An Iterative and Feedback-based Data Cleaning approach for enhancing Machine Learning Models

Rishi Mody, Satya Narayan Shukla and Utkarsh Srivastava
College of Information and Computer Sciences
University of Massachesetts Amherst
Amherst, MA 01003-9264

***Abstract*—Processing and cleaning dirty datasets has always been a major bottleneck for any data analytics process. For a general Machine Learning prediction task, correctness in data implies higher final model accuracy. Generally cleaning is done iteratively - clean some data, analyze and clean some more data based on the analysis. Unfortunately, for statistical models, iterative cleaning and re-training can lead to misleading results even for a simple model [explained using Simpsons Paradox]. The paper explores an iterative data cleaning process in the context of training Machine Learning models that only enhances modeling accuracy at every step of the cleaning process by updating model parameters. The paper further provides a novel way of integrating a feedback-mechanism from the Cleaner module and the updated ML model for choosing subsequent dirty samples for clean-up. The experimentation is done on the raw UCI Adult dataset with additional inclusions of corrupted data. Evaluation of results on this dataset conforms with the expected nature described in the paper.**

Keywords*—*Active Clean, Data Cleaning, Machine Learning, Convex Loss Functions, Prediction Task, Cleaner, Sampler, Conditional Functional Dependency, Integrity Constraints, Dirtiness Confidence Evaluator.

I. Introduction

In recent times, Machine Learning is being increasingly used in diverse world domains to solve problems that can be very complex for a human-being. Availability of large amounts of documented data has only facilitated research in this area. Having a well structured data-representation (space) is necessary but not sufficient to extract as much information from data as possible for a general Machine Learning task. This requires making sure that the data is "correct" which directly co-relates with the field of "Data Cleaning" in Database theory. This provides a great opportunity to couple Data Cleaning practices and Machine Learning approaches together efficiently, where one can benefit from the other. Hence, in this paper we aim to provide a novel implementation and an extension of prior work, in presenting an interactive and feedback-based data cleaning approach for improving accuracy and correctness of Machine Learning prediction models. This paper focuses on a certain kind of Machine Learning model whose loss function converges with iteration count. Using such models can help us in giving more structure to this task - where instead of re-training the model each time upon cleaning a set of sample records, we can update the model parameters by making sure that with each parameter update step, the model accuracy only increases. Further, by analyzing the degree of improvement gained in accuracy for a given iteration, we can get an idea on the kind of data we should sample from the dirty database in the next iteration to clean and update in order to get a better prediction model. The data cleaning approach used in the paper can be formalized using traditional database practices like Conditional Functional Dependencies, Integrity Constraints etc. by evaluating the property of the given dataset being worked upon. Since a major chunk of our project work includes validating the approach taken by 'Active Clean' [1] and extending it using novel implementation approaches, the implementation has been coded up from scratch.

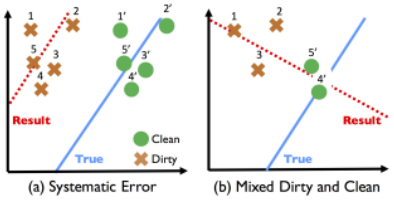


Fig. 1: (a) Systematic corruption in one variable can lead to a shifted model. The dirty examples are labeled 1-5 and the cleaned examples are labeled 1-5. (b) Mixed dirty and clean data results in a less accurate model than no cleaning.

With the requirement of having clean and correct data for better Machine Learning model creation, we can argue why simple iterative cleaning and model re-training will not necessarily be a good approach. Firstly, the presence of very high-dimensions in data is a very strong factor affecting this analysis. Further, observing trends of the Simpsons Paradox on the model training process provides a clear picture of the weakness of this approach - if we assume training a Linear Regression model with a given dirty dataset [Fig. 1a] and the data cleaning process cleans only a few data points [Fig. 1b], then the intermediate result reveals an even worse model than [Fig. 1a] which is no-where close to the "True" scenario [Fig. 1a]. Given the vast scope of ML tasks in application and the desired high accuracy levels of such methods, we need to tackle the above mentioned challenges by specifically focusing on analyzing and maintaining the well-structuredness and correctness of data - which can be hard. Further, this area of research work on the intersection of Database Theory and Machine Learning for Data Cleaning is relatively newer in its initiation. As a solution to the mentioned problem, we can formalize a method to explore an iterative data cleaning process in the context of training machine learning models that only enhances modeling accuracy at every step of the cleaning process without re-training along with providing a feedback to sample dirty data for cleaning that affects positive prediction

accuracy more (as described above). Further, in the presence of resource or time limitations, this iterative approach can scale well in still providing a better trained model along increasing timeline steps.

The structure of this paper details information about existing research work done in similar areas and broad Literature Survey in Section II. Then the paper proceeds to discuss and compare the system architecture of ActiveClean and the proposed extension in Section III. Section IV describes the Dataset chosen for evaluation along with all the Database technologies used in the Cleaner module in Section V. Section VI presents a detailed analysis of the experimental setup and produced results. The paper concluded with Section VII along with a discussion on some scope of future work in Section VIII.

II. Literature Review

Data Cleaning is a useful and necessary step to remove errors and inconsistencies from a database in order to ensure correctness in data [2]. Since data can be aggregated from multiple sources and could get very large in size, an efficient data cleaning process is desired. Since manual data cleaning might be burdensome, innovative automated techniques need to be formulated that satisfy generalizability as well. The data cleaning component can be applied to both the schema-level and data instance-level. Conventional techniques suggested by [2] take into account data analytics using profiling of actual data instances to determine a better schema, standardization of data entries (in terms of format), writing user-defined functions in SQL and determining a similarity threshold for matching instances, among other techniques. Though, a lot of them have limitations in the form of domain-specific application, function re-implementation and sometimes, manual intervention. It further provides examples of real-world tools being applied in practice - textitMigration Architect, Integrity, DataCleanser etc.

With the advent of strong Machine Learning techniques in recent years, integration of ML based approaches with Database technology has not been thoroughly explored. [3] tries to take this approach one step further by applying fuzzy inference learning techniques from the area of natural language processing in specifically doing duplicate data identification. This not only helps in minimizing coverage of hard-coded rules, but also paves way for making the process take advantage of its own data representation (with very little user intervention) and make the machinery flexible. But it still remained weak due to relying on domain-specific knowledge.

