Training Models on Dirty Data Under Time Constraints

ABSTRACT

Databases are susceptible to various forms of corruption, or *dirtiness*, such as missing, incorrect, or inconsistent values. Increasingly, modern data analysis pipelines involve Machine Learning for predictive models. In this paper, we propose CleanML (CML), an anytime framework for training Machine Learning models with data cleaning. As we draw samples of our training examples as a part of stochastic optimization, we clean the data as required avoiding data that we have already cleaned or we believe is clean. The sampling is distribution is non-uniform prioritizing points that will likely affect the model significantly. The optimization iteratively converges giving us the desired anytime behavior. We index dirty and clean data to achieve to implement the framework efficiently. We evaluate CML on 4 real datasets and find that our methodology can return more accurate models for a smaller cost than alternatives such as uniform sampling and active learning.

1. INTRODUCTION

Databases are susceptible to various forms of corruption, or *dirtiness*, such as missing, incorrect, or inconsistent values. Numerous surveys of industry have shown that dirty data are prevelant [23], and there is now a growing industry around processing dirty data at scale [1]. Analysts are increasingly deriving data from inherently error-prone processes such as extracting structured data from websites, synthesizing data from multiple networked sensors, and linking entities in disparate data sources. As these data grow, new methodologies for scalable and reliable analysis in the presence of errors are required.

Increasingly, modern data analysis pipelines involve Machine Learning for predictive models. The endpoint of these pipelines can be any number of "data products", such as recommender systems, spam detectors, and forecasting models, all of which can be very sensitive to data quality [**?**]. When data error is systematic, or correlated with the data, errors can significantly bias predictions by a model. For example, in a recommender system, we may find that all users from one region have a missing age attribute. In this setting, discarding corrupted data or ignoring the problem can make predictions for the affected subpopulation untrustworthy. A more sophisticated approach is to apply robust statistical methods, but these are designed to mitigate the effect of high-magnitude random outliers, and not compensate for systematic errors.

For some types of systematic error, we can apply data cleaning, which is an extensively studied field (see Rahm and Do [21] for a survey). Instead of avoiding the problem, cleaning works by repairing (or approximately repairing) the corruption. However, cleaning large data can be expensive, both computationally and in human effort, as an analyst has to program repairs for all errors manifest in the data [15]. In some applications, scripted data transformations may not be reliable necessitating the use of even costlier crowdsourcing techniques [13,19].

An emerging solution to the growing costs of data cleaning is sampling [26] where the analyst cleans a small sample of data and can estimate the results of aggregate queries.Analysts can sample a large dataset, prototype changes on the sample, and evaluate whether these changes have the desired affect. The case for sampling is analagous to arguments for Approximate Query Processing (AQP) [5], where a timely approximate answer is more desirable than an exact slow answer. Sampling provides a flexible tradeoff between cleaning cost and query result accuracy for aggregate queries, but the question we explore in this paper is how this tradeoff extends to Machine Learning. Machine Learning is far more sensitive to sample size (training data) than aggregate queries and we explore how we can exploit the geometery of the model to prioritize sampling.

This insight is not surprising as the growing consensus in Machine Learning research is that all training data are not created equal and some data are more informative than others [10,22]. In Active Learning, we often sequentially select the most important points to label [22]. However, the key new challenge in data cleaning is that features and examples that may look unimportant in the dirty data may be important in the clean data and vice versa. We exploit two properties of this problem, sparsity and history, to build a priortization framework that identifies examples that are valuable to the clean model. First, errors are often happen in batches are often tightly clustered, that is they affect similar data. Second, as we clean more data, we learn how errors affect the data. We can use these two insights to partition data that we believe is clean from the dirty data, and estimate the effects of cleaning on the dirty data based on prior examples. The resulting algorithm selects examples valuable to the clean model with higher probability and as an analyst cleans data we further guide the cleaning.

In this paper, we propose CleanML (CML), an anytime framework for training Machine Learning models with data cleaning. CML supports a class of models called regularized-convex loss problems which includes linear regression, logistic regression, generalized linear models, and support vector machines. These models are typically solved with a stochas-

tic optimization technique such as Stochastic Gradient Descent (SGD). The basic idea of SGD is to draw a data point at random, calculate the gradient at that point, and then update a current best estimate with that gradient. This stochastic process fits nicely with the idea of sampling and cleaning. As we draw samples of our training examples, we clean the data as required by the optimization intelligently avoiding data that we have already cleaned or we believe is clean. Our main insight is that these samples can be drawn with importance sampling, which has recently been studied in stochastic optimization [27] but never in the context of online data cleaning. The optimization iteratively converges giving us the desired anytime behavior.

