyperparameters are tuned, making it easier to implement. However, when data redundancy in the dataset is high and error rate is low, SGD and adaptive subgradient optimizers like Adam

achieved higher precision than Rprop, but lower recall and F_1. Therefore if the dataset is cleaner, it would be recommended to implement an optimizer such as Adam if precision is the priority. The second-order optimizer tested, LBFGS, was outperformed by first-order optimizers even though this is theoretically unlikely, so empirical conclusions about this optimizer have not been made.

Regarding hyperparameter tuning, the initial seed was shown to have little impact on the performance of *HoloClean*, nor did increasing the number of epochs, as for the majority of optimizers tested convergence occurred in the same manner and at the same rate regardless of the initial weights of the single-layer model. So it is recommended both are left at their default values of 45 for seed and 10 for epochs. It would also be recommended that the batch size is kept as low as possible (1) because given how model fitting was implemented, the number of weight updates was proportional to the batch size when the number of epochs is fixed. A lower batch size led to the most updates possible within the optimal 10 epochs, increasing the likelihood of convergence to the minimum loss. However the Rprop optimizer did still show good performance using a high batch size of 32, likely because Rprop is better suited to full batch learning instead of mini-batching. It is recommended that if the momentum hyperparameter is used it should be set to a high value of 0.9, allowing local minima to be overcome, due to a potentially complex loss landscape and to reduce the impact of noise in training data.

Both the hyperparameters of learning rate and the weight decay led to a trade-off. For the majority of optimizers tested, a lower learning rate allowed for increased precision whereas a high learning rate led to better recall and F_1. On the other hand, it was shown that a low weight decay led to higher recall and F_1, and a high weight decay produced higher precision. The explanation for these tradeoffs is that both a high weight decay and a low learning rate reduce the likelihood that large adjustments to parameters would be made to account for training cases that deviated heavily from the norm. This gives less chance that a similarly different repair candidate is identified correctly, reducing the recall. The inverse effect is true for high learning rate and low weight decay. It would be recommended the user tunes these hyperparameters before cleaning datasets to find their preferred balance between recall and F_1, and precision.

Finally, testing on the *HoloClean*-specific settings found three things. The first is that the domain pruning threshold used as a cut-off for weak labelling was optimal at the same point across different optimizers but no clear pattern emerged across the three datasets tested, so it would be advised that a range

of values are tested. The weak label threshold is advised to always be kept at a high value. If this is reduced then incorrect candidate repair values are included in the training data, negatively affecting how the tiedLinearLayer model infers repairs. Regarding the classifiers used in the weak-labelling process, Naive-Bayes was dominant in performance over Logistic Regression, likely because its simplicity was better suited to the high dimensionality of the input data. However, background reading showed that for a multi-class problem such as this one, it is difficult to determine which classifier will perform best. For *HoloClean* Naive-Bayes is recommended, but for developers of similar data cleaning frameworks, it is recommended multiple classifiers are tested.

These findings provide a reasonable contribution to the field of probabilistic data cleaning by showing based on empirical information that the effect of optimizers and hyperparameters used for *HoloClean* is largely in line with their theoretical expectations. Suggestions are also given for any deviations from the expected outcomes. With these insights, users of *HoloClean* should be able to tune specific settings to achieve ideal data cleaning, and developers of similar software can use the values performed in these tests to achieve better performance for themselves. It is unlikely that the exact values for hyperparameter tuning will be the same for similar frameworks but trends and trade-offs for optimizers and hyperparameters have been confirmed present for a relevant machine learning-based data cleaning framework. With this knowledge, researchers in the field can select optimizers and tune hyperparameters based on empirically-backed recommendations, reducing test time required, and with an increased understanding of how any improvements may be made.

6.1 Limitations and Suggestions for Future Work

There were several key limitations to the methodology used to address the aims of this investigation, from which suggestions for future work can be made.

As mentioned, for empirical data relating to the impact of specific hyperparameters in a given context to be investigated rigorously, a grid search is the best approach. As also stated this was not possible due to the given time frame and computational resources available during this investigation. In future, it would be recommended to perform a grid search for both the purposes of replicating and confirming the results of this investigation, as well as checking a wider search space to assess if any other key findings relating to hyperparameters used on *HoloClean* had been missed.

A second limitation of this study was the framework that was tested. *HoloClean* was last updated in 2019 making it comparatively old, limiting the relevance of findings made as more recent machine learning-based repair frameworks likely use improved or different techniques for error repair. For example the *Horizon* framework also uses a factor graph that captures features of the data to infer correct repairs (Rezig et al., 2021) but because the tiedLinearLayer model tested in this investigation is specific to *HoloClean*, it cannot be guaranteed that these findings carry over to a framework like *Horizon*. This limitation was caused by the low number of open-source data cleaning frameworks available, so if a similar and more up-to-date framework becomes open-source and is described in an academic setting, then it would be ideal to compare and evaluate the user of optimizers on that framework.

The investigation was also limited by the number of benchmark datasets that produced results on *HoloClean*. As described, benchmarks such as *Rayyan* and *Taxes* were tested, but the nature of the errors meant they could not be detected. In addition, the error injection method used during the investigation was fairly simple, only creating one-letter typos. Future work could include testing the framework on a wider variety of datasets, including those where it is easier to identify the denial constraints required for the error detection phase and those where there are more types or errors. This would likely include the introduction of datasets that have not been used as benchmarks previously.

Then there is the case of the optimizers selected and implemented. It was clear that the second-order LBFGS optimizer may not have been implemented correctly so it would be suggested that this is examined in future to assess its performance and feasibility given its extended runtime. There were also other applicable optimizers such as Nadam (Dozat, 2016) and Radam (L. Liu et al., 2019) that were not implemented because the latest version of Pytorch could not be used due to package conflicts, and manual implementation was outwith the scope of the project. Future testing of optimizers on machine learning-based cleaning frameworks should also consider these more recent optimizers where applicable.

This investigation also did not lead to any empirical findings about the runtime and scalability of optimizers in an objective manner, which would be recommended, given that these are crucial factors for cleaning larger datasets, especially now that the cleaning of Big Data is becoming a relevant research topic (Tang, 2014).

It's also noted that while the recommendations made are useful, the extent of the investigation was not big enough to provide concrete test suggestions

that would eliminate the necessity for either a random search or grid search for similar frameworks. It is not clear if this is possible, but if theoretical and empirical studies concerning optimizers continue to be published, a meta-analysis may be able to identify trends that are consistent enough for important hyperparameters to require far fewer tests.

Future work could also involve the assessment of optimizers involved in classifiers used for error detection. This investigation only tested models involved in error repair. If a combined set of recommendations can be made about how to tune settings for both the error detection phase and the error repair phase, it could reduce testing required for end-to-end data cleaning applications, like the pairing of *Raha* and *Baran*.