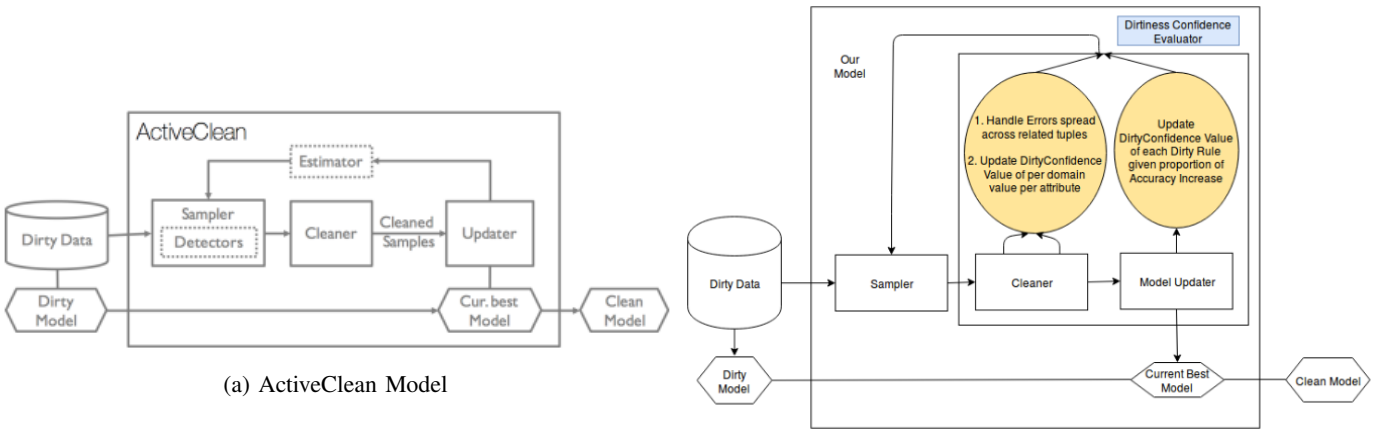
Sample and Clean [4] talks about a framework for fast and accurate query processing on dirty data, where data cleaning is applied to a small subset of data and the results of the cleaning process are further used to lessen the impact of dirty data on aggregate query answers. This framework can produce significant improvements in accuracy compared to query processing without data cleaning and speed compared to data cleaning without sampling. Sampling has also been used to find duplicates in relation [14]. Similarly, the problem of query-oriented data cleaning has been explored in [15], where they clean data relevant to the given query only.

There has been research on data repairing techniques [5] using machine learning (ML) methods for cleaning dirty databases by modifying the erroneous value. It addresses the scalability and accuracy of the replacement value by using statistical ML techniques to predict better updates for repairing dirty databases. This method is primarily based on maximizing the likelihood of the data to be replaced given the underlying distribution of data - which is achieved by training a model to evaluate the likelihood of the proposed replacement. This method outperforms other existing methods on large databases with any type of data (i.e. string, numeric, categorical, ordinal) or any type of error (missing, incomplete or inconsistent values). This method has several advantages over previous methods: the accuracy and the scalability is guaranteed; the cost of repair is bounded. Machine Learning can also be used to extrapolate the rules from a small set of data cleaned by human [12], [16]. This can be combined with active learning [17] to learn an accurate model with fewest possible number of examples. While, our project is similar to these approaches, it addresses a very different problem of data cleaning before user-defined modeling.

A lot of research has been focused on the use of preliminary analysis on dirty data as a guide to help identify potential errors and design repairs [6], [7], [8]. But in case of statistical models, iteratively cleaning and re-training on partially cleaned set can lead to totally unexpected and misleading results. This is a typical example of Simpsons paradox where a trend appears in different groups of data but could disappear or reverse in behavior when these groups are seen as a whole [9]. The challenges with Simpsons paradox still prevail because recent SQL data cleaning techniques like Sample-and-Clean [4] and Progressive Data Cleaning [10], [11], [12] actually focus on cleaning subsets of data to avoid expensive costs. Such approaches need to be re-evaluated in data cleaning for statistical modeling, and this paper explores the extensions of such approaches for statistical modeling by providing guarantees of convergence for convex loss models.

III. System Architecture

The research work being done in ActiveClean [1] has been a strong step towards coupling Data Cleaning with Machine Learning models. As described in previous sections of the paper, the need for working with “correct” training data for ML models is highly desirable. It focuses effort on cleaning data for statistical models which can be represented as a minimization of convex loss functions like logistic regression (LR) and support vector machines (SVM). In our current work, we have taken inspiration from the framework developed as a part of ActiveClean machinery. We have taken inspiration from the design of the basic pipeline of ActiveClean and combined it with our own modifications to propose extensions. Since this work tries to reproduce and compare the solution provided by ActiveClean and our extensions to this present framework, it would be nice to give a brief overview of the main ActiveClean modules before we proceed to highlight the similarities and differences between the existing framework and our proposed model.



(a) ActiveClean Model

(b) Our Model

Fig. 2: Comparison between ActiveClean and Our Model

A. *ActiveClean*

The system architecture of ActiveClean can be seen in Figure 2a. It iteratively cleans dirty data for statistical modeling but does not re-train the entire model. Data tuples are cleaned in batches and then fed back into the model to update the model parameters. A further optimisation in ActiveClean tries to reach convergence faster by using a separate Machine Learning estimator as an optimisation step to also sample clean data with certain confidence from the dirty dataset, in each subsequent iteration.

1) Initialization: Initially, for an assumed dirty dataset R, let $R_{dirty} = R$ and $R_{clean} = \emptyset$. The system first trains a model on the dirty data to find initial model parameter θ which needs to be improved iteratively till convergence. Stochastic Gradient Descent (SGD) converges for any arbitrary initialization [13], so θ need not be accurate.

2) Sampler: The Sampler selects a fixed batch of data tuples (S) from R_{dirty} randomly which are inspected to be dirty. These are then sent to the Cleaner module.

3) Cleaner: The Cleaner is an independent module which inspects a data tuple for being dirty and cleans it using technology described in Section V. Identification of a tuple being dirty and cleaning it are 2 separate sub-steps. In a general framework, this can be manual or automated. Further, ActiveClean assumes that no error exists that simultaneously affects multiple tuples - including data duplicacy issues.

4) Updater - Model Parameter: The main focus here is to model data cleaning problem as a stochastic gradient descent (SGD) step [13]. SGD is an iterative optimization procedure that tries to minimize an objective function by taking steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. In the current instance, parameter θ is updated iteratively (in the direction of θ corresponding to total clean data) by updating gradient with respect to current clean data. This helps in fitting a better model by reassigning weights of the parameter constituents instead of entire re-training - close to what a total clean dataset parameter would behave like. The definition of "better" is defined to be the one having higher training accuracy.

The current implementation trains and updates models based on Logistic Regression. For such convex loss models, SGD converges to a minimum.