In summary, our contributions are

- We propose CleanML which given dirty data, a convex loss model, and a time budget, returns a model trained with respect to the clean data.
- CleanML is implemented with an non-uniform importance sampling approach that priortizes points in the clean model that are most imformative, and we provide analysis to identify the key tradeoffs.
- There are numerous database techniques that we apply to make this practical. We have to index dirty and clean data, amortize scan costs using shared cursors, and modify some data cleaning operations for correctness when applied to progressively increasing sample sizes.
- We evaluate CleanML on real and synthetic datasets to show that non-uniform sampling achieves improved performance in comparison to uniform sampling, and dirtiness-agnostic prioritization.

2. BACKGROUND

2.1 Motivating Example

To motivate our solution, we present a running example scenario inspired by one of our experimental datasets. In this problem, an analyst is training a classifier to predict the occurance of seizures based on EEG data.

EXAMPLE 1. *An analyst is given a 15-dimensional feature vector $x \in \mathbb{R}^{15}$ of electrical signals and a label $y \in \{0,1\}$ of whether the patient is having a seizure. The analyst trains an L1-Loss SVM on this data. After training, the analyst is informed of an inconsistency in the labels $y \in \{0,1\}$ where some patients unafflicted patients were marked as having seizures. However, the procedure to determine which patients' data are corrupted requires querying medical records.*

With existing tools, the analyst has a few options: (1) she can ignore the errors and assume that the inconsistencies do not affect the results, (2) she can discard the data and assume that the errors are not correlated with the other covariates, or (3) she can clean the entire dataset and retrain her model. In CML, we present the analyst with a middle ground where she can initialize the framework with her existing model and iteratively converge towards the clean model. To optimize the convergence rate, we feed information of each succesive model iteration back to prioritize which data to sample.

2.2 Machine Learning and Loss Minimization

The SVM model in our example is an instance of parametric Machine Learning. In parametric Machine Learning, the goal is to learn a set of model *parameters* θ from training examples. A common theoretical framework in Machine Learning is empirical risk minimization with linear predictors. We start with a set of training examples $\{(x_i, y_i)\}_{i=1}^N$ on which we minimze an loss function ϕ at each point parametrized that is parametrized by θ.

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \phi(x_i^T\theta, y_i)$$

For example, in a linear regression ϕ is:

$$\phi(x_i^T\theta, y_i) = \|\theta^T x_i - y_i\|_2^2$$

ϕ is often designed to be *convex*, essential meaning bowl-shaped, to make the training this model tractable. This class of problems includes all generalized linear models and support vector machines.

Typically, a *regularization* term $r(\theta)$ is added to this problem. The regularization term $r(\theta)$ is traditionally what is used to increase the robustness of the model. $r(\theta)$ penalizes high or low values of feature weights in θ to avoid overfitting to noise in the training examples.

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \phi(x_i^T\theta, y_i) + r(\theta)$$

For example, a popular variant of linear regression is called LASSO which is:

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \|\theta^T x_i - y_i\|_2^2 + \lambda \cdot \|\theta\|_1$$

By applying the L1 regularization term, if one of the features is particularly noisy, and does not add predictive value, it will get excluded.

Along these lines, other robust techniques have also been proposed. For example, in the case of linear regression, we can change the L_2 norm to an L_1 norm to mitigate the effect of outliers:

$$\phi(x_i^T\theta, y_i) = \|\theta^T x_i - y_i\|_1$$

The quadratic L2 loss implies that examples that deviate far from the typical example are quadratically penalized as opposed to linearly penalized with the L1 loss. There is a natural tradeoff between robustness and efficiency. The more robust a technique is, the less efficient it will be (i.e. estimate variance for a fixed number of training examples).