6.2 Self Appraisal

The honours project process was a great learning experience but not without its hurdles. Both strengths and limitations became apparent as the project progressed.

First, there is the subject of the original project goals. The aims set out in the initial report in September were to compare and evaluate different machine learning-based data cleaning frameworks and, based on this learning, extend an existing framework to generate a novel data cleaning approach to address the limitations identified when evaluating existing work. Evidently, this is not what the project has ended up being centred around. Firstly, the feedback from the initial report outline was that the scope was too large. Additionally, when the literature review commenced it became clear that the learning required to develop a novel machine learning-based data cleaning application was unreasonable given the scope of an honours project. For example, even just creating a featurized dataset to train a classifier is a difficult task in itself. One of the strengths I displayed at the start of this project was being able to take the advice from my supervisor and second marker into account and make changes. Based on this advice I reduced the scope to testing and evaluating existing frameworks instead, with extending or improving an existing framework being left as optional. This also showed the resourcefulness to be able to change the research aims if necessary to still produce interesting findings in the face of steep learning curves.

Unfortunately, in March, it became clear that it would be difficult to continue with the goal of testing and evaluating frameworks because of how few of them could be implemented and from which results could be reproduced, three to be precise. I was also unable to determine how to test these

frameworks on new datasets, a vital component. This highlights some of the weaknesses displayed in the project process, which were time management and not considering the technical development phase earlier. Had I completed the literature review earlier and had I looked into which of the frameworks I was reading about when writing the literature review could actually be compared and evaluated, this obstacle would have been spotted much sooner so mitigating it would have been easier.

Fortunately, the same strength of being able to adjust the research of the project when necessary was shown. I had spent a lot of time testing various parameters of the *HoloClean* framework to try and understand how it worked, largely in a vain attempt to reproduce the results from the original 2017 paper. It was in doing so that I noticed the optimizer used and decided, on a whim, to change it from Adam to AdamW, which improved the results considerably. So I decided to focus the testing on optimizers and hyperparameters as I realised this would still make for an interesting investigation.

Changing the focus as late as March was not easy. While testing optimizers and hyperparameters is quite a straightforward process in the sense that implementation does not take long, there was a lot of learning required to be able to explain these results. At the start of March, I would not have been able to explain the role of optimizers in machine learning properly, let alone the internal mechanics of these optimizers. However, one of the other strengths I have displayed in this project is to dedicate time to learning topics I have had little experience in previously. Within two months I worked to learn how the majority of popular optimizers and hyperparameters worked, including learning about the knowledge gained by observing the first and second derivatives of the loss, and what information can be gained from analysing the Hessian Matrix. Given I had already spent a lot of time reading technical papers on data cleaning, dedication to learning new concepts to be able to address the research goal was a clear strength.

However, working on a project where a lot of the concepts were very new was also a limitation. Spotting that the *HoloClean* framework could be improved by changing the optimizer was actually quite lucky, had I been working with software that I was more familiar with then it would have probably been much easier to spot avenues for improvement. So I think more planning before settling on a topic instead of investigating a domain of computer science that I was very unfamiliar with would have helped most of the obstacles that had to be dealt with.

Generally speaking, as the progress diaries show, when I was managing time

effectively, I communicated with my supervisor regularly, so that ideas could be discussed. It was a real advantage to have meetings because it meant the state of the project was reviewed regularly, making it clearer what work still needed to be done. This was one of the key factors in the project being a success despite the setbacks. Although it is worth mentioning that, as can be seen from the progress diaries, deadlines relating to when different parts of the project should have been completed were not adhered to as strictly as they should have been, nor was the Gantt chart used to outline the project timeline updated often enough. For example, had the deadline of the literature review being finished earlier been adhered to, more time would have been available to plan for the development phase. This extra planning time may have allowed for discoveries to be made which meant some of the project's original aims could have been met.

Overall, I would say that the project culminated in a successful investigation. Better findings could have been made had more time been dedicated to the project in the earlier months and if planning to help mitigate predictable obstacles like the lack of available frameworks to test had been introduced earlier. But despite these issues, findings useful to researchers in the domain of data cleaning were still made because of dedication to the project, resourcefulness, and not being afraid to spend the time to learn about new aspects of computing.

References

Berahas, A. S., Nocedal, J., & Takác, M. (2016). A multi-batch l-bfgs method for machine learning. *Advances in Neural Information Processing Systems*, *29*.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, *24*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Bishop, C. M. (2006). Pattern recognition and machine learning. *Springer google schola*, *2*, 645–678.

Bishop, C. M., & Nabney, I. (2008). *Pattern recognition and machine learning: A matlab companion.* Springer.

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4) (No. 4). Springer.

Bollapragada, R., Nocedal, J., Mudigere, D., Shi, H.-J., & Tang, P. T. P. (2018). A progressive batching l-bfgs method for machine learning. In *International conference on machine learning* (pp. 620–629).

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of compstat'2010: 19th international conference on computational statisticsparis france, august 22-27, 2010 keynote, invited and contributed papers* (pp. 177–186).

Chen, J., Zhou, D., Tang, Y., Yang, Z., Cao, Y., & Gu, Q. (2018). Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint arXiv:1806.06763*.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*.

Chu, X., Ilyas, I. F., Krishnan, S., & Wang, J. (2016). *Data cleaning: Overview and emerging challenges.* Retrieved from `http://dx.doi.org/10.1145/2882903.2912574` doi: 10.1145/2882903.2912574

Chu, X., Ilyas, I. F., & Papotti, P. (2013). Holistic data cleaning: Putting violations into context. In (p. 458-469). doi: 10.1109/ICDE.2013.6544847

Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., & Ye, Y. (2015). Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 acm sigmod international conference on management of data* (pp. 1247–1261).

Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouz-

zani, M., & Tang, N. (2013). Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 acm sigmod international conference on management of data* (pp. 541–552).

Demaris, A. (1992). *Logit modeling: Practical applications* (No. 86). Sage.

Dozat, T. (2016). *Incorporating nesterov momentum into adam.*

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

Farid, D. M., Zhang, L., Rahman, C. M., Hossain, M. A., & Strachan, R. (2014). Hybrid decision tree and naïve bayes classifiers for multi-class classification tasks. *Expert systems with applications*, *41*(4), 1937–1946.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*.

Igel, C., & Hüsken, M. (2003). Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, *50*, 105–123.

Igel, C., & Hüsken, M. (2000, 01). Improving the rprop learning algorithm..

Ilyas, I., Aref, W., & Elmagarmid, A. (2004, 09). Supporting top-k join queries in relational databases. *The VLDB Journal*, *13*. doi: 10.1007/s00778-004-0128-2

Ilyas, I. F., & Chu, X. (2019). *Data cleaning*. Association for Computing Machinery.