Loss Function : Let x_i is a feature vector and y_i is a label, the loss function of Logistic Regression is defined as:

$$\mathcal{L}(x_i, y_i, \theta) = \frac{1}{ln2}(ln(1 + e^{-y.f(x)}))$$

$$\theta = argmin_\theta \sum_{i=1}^{N} \mathcal{L}(x_i, y_i, \theta) + r(\theta)$$

$$\theta : \{\mathbf{W}, b\}$$

where $r(\theta)$ is the regularizer term that is added to the loss function to penalize overly large weights.
The gradient with respect to θ i.e. $\mathbf{W}$ and b is computed as follows:

$$\frac{\partial \mathcal{L}(x_i, y_i, \theta)}{\partial \mathbf{W}} = \frac{-y.x.e^{-y.f(x)}}{1 + e^{-y.f(x)}} + 2W$$

$$\frac{\partial \mathcal{L}(x_i, y_i, \theta)}{\partial \mathbf{b}} = \frac{-y.e^{-y.f(x)}}{1 + e^{-y.f(x)}} + 2b$$

Model Update : Our aim is to obtain a set of model parameters which minimizes the loss function. This becomes an optimization problem to solve for θ minimizing the convex loss. The model parameters are updated iteratively by taking a sequence of steps in opposite direction of the gradient. At the optimal point, the magnitude of gradient will become zero. The update equation at the $(t+1)^{th}$ step can be written as (given γ learning rate):

$$\theta^{(t+1)} \leftarrow \theta^t - \gamma \cdot \nabla \mathcal{L}(\theta^t)$$

Technically, this gradient depends on all of the cleaned data, but here we try to approximate it with a batch of cleaned data iteratively. This is done such that if we need to stop at

any iteration, we always have the most accurate model based on training accuracy metric. The true gradient $g^*(\theta)$ can be expressed as:

$$g^*(\theta) = \nabla\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\nabla\mathcal{L}(x_i, y_i, \theta)$$

We approximate this gradient with constituents from both already cleaned and sampled data cleaned in current iteration, as follows:

$$g(\theta) = \frac{|R_{clean}|}{|R|}\cdot g_c(\theta) + \frac{|R_{dirty}|}{|R|}\cdot g_s(\theta)$$

$g_c(\theta)$ is the gradient calculated on already cleaned data and can be expressed as:

$$g_c(\theta) = \frac{1}{|R_{clean}|}\sum_{i\in R_{clean}}\nabla\mathcal{L}(x_i, y_i, \theta)$$

$g_s(\theta)$ is the gradient estimated on newly cleaned data and can be expressed as:

$$g_s(\theta) = \frac{1}{|S|}\sum_{i\in S}\nabla\mathcal{L}(x_i, y_i, \theta)$$

where S is the sample of data taken from dirty set and cleaned with cleaner $C(\cdot)$. Then, at each iteration:

$$\theta^{(t+1)} \leftarrow \theta^t - \gamma\cdot g(\theta^t)$$

5) Updater - Data Tabulation: The cleaned sample is added to R_{clean} and its original entry is removed from R_{dirty}. Finally, the system will terminate if all the data is cleaned or training error converges.

6) Optimization Modules : Detector & Estimator: These 2 modules, in sync, act as optimizers in the existing machinery that help in faster convergence to a completely clean model. They help in identifying and estimating the effect of cleaning a given set of tuples on the current model. However, it is partly rule-based and partly Machine Learning-based. The necessary information is gathered from the output of the Cleaner module using which each tuple in the sample is classified with a label of being 'dirty' v/s being 'clean' and added to the current Estimator model for entire re-training.

This updated trained model is applied in the next iteration of sampling to the entire updated R_{dirty} to predict and find any tuples that belong to the 'clean' label and consequently, should ideally be pushed to R_{clean}. Using this information, even before sampling a new batch, a part of R_{dirty} is directly added to R_{clean} and the normal sampling continues. This prediction estimator model is re-trained using label information from each new sample with increasing iteration. Since, in each iteration, atleast S tuples are extracted from R_{dirty} at the end of the Sampler module, we will reach convergence faster in terms of total no. of data elements in R_{dirty} to inspect. But, as is true for any ML system, this suffers from having False Positives - i.e. dirty tuples in reality being labeled wrongly as clean. This introduces a trade-off between accuracy v/s speed.

B. Our Model - Similarities and Differences

Since the scope of this work is not limited to solely reproducing and validating the claims made by the work done in ActiveClean [1], we propose a modification and further extension to the existing framework of ActiveClean. It cherry picks the best part about the model machinery described in the existing work and couples it with separate modules that deal with limitations of ActiveClean to:

(a) handle the same erroneous structure across multiple tuples, and

(b) improve the sampling algorithm by using a combination of domain-dependent and domain-independent heuristic which still converges faster than the base algorithm, but does not provide scope for any false positives (i.e. to replace the ML-based Estimator-Detector module).

The complete architecture of our extended model is described in Figure 2b.

Similarities to ActiveClean : The basic pipeline for the framework is inspired from the skeleton architecture of ActiveClean. Even though being different in implementation, our framework also consists of a Model Initializer, a Sampler, a Cleaner and a Model Updater. The intuition of using newly cleaned sample to update the model parameter through the gradient update step has been borrowed in our implementation as well.

Differences from ActiveClean : Though the basic pipeline follows a similar structure to ActiveClean, a very major component of the pipeline is to intelligently select samples of data from R_{dirty} to clean in each iteration. Since, the initial R_{dirty} assumes the entire dataset to be entirely dirty and our model parameter update depends on cleaning strictly dirty data, it is always better to ideally sample the 'real' dirty tuples first and then leave the 'real' clean ones for later iterations. This will not only help us in reaching convergence faster but also not introduce any errors in the cleaning process due to wrongful estimates. For this, we have designed a new module called '**Dirtiness Confidence Evaluator**' (refer Figure 2b) which is based on a combination of domain-dependent and domain-independent heuristic. However, this requires one pass of pre-processing over the entire dataset R_{dirty} before starting the cleaning process (for one of its modules).

Additionally, in all the primary examples described in ActiveClean, the Cleaner module talks about manually cleaning the samples. We argue that even though manual cleaning will be more accurate, we need to find a trade-off between effort and accuracy. It is better to have a pre-defined set of rules than a manual cleaner, such that if the coverage of the set of rules is large, the effort required to clean each individual tuple becomes automated. Further, as the size of dataset increases, manual cleaning can become cumbersome. It is true that for this to be fault-free, the coverage of rules needs to be large and should incorporate information from within and outside the data-domain, it is still a relatively feasible option. For the extension described above, we build a few parts of our pre-processing module on this fundamental rule.

Dirtiness Confidence Evaluator : This module tries to incorporate information from not only the Cleaner (as described in existing work), but also uses information from the Model itself. We introduce the notion of an entity being '*Dirty*' here.