2.3 Systematic Biases

We often in the problem setting where our data is systematically biased. Consider an image classification task with incorrect labeling. If we train a model with respect to the incorrect labels, while we might have achieve a good out-of-sample accuracy on the incorrect labels, the classifications are incorrect in a semantic sense. Likewise, consider the case where we are predicting future product demand based on corrupted historical data. Training a model with respect to the corrupted data might have a low R^2 cross-validation error, but is incorrect at predicting the future trends. In such scenarios, we see data cleaning as complementary to robust statistics. Data cleaning gives us information about the "true" data distribution, which is highly beneficial when errors have systematic biases. Without cleaning, certain subpopulations of data might be frequently mispredicted.

2.4 SampleClean Project

In our previous work, we studied the relationship between approximate query processing, data cleaning, and sampling [16,26]. Traditionally, data cleaning has explored expensive, up-front cleaning of entire datasets for increased query accuracy. We proposed the SampleClean problem, in which an analyst cleans a small sample of data, and then estimates the result to an aggregate query e.g., `sum`, `count`, or `avg`. The main insight from the SampleClean project is that highly accurate answers for aggregate queries does not require cleaning the full dataset. Generalizing this insight, there is a deep relationship between the application (i.e., the query) and how an analyst should budget their effort in data cleaning. In fact, `avg` and `sum` queries are a special case of the convex loss minimization discussed in the previous section:

$$\phi = (x_i - \theta)^2$$

We then extended the SampleClean work to study cleaning Materialized Views [16]. Suppose base data is updated with insertions, deletions, or updates, we explored how we could efficiently propagate changes to a sample of the view instead of the full view. Subsequent queries on the view could be answered approximate.

The SampleClean problem inspired an eponymous system that implements sampling, data cleaning, and approximate query processing on the Apache Spark stack [4]. Also included in the Apache Spark stack are Machine Learning libraries including MLlib [3] and GraphX [2]. The in-memory architecture of the Apache Spark stack allows for increasingly interactive analysis [6,8]. Analysts can prototype data processing workflows on samples to evaluate performance before running expensive batch processing jobs on entire datasets. With data cleaning and machine learning libraries in the same software ecosystem, we see a new opportunity for joint optimization for interactive model building.

We see this work as an extension and generalization of the work that we did in the past. There are several new contributions. First, Machine Learning models give us a richer geometric information (e.g., gradients), which we can use prioritizing sampling. In this work, we explore non-uniform sampling as opposed to the uniform sampling methodologies studied before. Next, machine learning models are trained iteratively making them more amenable to adaptive processing where cleaned data can be fed back to inform the next set of samples. Finally, in this work, we give a deeper treatment to the problem of sparsity where only small clustered subsets of data are corrupted. We address this by using intelligent partitioning that segregates clean and dirty data.

2.5 Stochastic Gradient Descent

Sampling is a natural part of any Machine Learning workflow, as stochastic optimization is widely used to fit model parameters. The problems described in the previous subsections are often trained using a technique called Stochastic Gradient Descent (SGD) or one of its variants. The basic idea of SGD is to draw a data point at random, calculate the gradient at that point, and then update a current best estimate with that gradient.

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma \nabla \phi(x_i^T \theta, y_i)$$

SGD can also be applied in a "mini-batch" mode, where we draw a subset of data $S^{(t)}$ at random and update with the average gradient.

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \frac{\gamma}{\|S^{(t)}\|} \sum_{i \in S^{(t)}} \nabla \phi(x_i^T \theta, y_i)$$

We can use this workflow for designing an anytime data cleaning methodology. As data is sampled, we can clean the samples. The analyst then can stop at anytime and use the best model at that instant. SGD and its variants are well-studied and there are lower-bounds on the convergence rates using these techniques. Recently, a number of works have explored non-uniform sampling distributions for stochastic optimization [20,27]. The main insight is that non-uniform distributions may on average estimate the gradient accurately. In this work, we explore how to design such a non-uniform distribution for iterative data cleaning.

3. SYSTEM ARCHITECTURE

3.1 Data Cleaning Model

We define data cleaning as transformations that happens to the base data that result in changes to the feature vector. We do not consider data cleaning operations that change the number of rows in the relation such as row deletion or row deduplication. Allowed operations must result in the following changes to the feature vector: feature transformation and feature addition. Therefore, there is at most a one-to-one relationship beftween features in the cleaned data and the dirty data.