Ilyas, I. F., & Rekatsinas, T. (2022, 7). Machine learning and data cleaning: Which serves the other? *ACM Journal of Data and Information Quality (JDIQ)*, *14*. Retrieved from `https://dl-acm-org.napier.idm.oclc.org/doi/10.1145/3506712` doi: 10.1145/3506712

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Karlaš, B., Li, P., Wu, R., Gürel, N. M., Chu, X., Wu, W., & Zhang, C. (2020). Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117*.

Kelleher, J. D., Mac Namee, B., & D'arcy, A. (2020). *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press.

Keskar, N. S., & Socher, R. (2017). Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kochenderfer, M. J., & Wheeler, T. A. (2019). *Algorithms for optimization*. Mit Press.

Li, X., Dong, X. L., Lyons, K., Meng, W., & Srivastava, D. (2015). Truth finding on the deep web: Is the problem solved? *arXiv preprint arXiv:1503.00303*.

Liu, D. C., & Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, *45*(1), 503–528.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Mahdavi, M., & Abedjan, Z. (2020, 7). Baran. *Proceedings of the VLDB Endowment*, *13*, 1948-1961. Retrieved from `https://dl.acm.org/doi/10.14778/3407790.3407801` doi: 10.14778/3407790.3407801

Mahdavi, M., & Abedjan, Z. (2021). Semi-supervised data cleaning with raha and baran. In *Cidr*.

Mahdavi, M., Abedjan, Z., Castro Fernandez, R., Madden, S., Ouzzani, M., Stonebraker, M., & Tang, N. (2019). Raha: A configuration-free error detection system. In *Proceedings of the international conference on management of data (sigmod)* (pp. 865–882).

Mitchell, T. M. T. M. (1997). *Machine learning*. McGraw Hill. (Includes bibliographical references and index.)

Mokhtari, A., & Ribeiro, A. (2015). Global convergence of online limited memory bfgs. *The Journal of Machine Learning Research*, *16*(1), 3151–3181.

Mueller, H., & Freytag, J.-C. (2005). Problems , methods , and challenges in comprehensive data cleansing.. Retrieved from `https://api.semanticscholar.org/CorpusID:15756458`

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Murphy, K. P. (2022). *Probabilistic machine learning: an introduction*. MIT press.

Norouzi, S., & Ebrahimi, M. (2019). *A survey on proposed methods to address adam optimizer deficiencies.*

Picard, D. (2021). Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, *4*(5), 1–17.

Polyak, B. T., & Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, *30*(4), 838–855.

Reddi, S. J., Kale, S., & Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Rekatsinas, T., Chu, X., Ilyas, I. F., & Ré, C. (2017, aug). Holoclean: holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, *10*(11), 1190–1201. Retrieved from `https://doi.org/10.14778/3137628.3137631` doi: 10.14778/3137628.3137631

Rennie, J. D. (2001). *Improving multi-class text classification with naive bayes.*

Rezig, E. K., Ouzzani, M., Aref, W. G., Elmagarmid, A. K., Mahmood, A. R., & Stonebraker, M. (2021, 7). Horizon: scalable dependency-driven data cleaning. *Proceedings of the VLDB Endowment*, *14*, 2546-2554. Retrieved from `https://dl.acm.org/doi/10.14778/3476249.3476301` doi: 10.14778/3476249.3476301

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Ieee international conference on neural networks* (pp. 586–591).

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Shi, N., & Li, D. (2021). Rmsprop converges with proper hyperparameter. In *International conference on learning representation.*

Shin, J., Wu, S., Wang, F., Sa, C. D., Zhang, C., & Ré, C. (2015). Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *8*, 1310. Retrieved from `/pmc/articles/PMC4852149//pmc/articles/PMC4852149/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC4852149/` doi: 10.14778/2809974.2809991

Stock, S., Pohlmann, S., Günter, F. J., Hille, L., Hagemeister, J., & Reinhart, G. (2022). Early quality classification and prediction of battery cycle life in production using machine learning. *Journal of Energy Storage*, *50*, 104144.

Sultana, J., & Jilani, A. K. (2018). Predicting breast cancer using logistic regression and multi-class classifiers. *International Journal of Engineering & Technology*, *7*(4.20), 22–26.

Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE transactions on*

cybernetics, *50*(8), 3668–3681.

Tang, N. (2014). Big data cleaning. In *Asia-pacific web conference* (pp. 13–24).

Thapa, S., Zhao, Z., Li, B., Lu, L., Fu, D., Shi, X., ... Qi, H. (2020). Snowmelt-driven streamflow prediction using machine learning techniques (lstm, narx, gpr, and svr). *Water*, *12*(6), 1734.

Tieleman, T. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26.

Wright, S. J. (2006). *Numerical optimization.*

Yakout, M., Berti-Équille, L., & Elmagarmid, A. K. (2013). Don't be scared: Use scalable automatic repairing with maximal likelihood and bounded changes. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 553-564. Retrieved from `https://dl.acm.org/doi/10.1145/2463676.2463706` doi: 10.1145/2463676.2463706

Yi, D., Ahn, J., & Ji, S. (2020). An effective optimization method for machine learning based on adam. *Applied Sciences*, *10*(3), 1073.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhou, P., Feng, J., Ma, C., Xiong, C., Hoi, S. C. H., & E, W. (2020). Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, *33*, 21285–21296.

Zhou, P., Xie, X., Lin, Z., & Yan, S. (2024). Towards understanding convergence and generalization of adamw. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zou, F., Shen, L., Jie, Z., Zhang, W., & Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 11127–11135).

Appendices

A Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Investigation into Machine Learning-Based Data Cleaning Approaches

Overview of Project Content and Milestones

Data cleaning is the process of detecting and correcting corrupt or inaccurate data within a data set and is vital if any meaningful analysis is to be drawn from the trends that occur within that data set. Conventional data cleaning is typically an expensive process that requires a large degree of input from human users, each of whom must be well-informed about the qualitative and quantitative constraints imposed on the data set in question. In recent years, however, techniques have been developed that utilise Machine Learning, a subset of Artificial Intelligence (AI), to help streamline many aspects of this data cleaning process. This subset of machine learning-based data cleaning is what this project will focus on. Specifically, the major milestones of the project are:

1. To research the machine learning-based data cleaning approaches that are currently available and have been described in an academic setting, and to draw a comprehensive set of conclusions from this research. These conclusions should outline the strengths and limitations of these data cleaning approaches and describe how the nature of the machine learning techniques involved will affect the performance of these cleaning processes when applied to data sets that vary in different ways. This variance may include factors such as data set size, numerical vs categorical data, or reduced data quality. This milestone is expected to take up the first two to three months of the project.

2. Based on this research, propose an improved method of machine learning-based data cleaning to address one or several of the limitations that have been identified within existing data cleaning approaches. Justifications should be provided as to where and how this novel method would outperform current software. This milestone is expected to take around a month.

3. Create a software prototype that implements this newly proposed machine learning-based data cleaning approach. This stage is expected to last from January to March approximately, because it will include the length process of training the machine learning model in question.