Further, we define a score metric called '*Dirtiness Confidence*' which describes the relative confidence of knowing whether an entity is dirty or not. Here, an entity is defined as either a tuple or a specific data entry in a tuple. Our heuristic to sample dirty data from the current R_{dirty} then boils down to ranking dirtiness confidence value for each tuple in the current state of R_{dirty} and choosing the top S entries (where S is the batch size). Since, cleaning of each data tuple gives us some information about an entity being dirty or not, the dirtiness confidence value of any entity keeps getting updated based on the current state of the system. This will ensure that only entities with high dirtiness confidence values get sampled in each turn of sampling.

Initially, at any given iteration, the probability of a tuple getting sampled through the Sampler is provided a uniform distribution in the total no. of tuples in the current state of R_{dirty}. The evaluation of how 'dirtiness factor' will tweak the porbability value of sampling for a given tuple can be broken down into 3 steps:

1) Handling Duplicates (H) : Information Available from a Related Entity: For any tuple in the current instantiation of R_{dirty}, it might be the case that there exists atleast one other tuple which follows the same error structure. This can be defined in two ways: (i) The current tuple under inspection is the same as the cleaned version of some tuple that was sampled and cleaned in some previous iteration; or (ii) The current tuple under inspection is the same as the dirty version of some tuple that was sampled and cleaned in some previous iteration. If we closely inspect both the cases, the 'Dirty Confidence' value of the tuple as per case (i) should be low and as per case (ii) should be high. But, ActiveClean cannot discover this hidden relationship across tuples.

We use the 'Cleaner' module to track this information. Whenever a sample is sent to the Cleaner module, 2 dictionaries D1 (for clean tuples) and D2 (for initial dirty tuples) are updated that contain the Hash-Value of each tuple, post and after cleaning. The Hash-value in the current task has been defined as a simple string concatenation function. If the tuple was already clean, its hash-value only gets updated in D1. Whereas for the dirty tuple case, its hash-value post cleaning gets updated in D1 and the original hash-value of the original dirty tuple in D2. Now these dictionaries can be used as a look-up function in each subsequent iteration to be used in choosing a better sample.

Using this information, if for any tuple in R_{dirty} belongs in the D1 dictionary, its sampling probability is set to 0 (since it is entirely clean). Alternatively, for the tuple present in D2, its sampling probability is increased by a constant factor F. Due to this re-ordering, the left-over mass is uniformly distributed across all other tuples in R_{dirty} .

2) Handling Attribute-wise Dirty Entries (C): Information Available from a Bound Attribute Value: Analyzing how data cleaning rules are applied, we can infer a very strong structure in how data entries get cleaned. In each cleaning process, a tuple changes a few attribute entries and makes the whole tuple clean. Hence, cleaning happens at the attribute value level. So, if we know all possible domain values of each attribute in the dataset, we can construct and update a dictionary to denote the confidence value of each domain value per attribute. For instance, suppose the attribute 'country' has only a few 'legal' domain values and a value of 'USA' gets cleaned to 'United States' every time it is seen, we know that the confidence of 'USA' being dirty is quite high. This is based on the frequency of a certain domain value getting cleaned v/s staying intact, given the cleaning operation. We try to incorporate this information in our heuristic.

For example, if we see 5 data tuples consecutively in which 'USA' gets cleaned to 'United States', then we know with slightly high confidence that seeing 'USA' another time will more likely by a part of a dirty tuple. Whereas, if we see a tuple with attribute value 'Turkey' and it stays 'Turkey' in consecutive dirty tuples after cleaning, the dirtiness confidence of seeing 'Trukey' in a tuple which is dirty itself, solely due to 'Turkey', is lesser in value. Hence, based on the cleaning process for every tuple in the sample, we can adjust the dirtiness confidence for the seen domain value in an attribute.

Now, to control the capacity of how high or low this confidence value can go, we need to keep bounds. In this model, we restrict the dirtiness confidence value between 0 and 10. Every domain value of domain is initialized with the maximum dirtiness factor of 10 in the beginning. This factor keeps switching based on each recent evidence that we find for that value entry in the cleaning phase. Update of dirtiness value for any given attribute value is slow but exponential in nature. But it should be noted that for a certain attribute, the extent of exponential increase in dirtiness confidence value for an assumed dirty value getting cleaned is lesser than the extent of exponential decrease in dirtiness confidence value for an assumed dirty value to not get cleaned. This is based on the logic that our calculation is more biased towards assuming data to be dirty, hence it would be more productive in finding an entry which could supposedly be clean for an attribute.

Since each update is highly affected by the new evidence seen, over a period of some iterations, the true distribution of dirtiness values for a given attribute will get approximated. For example, if 'United States' never gets cleaned and stays as 'United States', its dirtiness confidence will go down to 0 after a certain no. of fixed iterations. Similarly, if 'USA' gets cleaned to 'United States' every time, its dirtiness confidence will be at the highest level of 10 always. Further, the dirtiness confidence of any missing value in an attribute is always set to 10.

Now to use this information in the current sampling phase, we can find the average dirtiness factor of a given tuple in R_{dirty}. This can be computed by taking the average of the dirtiness confidence of all constituent value entries in the tuple. This can be used as a factor in weighing the probability sampling across all sample contenders from R_{dirty} to be described further.

3) Handling Dirtiness Across Cleaning Rules (CR): Information Available from each Model Update Iteration: The last component included in the proposed extension uses information from both the Cleaner and the Model Update module. This module tries to gather information and update the dirtiness confidence for each tuple based on the set of cleaning rules that can be applied to this tuple. This is the module which requires some information about the relationship between a tuple and a cleaning rule, which needs to be stored as a pre-processing

step even before starting the sample-update iteration. Since the set of data cleaning rules is available before-hand, as a pre-processing step, we can store a dictionary of each tuple mapped to all kinds of cleaning rules that can be applied to the tuple once it has been sampled. This information can be gathered using the dirty data identification sub-step from the Cleaner as a *util* function. It should be noted that there exists a trade-off between efficiency and effort for this computation, we are doing extra analysis of dirty data identification (but not repair) at the cost of converging faster.

The logic behind keeping a mapping for each tuple in the dataset with all possible set of rules that can be applied to it, can be useful in strengthening the dirtiness confidence of a tuple indirectly. To fully flesh it out, we can define a '**Strength Factor**' associated with each rule in the Cleaner rule set. This Strength Factor describes how relevant is a rule in increasing the effective accuracy in each subsequent step of the iterative process.