3.2 Problems

We believe that interactive model building problem is best addressed by an anytime approach, where an analyst has a time budget and wants a best effort result given this time budget. CML optimizes the convergence of the model by adaptively changing the sampling distribution. CML is an optimizer that wraps around the SampleClean data cleaning library and Machine Learning libraries in Spark.

Problem 1 (Importance Sampling).

Problem 2 (Model Processing with Predicates).

Problem 3 (General Model Processing).

3.3 System Architecture

In Figure 1, we illustrate our system archictecture in contrast to a traditional data cleaning architecture. Traditionally, data cleaning has explored expensive, up-front cleaning of entire datasets for increased accuracy on all possible queries. Raw data is cleaned, then featurized, and then a model is trained based on the clean data. In contrast, CML explores a different architecture where data is progressively processed. When we know the desired model in advance, we can feed this information back to guide and priortize data cleaning. In the process of fitting the model, we can integrate data cleaning rather than divorcing the two parts.

3.4 Parameter Updates

To achieve this architecture, CML needs to maintain two states the parameter vector θ and the sampling distribution P. We will formally describe the mathematics behind this process in the later sections. In Figure 2, we illustrate the structure of the parameter updates. At each iteration, we draw a batch of dirty data using the sampling distribution P. We calculate the average gradient for this data, and combine this gradient with the average gradient of the cleaned data. We then update the parameter θ which in turn updates the sampling distribution.

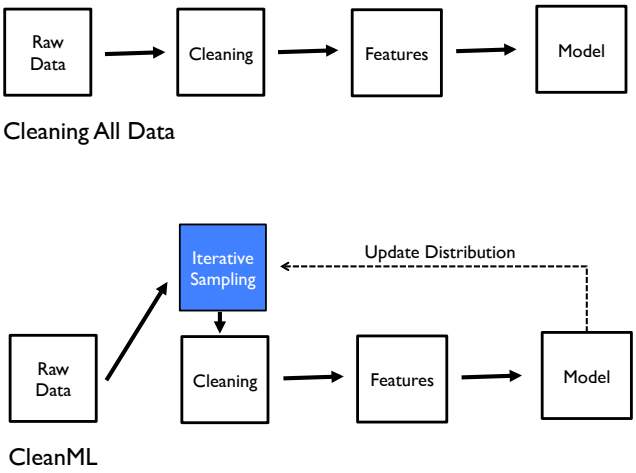

Figure 1: CleanML (CML) is an anytime framework from training models on dirty data. Expensive data transformations prior to model training are budgeted with a non-uniform sampling that priortizes examples that are most likely to change the model. As more data is cleaned the sampling distribution is updated.

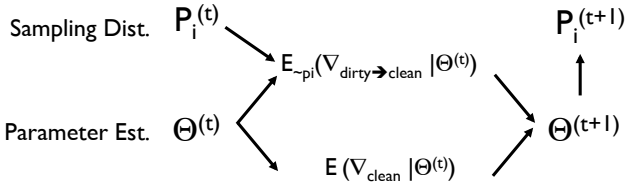

Figure 2: The CleanML operation DAG. At each iteration, we partition the data into clean and unknown quality. We fuse gradient estimates from the two partitions: (1) a sample from the unknown data using importance sampling, (2) a full (non-stochastic) gradient step on the clean data. After this estimate, we update the sampling distribution and the parameter.

4. IMPORTANCE SAMPLING AND SGD

In this section, we will describe the main algorithmic contribution of this paper. We will start by introducing the anytime sampling problem as Stochastic Gradient Descent problem. We will start by presenting the intuition in an artifical oracular setting, then will complicate the story by discussing what we have to do to make this practical.

4.1 Basic Algorithm: Oracular Case

Mini-batch stochastic gradient descent is an algorithm for finding the optimal value of θ, given the convex loss ϕ, and data. At each iteration, we draw a subset of data at random and update with the average gradient of that subset of data.

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma \sum_{i \in S^{(t)}} \nabla \phi(x_i, y_i, \theta^{(t)})$$

γ is a hyperparameter which is called the learning rate. There is extensive literature in machine learning for choosing γ appropriately, and in this work, we will assume that it is set by the system.