4. Create a testbed to compare the prototype against any available machine learning-based data cleaning software. Analysis of the test's results would help evaluate how the prototype performed, if it successfully addressed the limitation it was designed to address, and what this indicates about the machine-learning method used. This milestone is predicted to take roughly one month, or the remainder of the project's time.

The Main Deliverable(s):

The main deliverable will be the software that implements the newly proposed machine learning-based data cleaning technique, complete with documentation for its features. Given the scope of the project, this deliverable should be a proof-of-concept prototype, not a finalised product.

The Target Audience for the Deliverable(s):

- Organisations that handle data sets large enough for human input to become a limiting factor for data cleaning.

- Data analysts looking for cheaper data cleaning methods, especially those with limited access to human resources.

- Data scientists and other researchers with a special interest in machine learning-based data cleaning because no doubt the prototype could be improved further.

The Work to be Undertaken:

1. Collect all the relevant research papers and data pertaining to existing machine learning-based data cleaning software.

2. Analyse the information that has been gathered and from these analyses, evaluate the strengths and limitations of each data cleaning method.

3. Design a novel machine learning-based data cleaning method and document the features of this new design.

4. Implement the new design using appropriate software, and, where necessary, train the machine learning component of this design on accurate data.

5. Select or design a testbed for testing the new data cleaning software and select data to test the prototype on. This will likely include "dirtying" existing data sets to a known degree, so it can be assessed as to how well the prototype cleaned that dirtied data.

6. Test the new data cleaning software against similar software on the market.

7. Analyse the data gained from testing the new software and evaluate how well it has addressed the limitation of existing software that it was designed to address.

Additional Information / Knowledge Required:

A greater understanding of how machine learning algorithms work will be needed to be able to think critically about how each data cleaning approach may behave given suboptimal conditions. This includes a better understanding of the maths behind these techniques, because it's the formulas that are referenced in the research papers that often reveal the most about that approach's capabilities. This is also the same understanding I will need to have if I'm to make any useful suggestions as to where these approaches could be improved.

Most of the extra knowledge required will be for developing the data cleaning prototype. I will have to learn how to train machine learning models correctly so that the software performs as expected, an area around which my knowledge is still limited. The testing of this prototype will then also require me to learn where to find opensource test data, and to correctly select data that is most likely to provide informative results about the prototype.

The testing process will also mean I will need to find out how to cost-effectively test data cleaning algorithms, whether that's using cloud computing software or simply reducing the test data down to a computationally affordable size.

Information Sources that Provide a Context for the Project:

Rekatsinas, T., Chu, X., Ilyas, I. F., & Ré, C. (2017). *HoloClean: Holistic Data Repairs with Probabilistic Inference*. http://www.vldb.org/pvldb/vol10/p1190-rekatsinas.pdf

Berti-Equille, L. (n.d.). *Machine Learning-Based Data Cleaning : Current Solutions and Challenges*. Retrieved September 22, 2023, from http://pageperso.lif.univ-mrs.fr/~laure.berti/https://diams.lis-lab.fr/

Chu, X., Ilyas, I. F., Krishnan, S., & Wang, J. (2016). *Data Cleaning: Overview and Emerging Challenges*. https://doi.org/10.1145/2882903.2912574

Yakout, M., Berti-Équille, L., & Elmagarmid, A. K. (2013). Don't be SCAREd: Use SCalable Automatic REpairing with maximal likelihood and bounded changes. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 553–564. https://doi.org/10.1145/2463676.2463706

The Importance of the Project:

The amount of data stored by corporate entities is increasing exponentially and if any meaningful business intelligence insights are to be made then that data should be cleaned to within a reasonable degree. Conventional methods do not scale well when the volume of data increases, nor are they able to cope with increased data variety. With the advent of Big Data, data sets that are too large or complex to be dealt with by traditional processing software, AI-based cleaning methods become vital if for keeping the speed and cost of data cleaning to an acceptable standard.

As this field of data science is still in its infancy, it requires more research and development into novel techniques that use machine learning and other branches of AI to address the problem of dirty data. This is why this project also aims to propose an improved data cleaning approach, rather than just evaluating software that is currently available.

The Key Challenge(s) to be Overcome:

The research papers that describe existing machine learning-based data cleaning techniques are highly technical, so a lot of time will have to be spent studying those papers, to make sure all relevant information has been extracted and that it this information is understood correctly, else the conclusions made about those cleaning processes will be incorrect. The proposal of a new data cleaning method will also lean heavily on the information gained in the research phase, highlighting again the importance that is care is taken to understand the machine learning techniques used fully.

For the implementation stage, time will have to be taken to learn how this implementation should take place: what kind of software to use, how to correctly train the machine learning model, and how to construct an appropriate testbed to effectively evaluate this new technique. It may be the case that the testing stage will not be cheap, so this risk should be assessed for before the implementation phase commences so any cost or time constraints can be addressed in advance. This process will take most of the project's time, so it's imperative that I manage my time effectively during this stage if all the project's aims are to be met by April.

B Results

Weak label threshold	0.99	0.7	0.5	0.3
Precision	**1.0**	**1.0**	0.42	0.41
Recall	**0.46**	0.45	0.44	0.42
F_1	**0.63**	0.62	0.43	0.41

Table 21: Impact of Weak Label Threshold on *Hospital* Performance Using Adam as Optimizer.

		0	0.1	0.3	0.5	0.7	0.9
Hospital	Precision	**1**	**1**	**1**	**1**	**1**	**1**
	Recall	0.46	0.32	0.33	**0.47**	**0.47**	0.45
	F_1	0.63	0.48	0.5	**0.64**	**0.64**	0.62
Flights	Precision	0.73	**0.84**	0.19	0.04	0	0
	Recall	0.34	**0.39**	0.07	0.01	0	0
	F_1	0.47	**0.53**	0.1	0.01	0	0
Adult	Precision	0.78	**1**	0.9	0.9	0.89	0.92
	Recall	**0.25**	0	0.04	0.14	0.15	0.05
	F_1	**0.38**	0	0.08	0.24	0.26	0.09

Table 22: Effect of Domain Pruning Threshold 1 on Performance using Adam Optimizer.

	Domain Threshold	0	0.1	0.3	0.5	0.7	0.9
Flights	Precision	0.78	0.83	0.73	0.78	**0.95**	0.93
	Recall	0.44	**0.61**	0.56	0.45	0.14	0.01
	F_1	0.56	**0.7**	0.63	0.57	0.25	0.01
Adult	Precision	0.51	0.55	0.54	**0.56**	**0.56**	0.43
	Recall	**0.31**	**0.31**	**0.31**	**0.31**	0.29	0.12
	F_1	0.39	**0.4**	**0.4**	**0.4**	0.38	0.19

Table 23: Effect of Domain Pruning Threshold 1 using Rprop Optimizer.