For example, say in iteration 1, out of 20 dirty tuples cleaned, 15 were cleaned due to application of some Rule R' and 5 were cleaned using some rule R" with an increase in training accuracy from 84% to 86% . Further, in the next iteration, out of some new 25 tuples cleaned, say 22 of them were cleaned on applying Rule R" and 3 due to application of Rule R' with an increase in training accuracy from 86% to 86.01%. Intuitively, we can see that if we sample more tuples that get cleaned using Rule R', we are likely to see a better increase in accuracy than using samples that get cleaned using rule R". This helps us in assigning a relative 'Strength Factor' to each rule based on its contribution in increasing the extent of training accuracy increase upon model parameter update. Hence, StrengthFactor(R') = $\frac{86}{84}\cdot\frac{15}{20} + \frac{86.01}{86}\cdot\frac{3}{25}$ and StrengthFactor(R") = $\frac{86}{84}\cdot\frac{5}{20} + \frac{86.01}{86}\cdot\frac{22}{25}$. Now using the pre-processed dictionary of tuple to rule-set mapping, we can compute a factor that also gets included in defining the dirtiness confidence of a tuple. For each tuple, this factor can be the average of the Strength Factor of all the rules that will be applied to this tuple if this tuple gets picked to be cleaned. This factor can be appropriately combined with estimates from the other 2 heuristic steps to compute a composite score.

Here, we use the information from the ML Updated model to track the accuracy increase (which is our metric of defining 'better' as discussed previously) and information from the cleaner on the type and count of data cleaning done in the current iteration. Since the process of updating this strength factor is incremental and iterative, over a course of iterations, ideally, the Strength Factor of rules that affect the model more in terms of increasing the training accuracy get picked up fast and come to a saturation value giving way for other rules in order based on relative increase in accuracy coming from their end. Since, this is an ideal behavior in theory, we might see a slightly disproportionate result in practice. Nonetheless, incorporating this factor into account will surely help in identifying and increasing the probability of selection for certain dirty tuples in the sample that heavily use such rules. This is because it adds a positive factor in the dirtiness confidence evaluation - hence it will always be better even if the Strength Factor update is disproportionate in nature.

For every new iteration of sampling, using information from each of the heuristic steps (H) + (C) + (CR) updated at the end of the previous iteration, we can normalize the weighted scores received from each of the 3 steps and define an ordered ranking for the tuples to be sampled. To validate the intuition of this logic, we will analyse the efficiency of our proposed heuristic in the Experimentation section.

Dirtiness Confidence for each tuple T_i in R_{dirty} (weighted sum)

$$= \alpha_1[H]_i + \alpha_2[C]_i + \alpha_3[CR]_i$$

Further, Effective Sampling Probability for T_i in R_{dirty}:

$$= \frac{\alpha_1[H]_i + \alpha_2[C]_i + \alpha_3[CR]_i}{\sum_i \alpha_1[H]_i + \alpha_2[C]_i + \alpha_3[CR]_i}$$

Using a rank-ordering from this probability distribution, we can sample dirty data effectively, based on all the evidence aggregated in the iterative process. Sine the range of values for each of the values received from Steps (H), (C) and (CR) have a different range value, we need appropriate values of each α to take a balanced proportion from each of the 3 heuristic sub-steps. It would be better to experiment with this value and choose a domain-independent evaluation for the same, but due to time constraints we decided to move forward with a fixed value of $\alpha's$ based on approximate estimates.

The approach mentioned in Our Model is a novel extension to the prior work of 'ActiveClean'. In subsequent sections, we will draw a comparison between experimental evaluation of (i) Active Clean (Vanilla - No Optimizers), (ii) Active Clean with Optimizers, and (iii) Our Model based on the results obtained on training the UCI Adult Dataset as an ML-prediction task. It should be noted that the implementation for each has been coded by us from scratch.

IV. Dataset

We have worked on a dataset obtained from the UC-Irvine database collection called UCI Adult dataset. This database contains general information of adults including but not limited to age, sex, work class and education. A label corresponding to each individual denotes if they earn more than $50K a year or not. The dataset is used as a classification machine learning problem through which we predict if a particular person will earn more than $50k a year or less than $50k a year. The dataset is partially-dirty with multiple missing values. It is highly skewed with majority of the labels being of people earning less than $50k a year.

The database obtained from the website is divided into two parts namely a training dataset and a testing dataset. With training set having approximately 32k entries and test set having approximately 16k entries. After the database is obtained from the website, we combine the two sets and it is randomly shuffled. This shuffled dataset is divided into a training set of about 38k entries and a test set of 10k entries. The dataset has approximately 3800 missing values (denoted as '?') which we will be editing to '-999' and '-NA-' for numeral-based entities and string-based entities respectively. This allows us to maintain a constant format of the data.

The dataset has 15 attributes with column names including age, workclass, finalweight, education, occupation, race, sex,

capital-gain, capital-loss, hours-per-week, native-country etc. Some of these columns contain discrete values while the rest are continuous values (integers).

The discrete values for attributes have been defined by the contributors as:

Workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

Education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

Marital-status: Married-civ-spouse, Divorced, Never-married, Separated,Widowed, Married-spouseabsent, Married-AF-spouse.

Occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlerscleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protectiveserv, Armed-Forces.

Relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

Race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black. Sex: Female, Male.

Native-Country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI etc.), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad & Tobago, Peru, Hong, Holand-Netherlands.

The final attribute which denotes the earnings of a person is valued as $\leq$\$50k and $>$\$50k respectively. We will edit this particular attribute so that it is represented as -1 and 1, which correspond to class labels in our prediction task. This mapping will help us in the further steps to simplify calculations. In certain attributes we have added more domain values that can be assigned to that attribute. Occupation has an option called none. Also, we append another column to the database which provides serial numbers - to be treated as the Primary Key for this database. This will be helpful when we create a list of tuples which contain missing data so that they can be mapped to their position in the original database.

V. Data Cleaning: Technology and Approaches Used

Any real dataset is bound to have various errors and wrong data entries. Manually detecting these errors in large datasets is a very tedious job and is highly time consuming. This error We detect these errors using Conditional Function Dependencies (CFDs), Integrity Constrains (ICs) and Natural Language Processing(NLP) rules.

A. Integrity Constraints:

This method detects errors which violate the characteristic range of a particular attribute. For example, if we consider the current dataset, it has attributes such as age and hours-per-week. These represent the age a person and the number of hours per week that he works. Hence, neither of these can be negative. If an entry is negative, upon detection its sign will get reversed [refer reason in section VI]. Also we are checking for an upper limit for these two attributes i.e. the age of a person has to be < 200 and that the hours-per-week has to be $<= 168$ (One cannot work for more hours than the number of hours in a week). If we do find such an entry, we assume that as an error, value of 200 has been added to the age and we subtract 200 from the entry [refer reason in section VI]. A similar constraint has been added for Education level where the value cannot be negative.