We treat the anytime sampling problem as a stochastic gradient descent problem. We start with an initialization which is the dirty model, the model trained completely on the dirty data. Then, we sample a mini-batch of data on which we apply our expensive data cleaning operations. We stop when the budget is reached. This basic algorithm has performance lower bound of:

$$\mathbb{E}(\|\theta^{(t)} - \theta^*\|^2) \leq O(\frac{\sigma^2}{bt})$$

and for the non-strongly convex case {{**explain better**}}

$$\mathbb{E}(\|\theta^{(t)} - \theta^*\|^2) \leq O(\frac{\sigma^2}{\sqrt{bt}})$$

where $\mathbb{E}(\|\nabla \phi\|_2^2) = \sigma^2$.

In the data cleaning setting, where the data cleaning is often far more expensive than calculating a gradient for each point (e.g., Entity Resolution), we can consider non-uniform sampling techniques that require some pre-processing of the data. If we choose our sampling technique carefuly, we can reduce the σ^2 term in the error bound. The technique that we will apply is called importance sampling. Importance sampling is a technique that allows us to draw a batch from a non-uniform distribution but reweight the result to give us an unbiased estimate of the gradient. The gradient update becomes:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma \sum_{i \in S^{(t)}} \frac{1}{p_i} \nabla \phi(x_i, y_i, \theta^{(t)})$$

It is established [27] that the optimal sampling distribution (lowest variance) is:

$$p_i \propto \|\nabla \phi(x_i, y_i, \theta^{(t)})\|$$

In the standard Machine Learning setting, it is not practical to use this optimal sampling distribution. Determining this sampling distribution requires calculate the gradient at every data point which defeats the purpose of stochastic optimization. However, in our setting, we are actually calculating the gradient of the cleaned data:

$$\phi(C(x_i), C(y_i), \theta^{(t)})$$

Part of the motivation of this work is that the function C is often expensive to evaluate. Instead, suppose we had an estimate of $\hat{C}$ that was cheap to evaluate. We could then apply this estimate to every point and get an approximately optimal sampling distribution:

$$p_i \propto \|\nabla \phi(\hat{C}(x_i), \hat{C}(y_i), \theta^{(t)})\|$$

This builds the inutition for the basic algorithm that we will propose. As we clean data, we learn an estimator $\hat{C}$ which can be used to calculate the approximately optimal sampling distribution.

1. Initialize with $\theta^{(0)}$ as the dirty model.
2. For rounds i=1...T
   (a) Update $\hat{C}$
   (b) Sample $\frac{B}{T}$ data points with probabilites as described above
   (c) Apply data cleaning to the sample of data, or if already cleaned skip
   (d) Calculate the average gradient of the sample
   (e) Apply weighted gradient descent
3. Return $\theta^{(T)}$

We have been purposefully vague about how to learn the estimator $\hat{C}$ to simplify the intuition and analysis of the

algorithm. In the subsequent subsections, we will describe different settings and how we can efficiently learn $\hat{C}$. Essentially, all variants of the basic algorithm will modify step 2 in differing ways.

4.1.1 Oracular Case: Analysis

Suppose, our estimator $\hat{C}$ is perfect where $\hat{C} = C$ for all input data. In this case, we can show that the error bound for the importance sampled algorithm is strictly lower than for the non-importance sampled algorithm.

THEOREM 1. *The importance sampled algorithm converges to the same solution as the uniformly sampled algorithm. Furthermore, $\sigma_{imp} \leq \sigma_{uni}$. The inequality is tight only when all the data is identical.*

PROOF. The importance sampling gives an unbiased estimate of the gradient at each iteration so it converges to the same solution. {{**Probably cite TR for proof**}} □

We know that learning a perfect estimator for C is equivalent to cleaning all of the data and is not feasible. We next analyze how inaccurate $\hat{C}$ and still preserve the gains of importance sampling.

THEOREM 2. *Let the importance sampling distribution be perturbed by ϵ_i at each data point:*

$$\|p_i \propto \nabla\phi(\hat{C}(x_i), \hat{C}(y_i), \theta^{(t)}) + \epsilon_i\|$$

If ϵ_i is unbiased (averages 0 over all data points) and varies less than gradient:

$$\mathbb{E}(\|\epsilon\|^2) \leq \mathbb{E}(\|\nabla\|^2)$$

then, $\sigma_{imp} \leq \sigma_{uni}$.