	Batch Size	1	2	4	8	16	32
Adam	Precision	**1**	**1**	**1**	**1**	**1**	**1**
	Recall	**0.46**	0.45	0.44	0.43	0.42	0.34
	F_1	**0.63**	0.62	0.61	0.6	0.6	0.51
AdamW	Precision	0.95	0.95	0.95	0.95	**0.96**	**0.96**
	Recall	**0.8**	**0.8**	0.79	0.79	0.78	0.69
	F_1	**0.87**	0.87	0.86	0.86	0.86	0.8
Rprop	Precision	**0.96**	0.84	0.5	0.42	0.41	0.8
	Recall	**0.83**	**0.83**	0.82	0.82	0.81	**0.83**
	F_1	**0.89**	0.83	0.62	0.55	0.54	0.82

Table 24: Effect of Batch Size on *Hospital* Performance.

	Batch Size	1	2	4	8	16	32
Hospital	Precision	0.95	0.95	0.95	0.95	**0.96**	**0.96**
	Recall	**0.8**	**0.8**	0.79	0.79	0.78	0.69
	F_1	**0.87**	**0.87**	0.86	0.86	0.86	0.8
Flights	Precision	0.74	**0.75**	**0.75**	0.74	0.72	0.67
	Recall	**0.35**	**0.35**	**0.35**	0.35	0.34	0.31
	F_1	**0.48**	**0.48**	**0.48**	**0.48**	0.46	0.43
Adult	Precision	0.62	0.63	0.64	0.67	0.71	**0.72**
	Recall	**0.31**	0.3	0.3	0.3	0.3	0.29
	F_1	0.41	0.41	0.41	0.41	**0.42**	0.41

Table 25: Effect of Batch Size on Performance using AdamW Optimizer.

	Weight Decay	0.00001	0.0001	0.001	0.01	0.1
Hospital	Precision	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
	Recall	**0.8**	**0.8**	**0.8**	**0.8**	0.73
	F_1	**0.87**	**0.87**	**0.87**	**0.87**	0.82
Flights	Precision	**0.74**	**0.74**	**0.74**	**0.74**	**0.74**
	Recall	**0.35**	**0.35**	**0.35**	**0.35**	**0.35**
	F_1	**0.48**	**0.48**	**0.48**	**0.48**	0.47
Adult	Precision	0.61	0.61	0.62	0.62	**0.7**
	Recall	**0.31**	**0.31**	**0.31**	**0.31**	0.29
	F_1	**0.41**	**0.41**	**0.41**	**0.41**	**0.41**

Table 26: Effect of Weight Decay on Performance using AdamW Optimizer.

C Denial Constraints

Listing 1: Denial Constraints for *Hospital*

```
t1&t2&EQ(t1.Condition,t2.Condition)&EQ(t1.MeasureName,
    t2.MeasureName)&IQ(t1.HospitalType,t2.HospitalType)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&IQ(t1.ZipCode
    ,t2.ZipCode)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&IQ(t1.
    PhoneNumber,t2.PhoneNumber)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&IQ(t1.
    MeasureName,t2.MeasureName)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&IQ(t1.Stateavg,
    t2.Stateavg)
t1&t2&EQ(t1.ProviderNumber,t2.ProviderNumber)&IQ(t1.
    HospitalName,t2.HospitalName)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&IQ(t1.Condition
    ,t2.Condition)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&IQ(t1.
    Address1,t2.Address1)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&IQ(t1.
    HospitalOwner,t2.HospitalOwner)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&IQ(t1.
    ProviderNumber,t2.ProviderNumber)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&EQ(t1.
    PhoneNumber,t2.PhoneNumber)&EQ(t1.HospitalOwner,t2.
    HospitalOwner)&IQ(t1.State,t2.State)
t1&t2&EQ(t1.City,t2.City)&IQ(t1.CountyName,t2.
    CountyName)
t1&t2&EQ(t1.ZipCode,t2.ZipCode)&IQ(t1.EmergencyService,
    t2.EmergencyService)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&IQ(t1.City,t2
    .City)
t1&t2&EQ(t1.MeasureName,t2.MeasureName)&IQ(t1.
    MeasureCode,t2.MeasureCode)
```

Listing 2: Denial Constraints for *Flights*

```
t1&t2&EQ(t1.flight,t2.flight)&IQ(t1.act_dep_time,t2.
    act_dep_time)
t1&t2&EQ(t1.flight,t2.flight)&IQ(t1.act_arr_time,t2.
    act_arr_time)
```

```
t1&t2&EQ(t1.flight,t2.flight)&IQ(t1.sched_dep_time,t2.
    sched_dep_time)
t1&t2&EQ(t1.flight,t2.flight)&IQ(t1.sched_arr_time,t2.
    sched_arr_time)
```