B. Conditional Functional Dependencies:

Many of the attributes in the data set are related to each other and are interdependent. Thus using CFDs we can find if an entry is breaching this relation. For example, there are certain entries which mention the work class to be never worked. If that is the case, then the hours-per-week cannot be anything but 0. An actual tuple in the database can be listed as:

Never-worked, 157131, 11th, 7, Never-married, ?, Own-child, White, Female, 0, 0, **10**, United-States, $<= \$50K$

Above is an entry from the database which classifies the person in the never worked category while it shows that she works for 10 hours per week, which is clearly not possible. Thus, there are multiple such fundamental conditions which can be identified and used for data cleaning. Such instantiations include:

If workclass $==$ 'never worked'
  occupation='none' and hours per week = '0'
If workclass =='never worked'
  class='-1'
If workclass =='without pay'
  class='-1'
If workclass=='private'
  occupation!='armed forces'
If workclass=='self emp not inc'
  occupation!='armed forces'
If workclass=='self emp inc'
  occupation!='armed forces'

The following CFDs ensure that if a person is married, in his/her relationship column we need to make sure the entry is of a husband/wife.

If married=='married civ spouse'
  relationship='wife' or 'husband'(depending on gender)
If married=='married absent spouse'
  relationship='wife' or 'husband'(depending on gender)
If married=='married af spouse'
  relationship='wife' or 'husband'(depending on gender)

If a person has been widowed, never married or separated, his/her relationship cannot contain husband/wife.

If married=='widowed'
  relationship!='wife' or 'husband'
If married=='never married'
  relationship!='wife' or 'husband'

If married=='separated'
relationship!='wife' or 'husband'

A member with occupation as armed force cannot be anything but federal government.
If occupation=='armed forces'
workclass='federal gov'
If occupation=='priv house serv'
workclass='private'

If relationship=='wife' and gender=='male' and if marital status != 'widowed' or 'never married' or 'separated'
gender='female'
If relationship=='husband' and gender=='female' and if marital status != 'widowed' or 'never married' or 'separated'
gender='male'

Education for every entry should match its education-num according to the mapping as given below: Preschool - 1; 1st to 4th - 2; 5th 6th - 3; ..... bachelors - 13; masters - 14; prof school - 15; doctorate - 16

Every rule has been given a unique id, which gets saved if this rule is used in cleaning, so that we can use it at a later stage to calculate α_3.(as mentioned in section III)

C. Natural Language Processing:

We have used Bayes' Theorem to choose the most likely spelling correction for any given word. We try to find the correction c out of all possible corrections that maximizes the probability that c is the intended correction given the original word w:

$$argmax_{c \in candidates} P(c|w)$$

By Bayes' Theorem, this is equivalent to:

$$argmax_{c \in candidates} P(c)P(w|c)/P(w)$$

Since, $P(w)$ is same for every possible c:

$$argmax_{c \in candidates} P(c)P(w|c)$$

We have created a simple candidate model to find all the possible corrections. A simple edit to a word is a deletion (remove one letter), a transposition (swap two adjacent letters), a replacement (change one letter to another) or an insertion (add a letter). Our model considers one and two simple edits because these are the most likely mistakes that could happen. $P(c)$ is the language model which tells the probability with which a particular word (c) appears in the database. This probability can be estimated by counting the number of times each word appears. $P(w|c)$ is the probability that w would be typed in a text when the author meant c. We have defined a trivial model that says all known words of edit distance 1 are infinitely more probable than known words of edit distance 2, and infinitely less probable than a known word of edit distance 0.

We are also handling abbreviations using simple natural language processing techniques- substring matching, similarity testing and pattern recognition.

VI. Experiments and Analysis

To simulate the desired behaviour and validate the explained solution in this paper, we started by taking an existing dataset namely UCI Adult as described in Section IV and followed by writing the code from scratch in python. This dataset is already dirty in its raw format - with the presence of missing values (refer Section IV) and violation of some factual CFD's (refer Section V). Keeping this in mind, we take the following approach in building a complete sample of train and test data for our ML prediction task.

- Combine the entire provided train+test UCI Adult dataset, say D [size=48,842].
- Randomly shuffle D and then extract 10,000 entries which do not have any missing value for either of the attributes.
- Apply CFD's to these 10,000 entries so that we can assert an almost "manual" level of correctness for them and keep them aside as the un-seen, clean Test Data (Te). If we assume that the CFD's which we are using cover the entire set of all possible evaluations of linked attribute values for this database, we can actually assert perfect correctness of data. But given in our case, we have started with a smaller set of CFD's to validate our approach, we can still assume near "correctness" for our current application over the formulated CFD's.
- For the remainder of the data in D [size=38,842], keep it aside and mark it as Train Data (Tr).
- Even though the original dataset has real errors, to be able to correctly test and validate the functionality of our proposed model we intentionally add various types of errors in the Train data set and see it getting cleaned as the entire mechanism proceeds in iterations.

Apart from $\sim$3.8k number of tuples containing missing data in one or more attribute fields in Tr, we go ahead and artificially insert the following errors as per the defined quantity:

- Randomly delete attributes and replace them with either '-999-' or '-NA-' depending on if the attribute is string-based or integer-based for Max of 5% of the dataset.[size=194]
- Max of 10% of the total records for which the 'age' attribute value is not missing, are modified to either negate the age value or add 200 to it, randomly. [size=3,884]
- Max of 10% of the total records for which the 'total no. of hours worked' attribute value is not missing,are modified to either negate the total hours value or add 200 to it, randomly. [size=3,884]
- 3000 records out of all the records for which the 'Country' attribute value is not missing, are modified to be abbreviations if the original country label belongs in the set United States, India, Holand-Netherlands.
- Randomly change the race for Max of 20% of the tuples. [size=7,768]
- Randomly edit the 'education num' for Max of 20% of entries by giving them random values between -16 to 16 irrespective of the actually education level reached. [size=7,030]
- Max of 25% of the total records have their 'gender' changed to either male or female depending of their relationship status being either wife or husband respectively.

[size=3,351]
- Randomly change the work-class of Max of 20% of total records from federal gov to local gov, state gov etc.
- Max of 20% of the total records of which the ones with the marital status as married, we change the relationship status by randomly selecting it from a list containing own child, not in family and unmarried. [size=3,322]
- Max of 15% of the total records of which the tuples having marital status as widowed, never married and separated we change the relationship to either wife or husband randomly. [size=2,170]

Many of these insertions may have happened in the same tuple. The final count of dirty entries is almost 27k out of the total 38k entries.

Since, Tr has been made sufficiently dirty, we can now move along with the actual experimentation to try out the implemented solution mechanism. As mentioned in section III, the initial Tr is trained on a Logistic regression classifier to get the initial ML model weights and bias parameters. Missing data values for this case are imputed before the model training begins.

Having attained the original parameter values, we can start our iterative process of cleaning, model parameter update and further sampling of dirty data based on ML-model feedback, as described below: (i) Initialise R = Tr, R_{clean} = $\{\}$ and R_{dirty} = Tr. (ii) Now, start an iterative loop to facilitate batch data-cleaning and update until all samples have been chosen and looked at once.