PROOF. {{**Probably cite TR for proof**}} □

From this analysis, we see that the allowed error in $\hat{C}$ can be quite large. As long as on average we estimate the right result, we can make mistakes on the order of the gradient's norm. We will use this insight to design a cheap coarse grained estimator than relies on error classification (i.e. is the point erroneous) rather than trying to learn a complicated function that predicts the clean value.

4.2 Error Predicate Case

If errors are sparse, then the previous algorithm still wastes a lot of computation. For example, for most of the data points:

$$\nabla\phi(x_i, y_i, \theta) = \nabla\phi(C(x_i), C(y_i), \theta)$$

Essentialy, we are computing the gradient at many points and then throwing that computation away when we sample. We now explore how we can use this previous computation to further reduce the variance σ^2.

In Section **??**, we described the basic API and user interface of our system. An analyst specifies multiple cleaning operations and then a model. Suppose, we decomposed the function C into its consituent cleaning operations C_i^k. To describe the parse errors, for each C_i there is some predicate ρ_i that selects the affected records. The next case that we analyze is when we know the perfectly ρ_i in advance. Unlike the oracular case, there are some real-world situations where this true. For example, if data is missing e.g., field misalignment it is often clear which rows are affected and which are not. Similarly, in the entire field of constraint-based data cleaning [12], we know in advance which records have constraint violations.

These predicates also simplify the construction of the estimator $\hat{C}$. For a tuple (x_j, y_j), there are basically two cases:

$$(x_j, y_j) = \begin{cases} (x_j, y_j) & \rho_i \text{ is false} \\ (C_i(x_j), C_i(y_j)) & \rho_i \text{ is true} \end{cases}$$

We can calculate the average change over already cleaned data that satisfy the predicate[1]:

$$\Delta_i = \sum_{j \in \rho(cleaned)} (C_i(x_j), C_i(y_j)) - (x_j, y_j)$$

the resulting estimator is:

$$\hat{C}_i = \begin{cases} (x_j, y_j) & \rho_i \text{ is false} \\ (x_j, y_j) + \Delta_i & \rho_i \text{ is true} \end{cases}$$

The oracular algorithm gets modified as follows:

1. Initialize all $\Delta = 0$
2. For rounds i=1...T
   (a) For each cleaning op j=1...K
      i. For all points that do not satisfy ρ_j, calculate the average gradient and call this result g_1.
      ii. For all points that do satisfy ρ_j, sample $\frac{B}{T}$ data points using $\hat{C}_j$ described above.
      iii. Apply data cleaning to the sample of data, or if already cleaned skip.
      iv. For all newly cleaned points update Δ_j
      v. Calculate weighted gradient of the sample points call this g_2.
      vi. For N_1 points that do not satisfy the predicate and N_2 points that do, apply gradient descent using the gradient $\frac{N_1 g_1 + N_2 g_2}{N_1 + N_2}$.
3. Return $\theta^{(T)}$

To actually implement this algorithm efficiently there are physical design considerations that we will discuss in the next section.

4.2.1 Error Predicate Case: Analysis

Using the error predicates can significantly increase the accuracy when errors are sparse. In fact, this saving is quadratic in the sparsity $O(sparsity^2)$.

THEOREM 3. *The error-predicate algorithm converges to the correct answer. Furthermore, suppose there are N_1 erroneous points out of N total points.*

$$\sigma_{ep}^2 = \frac{N_1^2 \sigma_{imp}^2}{N^2}$$

PROOF. Gradient is unbiased at each step. {{**Probably cite TR for proof**}} □

THEOREM 4. *Δ_i error satisfies variance conditions under NormalizedSC like conditions.* {{**Do the analysis**}} {{**Also number of rounds before this happens**}}

PROOF. {{**Probably cite TR for proof**}} □

4.3 General Case

Finally, in the general case, we do not know the error predicate in advance. Consider a crowd-based cleaning technique whose effects are hard to predict a priori. In this case, we have to continuously update a classifier to partition the data

[1]For added features, we create dummy feature vectors in the dirty data

into clean and dirty. The challenge is that this potentially biases our result if our classifier is not perfect. Some dirty points may be classified as clean. The quadratic savings seen in the predicate case can often outweigh small biases due to classification error.