Listing 3: Denial Constraints for *Adult*

```
t1&EQ(t1.Relationship,"husband")&IQ(t1.Sex,"male")
t1&EQ(t1.Relationship,"wife")&IQ(t1.Sex,"female")
t1&EQ(t1.Maritalstatus,"married-civ-spouse")&IQ(t1.
    Relationship,"husband")&IQ(t1.Relationship,"wife")
t1&EQ(t1.Maritalstatus,"divorced")&IQ(t1.Sex,"male")&IQ
    (t1.Sex,"female")
t1&EQ(t1.Relationship,"unmarried")&IQ(t1.Sex,"male")&IQ
    (t1.Sex,"female")
t1&EQ(t1.Relationship,"not-in-family")&IQ(t1.Sex,"male"
    )&IQ(t1.Sex,"female")
t1&EQ(t1.Relationship,"other-relative")&IQ(t1.Sex,"male
    ")&IQ(t1.Sex,"female")
t1&EQ(t1.Relationship,"own-child")&IQ(t1.Sex,"male")&IQ
    (t1.Sex,"female")
t1&EQ(t1.Relationship,"wife")&IQ(t1.Sex,"male")&IQ(t1.
    Sex,"female")
t1&EQ(t1.Relationship,"husband")&IQ(t1.Sex,"male")&IQ(
    t1.Sex,"female")
t1&EQ(t1.Sex,"male")&IQ(t1.Workclass,"?")&IQ(t1.
    Workclass,"federal-gov")&IQ(t1.Workclass,"private")&
    IQ(t1.Workclass,"self-emp-inc")&IQ(t1.Workclass,"
    self-emp-not-inc")&IQ(t1.Workclass,"state-gov")&IQ(
    t1.Workclass,"local-gov")
t1&EQ(t1.Sex,"female")&IQ(t1.Workclass,"?")&IQ(t1.
    Workclass,"federal-gov")&IQ(t1.Workclass,"private")&
    IQ(t1.Workclass,"self-emp-inc")&IQ(t1.Workclass,"
    self-emp-not-inc")&IQ(t1.Workclass,"state-gov")&IQ(
    t1.Workclass,"local-gov")
t1&EQ(t1.Sex,"male")&IQ(t1.Race,"amer-indian-eskimo")&
    IQ(t1.Race,"asian-pac-islander")&IQ(t1.Race,"black")
    &IQ(t1.Race,"other")&IQ(t1.Race,"white")
t1&EQ(t1.Sex,"female")&IQ(t1.Race,"amer-indian-eskimo")
    &IQ(t1.Race,"asian-pac-islander")&IQ(t1.Race,"black"
    )&IQ(t1.Race,"other")&IQ(t1.Race,"white")
t1&EQ(t1.Sex,"male")&IQ(t1.Age,"<18")&IQ(t1.Age,">50")&
```

```
    IQ(t1.Age,"31-50")&IQ(t1.Age,"22-30")&IQ(t1.Age,"
    18-21")
t1&EQ(t1.Sex,"female")&IQ(t1.Age,"<18")&IQ(t1.Age,">50"
    )&IQ(t1.Age,"31-50")&IQ(t1.Age,"22-30")&IQ(t1.Age,"
    18-21")
t1&EQ(t1.Sex,"male")&IQ(t1.Income,"lessthan50k")&IQ(t1.
    Income,"morethan50k")
t1&EQ(t1.Sex,"female")&IQ(t1.Income,"lessthan50k")&IQ(
    t1.Income,"morethan50k")
t1&EQ(t1.Income,"morethan50k")&IQ(t1.Sex,"female")&IQ(
    t1.Sex,"male")
t1&EQ(t1.Income,"lessthan50k")&IQ(t1.Sex,"female")&IQ(
    t1.Sex,"male")
t1&EQ(t1.Sex,"male")&IQ(t1.Education,"10th")&IQ(t1.
    Education,"11th")&IQ(t1.Education,"12th")&IQ(t1.
    Education,"1st-4th")&IQ(t1.Education,"5th-6th")&IQ(
    t1.Education,"7th-8th")&IQ(t1.Education,"9th")&IQ(t1
    .Education,"assoc-acdm")&IQ(t1.Education,"assoc-voc"
    )&IQ(t1.Education,"bachelors")&IQ(t1.Education,"
    doctorate")&IQ(t1.Education,"hs-grad")&IQ(t1.
    Education,"masters")&IQ(t1.Education,"prof-school")&
    IQ(t1.Education,"some-college")
t1&EQ(t1.Sex,"female")&IQ(t1.Education,"10th")&IQ(t1.
    Education,"11th")&IQ(t1.Education,"12th")&IQ(t1.
    Education,"1st-4th")&IQ(t1.Education,"5th-6th")&IQ(
    t1.Education,"7th-8th")&IQ(t1.Education,"9th")&IQ(t1
    .Education,"assoc-acdm")&IQ(t1.Education,"assoc-voc"
    )&IQ(t1.Education,"bachelors")&IQ(t1.Education,"
    doctorate")&IQ(t1.Education,"hs-grad")&IQ(t1.
    Education,"masters")&IQ(t1.Education,"prof-school")&
    IQ(t1.Education,"some-college")
t1&EQ(t1.Sex,"male")&IQ(t1.Occupation,"?")&IQ(t1.
    Occupation,"adm-clerical")&IQ(t1.Occupation,"craft-
    repair")&IQ(t1.Occupation,"exec-managerial")&IQ(t1.
    Occupation,"farming-fishing")&IQ(t1.Occupation,"
    handlers-cleaners")&IQ(t1.Occupation,"machine-op-
    inspct")&IQ(t1.Occupation,"other-service")&IQ(t1.
    Occupation,"priv-house-serv")&IQ(t1.Occupation,"prof
    -specialty")&IQ(t1.Occupation,"protective-serv")&IQ(
    t1.Occupation,"sales")&IQ(t1.Occupation,"tech-
    support")&IQ(t1.Occupation,"transport-moving")
```

```
t1&EQ(t1.Sex,"female")&IQ(t1.Occupation,"?")&IQ(t1.
    Occupation,"adm-clerical")&IQ(t1.Occupation,"craft-
    repair")&IQ(t1.Occupation,"exec-managerial")&IQ(t1.
    Occupation,"farming-fishing")&IQ(t1.Occupation,"
    handlers-cleaners")&IQ(t1.Occupation,"machine-op-
    inspct")&IQ(t1.Occupation,"other-service")&IQ(t1.
    Occupation,"priv-house-serv")&IQ(t1.Occupation,"prof
    -specialty")&IQ(t1.Occupation,"protective-serv")&IQ(
    t1.Occupation,"sales")&IQ(t1.Occupation,"tech-
    support")&IQ(t1.Occupation,"transport-moving")
```