- Based on the dirtiness confidence of tuples (as mentioned in section III), choose a batch size of dirty data sample to clean in each iteration, say = 500.
- Apply data cleaning techniques in a loop of data-correction for present values and data imputation for missing values on the given sample [using k-nearest neighbour approach], until any further change occurs. This step includes application of Integrity Constraints, Conditional Functional Dependencies (CFDs), NLP-based rules for correcting abbreviations, spelling corrections and other string-based patterns -which is defined in the Section V.
- After cleaning the current sample, the next step includes update of model parameters (θ_t) using a gradient-based approach found by combining of information from the cleaned sample, set of already clean data entries and the remaining dirty data component. This can be found in Section III as evaluations for g_s; g_c and the formula for model parameter update. The value of γ has been chosen to be 0.001 initially which decays over time.
- The confidence factor of each tuple being dirty gets updated depending on the fact if in the current iteration it was cleaned or no. If it was cleaned the confidence increases while if it wasn't then the confidence reduces. This confidence factor(CF) is composed of three components [refer section III]. To enlist them: i) Hashing factor(H): Handles duplicacy of dirty tuples ii) Average dirtiness confidence(C) of the tuple which was cleaned based on the individual confidence of its attribute values iii) Factor of contribution of data cleaning rules in cleaning that particular sample(CR).
- To calculate the final confidence of each tuple we take the weighted contribution of the three components mentioned above, giving us the confidence as:

$$CF = \alpha_1 * H + \alpha_2 * C + \alpha_3 * CR$$

The value of H is of the order of $\frac{1}{38000}$, while C lies between 0 and 10 and CR is a slowly increasing function. Thus we select the alphas such that none of the component is overshadowed by the others. Keeping this in mind we select values giving us $\alpha_1 = 0.5$,$\alpha_2 = 0.0025$ and $\alpha_3 = 0.04$.
- Using the sample from above, each of R,R_{clean} and R_{dirty} is updated as :

$$R_{dirty} = R_{dirty} - cleaned\,sample$$

$$R_{clean} = R_{clean} + cleaned\,sample$$

$$R = R_{dirty} + R_{clean}$$

And the next iteration continues.

ActiveClean [1] also proposes the idea of using a machine learning based estimator which would use ML models to predict the dirty data entries and pick only them to be put through the cleaner. We implemented this version of ActiveClean using Logistic Regression classifier as the estimator after analyzing its working as laid out in the paper. We used this model on the same data set and though the number of iterations did decrease and the model converged quicker than our final model, this method has a major drawback of "false positives". This method assumes that the prediction made by its model is absolutely correct and if it labels a tuple is clean, it is directly transferred to R_{clean} without any checks. This ML model is trained on the first sample(sample size = 500) after it has been cleaned using data cleaning rules, which is a very small size of a training data set. Thus if using this model, it predicts previously unseen dirty tuples as clean, it will never be able to identify similar instances in the future iterations also. Though it does improve to an extent with every iteration, at least in the initial stages it has not been trained so well, all the entries labelled as "clean" need not be actually clean. After we ran our code for this model, we noticed that out of about 27k dirty tuples, it could identify only 12k tuples, thus there were over 14k false positives. Since the data set we used is highly skewed as mention in Section IV, the labels ended up being correctly predicted which would not be the case if the data set was more evenly balanced, wherein our model would have performed better. Figure 3 shows the comparison of the number of dirty data sampled with increase in iteration for vanilla ActiveClean, ActiveClean with ML estimator and Our model. As we can see that our model first samples only the dirty data while in other two cases dirty data size is around ~ 350 (out of 500 sample size). So, our model converges faster than ActiveClean and is more accurate than ActiveCean with ML estimator (because of the false positives).

Table I shows the comparison of our model with vanilla ActiveCean [1], ActiveClean with ML estimator (extension) and two baselines described below.

- **Baseline1** $\rightarrow$ Obtained by running our model only on the clean part of R_{dirty} i.e. around 11k entries.

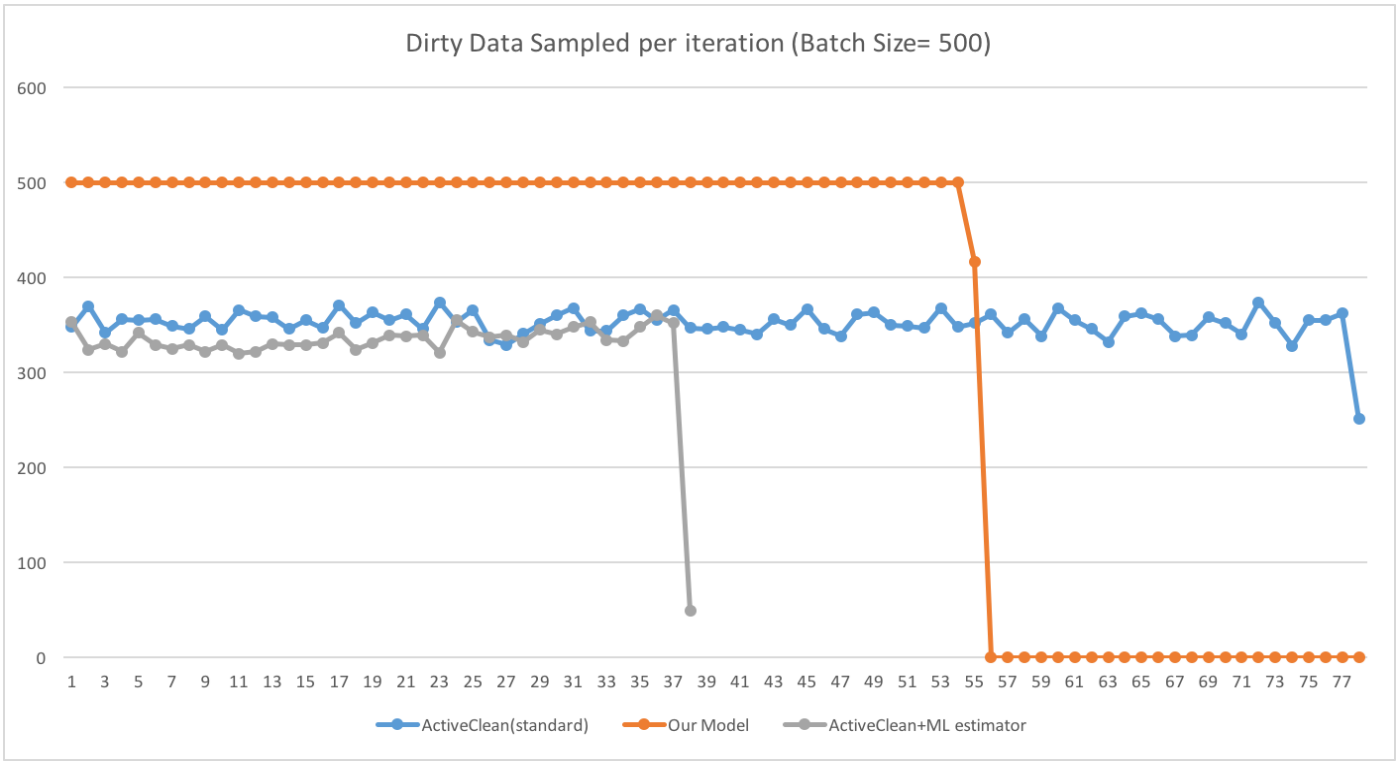


Fig. 3: Comparison of Dirty Data Sampled per iteration between ActiveClean, Our Model and ActiveClean with ML estimator.