D Extracts of dirty datasets

Listing 4: First 15 lines of adult.csv

```
ProviderNumber,HospitalName,Address1,Address2,Address3,
    City,State,ZipCode,CountyName,PhoneNumber,
    HospitalType,HospitalOwner,EmergencyService,
    Condition,MeasureCode,MeasureName,Score,Sample,
    Stateavg
10018,callahan eye foundation hospital,1720 university
    blvd,,,birmingham,al,35233,jefferson,2053258100,
    acute care hospitals,voluntary non-profit - private,
    yes,surgical infection prevention,scip-card-2,
    surgery patients who were taking heart drugs caxxed
    beta bxockers before coming to the hospitax who were
     kept on the beta bxockers during the period just
    before and after their surgery,,,al_scip-card-2
10018,callahan eye foundation hospital,1720 university
    blvd,,,birmingham,al,35233,jefferson,2053258100,
    acute care hospitals,voluntary non-profit - private,
    yes,surgical infection prevention,scip-inf-1,surgery
     patients who were given an antibiotic at the right
    time (within one hour before surgery) to help
    prevent infection,,,al_scip-inf-1
10018,callahan eye foundation hospital,1720 university
    blvd,,,birmingham,al,35233,jefferson,2053258100,
    acute care hospitals,voluntary non-profit - private,
    yes,surgical infection prevention,scip-inf-2,surgery
     patients who were given the  right kind  of
    antibiotic to help prevent infection,,,al_scip-inf-2
10018,callahan eye foundation hospital,1720 university
    blvd,,,birminghxm,al,35233,jefferson,2053258100,
    acute care hospitals,voluntary non-profit - private,
    yes,surgical infection prevention,scip-inf-3,surgery
     patients whose preventive antibiotics were stopped
    at the right time (within 24 hours after surgery),,,
    al_scip-inf-3
10018,callahan eye foundation hospital,1720 university
    blvd,,,birmingham,al,35233,jefferson,2053258100,
    acute care hospitals,voluntary non-profit - private,
    yes,surgical infection prevention,scip-inf-4,all
```

```
   heart surgery patients whose blood sugar (blood
   glucose) is kept under good control in the days
   right after surgery,,,al_scip-inf-4
10018,callahan eye foundation hospital,1720 university
   blvd,,,birmingham,al,35233,jefferson,2053258100,
   acute care hospitals,voluntary non-profit - private,
   yes,surgical infection prevention,scip-inf-6,surgery
    patients needing hair removed from the surgical
   area before surgery who had hair removed using a
   safer method (electric clippers or hair removal
   cream c not a razor),,,al_scip-inf-6
10018,callahan eye foundation hospital,1720 university
   blvd,,,birmingham,al,35233,jefferson,2053258100,
   acute care hospitals,voluntary non-profit - private,
   yes,surgical infection prevention,scip-vte-1,surgery
    patients whose doctors ordered treatments to
   prevent blood clots after certain types of surgeries
   ,,,al_scip-vte-1
10018,callahan eye foundation hospital,1720 university
   blvd,,,birmingxam,al,35233,jefferson,2053258100,
   acute care hospitals,voluntary non-profit - private,
   yes,surgical infection prevention,scip-vte-2,
   patients who got treatment  at the right time (
   within 24 hours before or after their surgery) to
   help prevent blood clots after certain types of
   surgery,,,al_scip-vte-2
10019,helen keller memorial hospital,1300 south
   montgomery avenue,,,sheffield,al,35660,jefferson
   ,2563864556,acute care hospitals,government -
   hospital district or authority,yes,heart attack,ami
   -1,heart attack patients given aspirin at arrival
   ,97%,33 patients,al_ami-1
10019,helen keller memorial hospital,1300 south
   montgomery avenue,,,sheffield,al,35660,jefferson
   ,2563864556,acute care hospitals,government -
   hospital district or authority,yes,heart attack,ami
   -2,heart attack patients given aspirin at discharge
   ,92%,13 patients,al_ami-2
10019,helen keller memorial hospital,1300 south
   montgomery avenue,,,sheffxeld,al,35660,jefferson
   ,2563864556,acute care hospitals,government -
```

```
    hospital district or authority,yes,heart attack,ami
    -3,heart attack patients given ace inhibitor or arb
    for left ventricular systolic dysfunction (lvsd)
    ,75%,4 patients,al_ami-3
10019,helen keller memorial hospital,1300 south
    montgomery avenue,,,sheffield,al,35660,jefferson
    ,2563864556,acute care hospitals,government -
    hospital district or authority,yes,heart attack,ami
    -4,heart attack patients given smoking cessation
    advice/counseling,100%,4 patients,al_ami-4
10019,helen keller memorial hospital,1300 south
    montgomery avenue,,,sheffield,al,35660,jefferson
    ,2563864556,acute care hospitals,government -
    hospital district or authority,yes,heart attack,ami
    -5,heart attack patients given beta blocker at
    discharge,86%,14 patients,al_ami-5
1xx19,helen keller memorial hospital,1300 south
    montgomery avenue,,,sheffield,al,35660,jefferson
    ,2563864556,acute care hospitals,government -
    hospital district or authority,yes,heart attack,ami
    -7a,heart attack patients given fibrinolytic
    medication within 30 minutes of arrival,,0 patients,
    al_ami-7a
```

Listing 5: First 15 lines of flights.csv

```
tuple_id,src,flight,sched_dep_time,act_dep_time,
    sched_arr_time,act_arr_time
1,aa,AA-3859-IAH-ORD,7:10 a.m.,7:16 a.m.,9:40 a.m.,9:32
     a.m.
2,aa,AA-1733-ORD-PHX,7:45 p.m.,7:58 p.m.,10:30 p.m.,
3,aa,AA-1640-MIA-MCO,6:30 p.m.,,7:25 p.m
4,aa,AA-518-MIA-JFK,6:40 a.m.,6:54 a.m.,9:25 a.m.,9:28
    a.m.
5,aa,AA-3756-ORD-SLC,12:15 p.m.,12:41 p.m.,2:45 p.m
    .,2:50 p.m.
6,aa,AA-204-LAX-MCO,11:25 p.m.,,12/02/2011 6:55 a.m.,
7,aa,AA-3468-CVG-MIA,7:00 a.m.,7:25 a.m.,9:55 a.m.,9:45
     a.m.
8,aa,AA-484-DFW-MIA,4:15 p.m.,4:29 p.m.,7:55 p.m.,7:39
    p.m.
9,aa,AA-446-DFW-PHL,11:50 a.m.,12:12 p.m.,3:50 p.m
```

```
    . ,4:09 p.m.
10,aa,AA-466-IAH-MIA,6:00 a.m.,6:08 a.m.,9:20 a.m.,9:05
    a.m.
11,aa,AA-1886-BOS-MIA,10:45 a.m.,10:55 a.m.,2:20 p.m
    .,1:40 p.m.
12,aa,AA-2957-DFW-CVG,7:55 a.m.,8:04 a.m.,11:05 a.m
    .,11:01 a.m.
13,aa,AA-1664-MIA-ATL,10:15 a.m.,10:18 a.m.,12:10 p.m
    .,11:56 a.m.
14,aa,AA-3979-CVG-ORD,7:30 a.m.,8:04 a.m.,8:00 a.m
    .,8:06 a.m.
```

Listing 6: First 15 lines of adult_dirty0.1.csv

```
Age,Workclass,Education,Maritalstatus,Occupation,
    Relationship,Race,Sex,HoursPerWeek,Country,Income
31-50,Private,Some-college,Marrxed-civ-spouse,Craft-
    repair,Husband,White,Male,40,United-States,
    LessThan50K
>50,Fexeral-gov,Some-college,Never-married,Exec-
    managerial,Own-child,Black,Female,40,United-States,
    LessThan50K
31-50,Private,Bachelors,Marriex-civ-spouse,Sales,
    Husband,White,Male,x0,United-States,LessThan50K
22x30,Self-emp-inc,HS-grad,Never-married,Craft-repair,
    Own-child,Whitx,Male,40,United-States,LxssThan50K
22-30,Private,HS-grad,Married-civ-spouse,Farming-
    fishing,Husband,White,Femalx,40,United-States,
    LessThan50K
31-50,Private,Some-college,Married-civ-spouse,Craft-
    repair,Husbxnd,White,Male,40,United-States,
    LessThan50K
31-50,Private,HS-grad,Never-married,Prof-specialty,Not-
    in-family,White,Female,3x,United-Stxtes,LessThan50K
31-50,State-gov,Prof-school,Married-civ-spouse,Prof-
    specialty,Husband,Asian-Pac-Islander,Maxe,72,Ixdia,
    MoreThan50K
18-21,Self-emp-not-inc,Some-college,Never-married,xdm-
    clerical,Own-child,White,Female,x5,United-States,
    LessThan50K
>50,Private,HS-grad,Married-civ-spouse,Farming-fishing,
    Husband,White,Male,50,United-States,LessThan50K
```

```
>50,Private ,Assoc-voc ,Married-civ-spouse ,Prof-specialty
    ,Husband ,Whxte,Male ,45 ,United-States ,LessThan50K
>50,Private ,HS-grad ,Married-civ-spouse ,Sales ,Husband ,
    White ,Female ,48 ,United-States ,MoreThan50K
31-50,Private ,Bachelors ,Married-civ-spouse ,Exec-
    managerial ,Husband ,White ,Male ,50 ,United-States ,
    MoreThan50K
22-30,Private ,HS-grad ,Never-married ,Craft-repair ,Not-in
    -family ,White ,Male ,40 ,United-States ,LessThan50K
```

E Progress Diaries

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF Computing, Engineering and the Built Environment

PROJECT DIARY

Student: Daniel Van Dijke **Supervisor: Dr Taoxin Peng**

1. **Date: 07/02/2024** **Last diary date: 31/01/24**

Objectives:

1. Define aims and Objectives of the project – in dissertation
2. Continue -- literature review (documentation)
    a. Background study
        i. Data Quality, Data Cleaning
        ii. Machine Learning
    b. Machine Learning in Data Cleaning
        i. In theory
        ii. In implementation
        iii. Benchmark datasets
    c. Related work - continue
    d. Try to find more recent published references (since 2020)
3. Data collection (benchmark data on this topic if possible)
4. Implement the HoloClean framework for the purpose of testing
5. Continue – update reference list

Progress:

1. Done. Aims and Objectives moved from updated IPO to dissertation
2. Continued. Extra section (~500 words) added to literature review
3. Partially done. 2 benchmark datasets found (“Hospital” and “Flights”), more still to look for
4. Done. Set up and ready for testing
5. Continued references being added.

Supervisor’s Comments:

Excellent progress. Two benchmark datasets are good enough. However, it would be even better if you are able to find a third one with a bigger size.

Also, a successful implementation of the HoloClean framework is good enough for some level of testing. It would be better if you can find another one which is not too old.

Another challenging task is to modify the existing HoloClean framework, based on recommendations by other researchers in literature, plus your won critical suggestions/ideas.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF Computing, Engineering and the Built Environment

PROJECT DIARY

Student: Daniel Van Dijke **Supervisor: Dr Taoxin Peng**

1. **Date: 31/01/2024** **Last diary date: 25/10/23**

Objectives:

1. Define aims and Objectives of the project
2. Update the project plan
3. Use the template – define a structure of the dissertation
4. Continue -- literature review (documentation)
    a. Background study
        i. Data Quality, Data Cleaning
        ii. Machine Learning
    b. Machine Learning in Data Cleaning
        i. In theory
        ii. In implementation
        iii. Benchmark datasets
    c. Related work
    d. Try to find more recent published references (since 2020)
5. Data collection (benchmark data on this topic if possible)
6. Implement the HoloClean framework for the purpose of testing
7. Find a couple of good sample dissertations that are similar to yours
8. Create a reference list

Progress:

1. Partially done. IPO has been updated but the aims still need to be written in the dissertation.
2. Done. Gantt Chart used in Jira updated
3. Done. Work migrated to the dissertation template
4. Further Completion. More references still to be written about, and literature review needs redrafting to provide a more cohesive structure.
5. Not Done. Still need to look into the benchmark datasets that are used in academic papers
6. Partially done. Relevant software installed, still needs to be set up.
7. Done
8. Done (see .bib file in dissertation).

Supervisor's Comments:

Good progress, especially starting to look at implementation possibilities. Some suggestions for materials that need in LR:

1. Background of data cleaning
2. Background knowledge of Machine learning in Data cleaning, including techniques/models

These should be reviewed before actual frameworks/approaches.

It is good to find another open-source framework, although it was year old. You can make some improvement based on the existing one.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF Computing, Engineering and the Built Environment

PROJECT DIARY

Student: Daniel Van Dijke **Supervisor: Dr Taoxin Peng**

1. **Date: 3/10/2023** **Last diary date: First**

Objectives:

1. Define aims and objectives of the project
2. Create a G-Chart (project plan)
3. Start literature review
    a. Search references, especially research papers.
    b. Background study
        i. Data Quality, Data Cleaning
        ii. Machine Learning
    c. Machine Learning in Data Cleaning
4. Look at the module's website, pay attention to following information:
    a. How to do a literature review
    b. How to search information, papers etc
    c. What is a methodology
    d. Find the dissertation template
    e. Find a couple of sample dissertations
5. Create a reference list

Progress:

1. Partially done, scope of project to be reassessed.
2. To be completed next week after project scope is redefined.
3. Partially done, more background study required
4. Done
5. Done

Supervisor's Comments:

F Gantt Chart

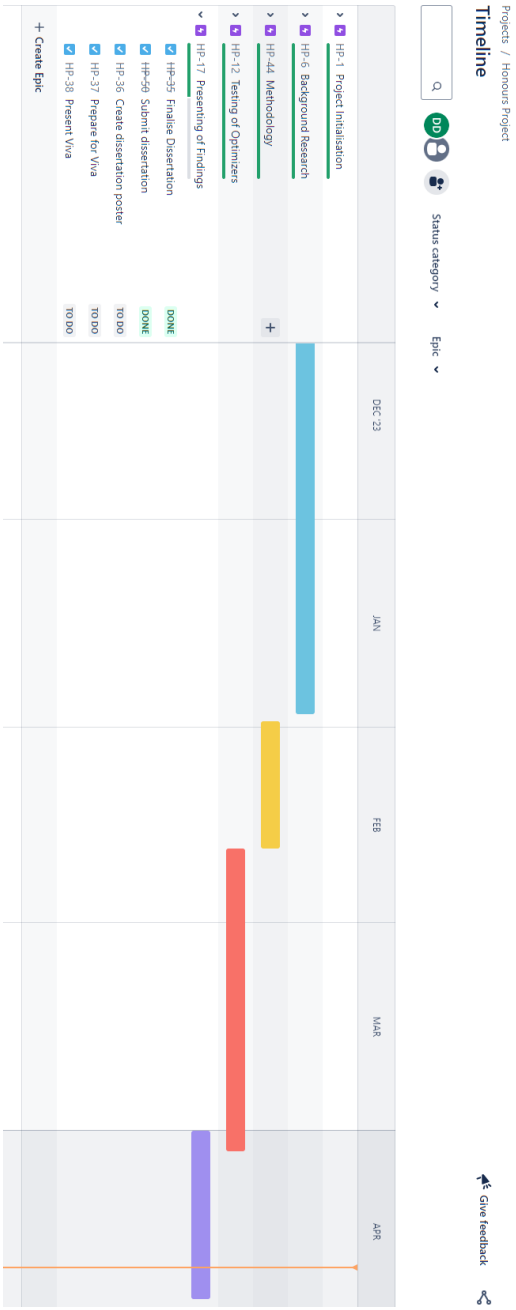

Figure 15: Screenshot of Gantt Chart used to manage project timeline