TABLE I: Comparison of different models on accuracy

Model	Train Accuracy	Test Accuracy
ActiveClean	82.53 %	58.11 %
Baseline1	85.93 %	58.95 %
Baseline2	84.03 %	51.45 %
ActiveClean with ML estimator	82.63 %	57.95 %
Our Model	82.64 %	57.94 %

- **Baseline2** $\rightarrow$ Obtained by running our model on R_{dirty} and only the cleaned part obtained as R_{clean} on running the algorithm of our model, which is added to the original R_{dirty} as a whole.

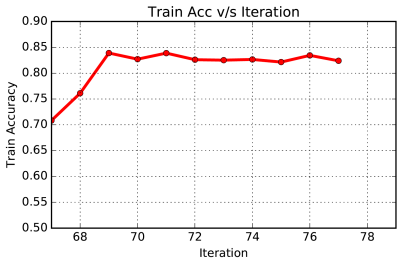


Fig. 4: Variation of training accuracy with iteration. The increase shows that our stochastic gradient update is working properly.

As can be seen in Figure 4, over the increase in iteration count, even though the increase in accuracy for the current case isnt much (maybe due to dirty data either not being sufficient in quantity to produce any major change or the data is too well-separated) or even too slow, we can still see some progress. This even though does not give us very good results but helps us in still validating our claim of performance increase.

VII. Conclusion

As discussed in this paper, ActiveClean is a strong step towards research being done at the intersection of Machine Learning and Database Technology. Even though the work is specifically presented for Machine Learning models with convex loss function, research work in this area is growing rapidly. With the initial aim to combat Simpson's paradox, ActiveClean excels in providing a theoretical standpoint in establishing why iterative cleaning and only parameter update is more relevant for such models than entire re-training or data sampling. We were able to validate the claims made by this existing work. Not only did we implement and validate the optimisations provided by the authors of ActiveClean on top of the basic framework, but were also successful in validating and formulating our own extension to their proposed model pipeline. Our proposed extension finds a balance between domain-dependent and domain-independent information in sampling intelligently by keeping track of a score to define 'Dirty confidence' for a data entity (tuple or value attribute). The results of the experimentation show promising performance for our model. Given the (skewed) nature of the dataset chosen, even though the results were comparable to the baseline methods and the variations of the actual ActiveClean implementation, our model showed its strength in sampling all dirty samples before any clean data tuple from the initial dirty dataset. Nonetheless,

there is abundant scope in refining our own implementation or even come up with other optimisations that can make the framework more flexible yet correct.

VIII. Future Work

Since the experimentation we tried to focus our evaluation on was skewed in terms of label proportion, we could not see much expected difference in the evaluation of results. It would be interesting to carry forward our approach and try it on a complete un-touched real-world dataset with proper balance of class distribution. Further, our model extensions rely heavily on using space to store information associated with Dirtiness Confidence Evaluation. Even though the size of this data will mostly be small given a specific problem dataset instance and number of cleaner rules, it would be interesting to figure out various compression or data encoding techniques to further reduce space complexity. Lastly, certain parameter values (like α's for dirty data confidence weighing, batch size for sampling etc.) in the experiments section have been defined by default by us based on general intuitive evaluation. It would be beneficial if we can come up with a framework that chooses for optimal values of these parameters.

Acknowledgment

We would like to thank our course Professor Alexandra Meliou, our course TA Keen Sung and all our classmates for constant feedback on our project deliverables, pushing us to improve our work and explore various novel ideas.

References

[1] S. Krishnan et al, *ActiveClean: Interactive data cleaning for statistical modeling*. PLVDB 9 (2016), no. 12, 948-959.

[2] Erhard Rahm and Hong Hai Do, *Data Cleaning : Problems and Current Approaches*, IEEE Data Eng. Bull., 2000.

[3] H. H. Shahri, *A Machine Learning Approach to Data Cleaning in Databases and Data Warehouses*, Progressive Methods in Data Warehousing and Business Intelligence: Concepts and Competitive Analytics, vol. 3 2009.

[4] Jiannan Wang, Sanjay Krishnan, et al, *A Sample-and-Clean Framework for Fast and Accurate Query Processing on Dirty Data*, SIGMOD, 2014.

[5] M. Yakout, L. Berti-Equille, and A. K. Elmagarmid, *Dont be scared: use scalable automatic repairing with maximal likelihood and bounded changes*, SIGMOD, 2013.

[6] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, *From data mining to knowledge discovery in databases*, AI magazine, 17(3):37, 1996.

[7] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, *Enterprise data analysis and visualization: An interview study*, TVCG, 2012.

[8] S. Krishnan, D. Haas, M. J. Franklin, and E. Wu, *Towards reliable interactive data cleaning: A user survey and recommendations*, HILDA, 2016.

[9] E. H. Simpson, *The interpretation of interaction in contingency tables*, Journal of the Royal Statistical Society. Series B (Methodological). JSTOR, 1951.

[10] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra, *Progressive approach to relational entity resolution*, VLDB, 2014.

[11] T. Papenbrock, A. Heise, and F. Naumann, *Progressive duplicate detection*, TKDE, 2015.

[12] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas, *Guided data repair*, VLDB, 2011.

[13] L. Bottou. Stochastic gradient descent tricks. In Neural Networks: Tricks of the Trade. 2012.

[14] A. Heise, G. Kasneci, and F. Naumann, *Estimating the number and sizes of fuzzy-duplicate clusters*, CIKM, 2014.

[15] M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan, *Query-oriented data cleaning with oracles*, SIGMOD, 2015

[16] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, *J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching*, SIGMOD, 2014.

[17] B. Mozafari, P. Sarkar, M. J. Franklin, M. I. Jordan, and S. Madden, *Scaling up crowd-sourcing to very large datasets: A case for active learning*, VLDB, 2014.