Like before, suppose we decomposed the function C into its consituent cleaning operations C_i^k. For each cleaning operation, we have a classifier, e.g., Support Vector Machine, that learns which points are dirty and which points are cleaned based on that operation. To account for potential bias in the classifier, we allow the analyst to specify the design point in the bias variance tradeoff. Many types of classifiers allow users to tradeoff precision and recall. For an SVM, we may only classify a point as clean if it is sufficiently far from the margin. Or for Logistic Regression, we may do so if its class likelihood is over 80%. In our experiments, we use SVMs and the margin distance provides a flexible way to tradeoff potential bias and gradient variance.

Adding the classifier also adds a complex depedence between batch size and performance. Initially, the classifier is likely very inaccurate due to the lack of training examples. As we clean more data, we can learn what is clean and what is dirty. Accordingly, we can delay the classifier's introduction by l iterations until such time it is accurate.

1. Initialize all $\Delta = 0$
2. For rounds i=1...T
   (a) For each cleaning op j=1...K
      i. If $i \geq l$, apply classifier to partition dirty and cleaned data.
      ii. Follow error predicate algorithm steps ii-v
      iii. For N_1 "clean" points and N_2 "dirty" points, apply gradient descent using the gradient $\frac{N_1 g_1 + N_2 g_2}{N_1 + N_2}$
      iv. Update classifier with newly cleaned data.
3. Return $\theta^{(T)}$

The general case algorithm is harder to analyze and thus we rely on empirical performance on real data.

5. PROGRESSIVE DATA CLEANING

In the previous section, we described our algorithmic insight into this problem. The algorithm treats the data cleaning as a black-box that materializes the clean value, however, to make such a system work there are numerous data cleaning considerations.

5.1 Incremental Application of Cleaning

There are some cases in which cleaning a row of data may require considering the other data in the sample. Consider entity resolution, where we have to resolve inconsistent representations of an attribute. When cleaning a sample of data this requires considering all of the inconsistent representations in the sample, and then resolving them to a canonical representation. However, such operations will fail in the iterative batch mode proposed by CML.

To avoid this issue, we define the data cleaning operator semantics in the following way. The data cleaning operation is applied to all previously cleaned points. At each iteration CML samples more data, and we have to define our data cleaning operations to incrementally update to the larger sample (including previously cleaned data). For the entity resolution operator, we define incremental maintenance procedures and fall-back to recomputation if there are user-defined operations.

Entity Resolution Operator: The entity resolution operator can be incrementally maintained by joining the new subsample with cleaned sample and taking the transitive closure. This can be implemented efficiently using a UNION-FIND data structure.

5.2 Incremental Re-train

As we increase the sample size, the already cleaned rows might require updating as well. Consider the entity resolution operator, which might find a better canonical representation in the new sample.

{{TODO}}

5.3 Physical Design

To make incremental cleaning feasible, there are also indexing/partitioning considerations. We need to quickly determine which rows are dirty and which have been previously cleaned.

{{TODO}}

5.4 Choosing Error-Predicate vs. General Case

Based on the cleaning operator, we can decide whether the operation is in the Error-Predicate Case or General Case. If we are in the error-predicate case, then we can build a predicate index to determine which records are clean and dirty efficiently. {{TODO}}

6. EXPERIMENTS

{{Systematic Error-Sort most valuable feature, remove set x-percent to mean value of remaining features}}

{{Random Error-In Most valuable feature, randomly set x-percent to value of highest feature}}

6.1 Experiment 1. Data Cleaning vs. Alternatives

{{Full dataset, cleaning, no cleaning, robust, discard}} Basic point, if errors are systematic cleaning is beneficial, robust, discard etc. are ok if things are random. Problem is you don't know where you are a priori.

6.2 Experiment 2. Sampling + Data Cleaning vs. Fixed

{{No Cleaning vs. SampleClean vs. CML vs. Active Learning}}

6.3 Experiment 3. Sampling + Data Cleaning vs. Error Rate/Selectivity

6.4 Experiment 4. Overhead of Scan

{{Do a real experiment}}

6.5 Experiment 5. Predicate vs. General Case

{{Do a real experiment}}

6.6 Experiment 6. End-to-end

{{MNIST}}

{{ER results}}

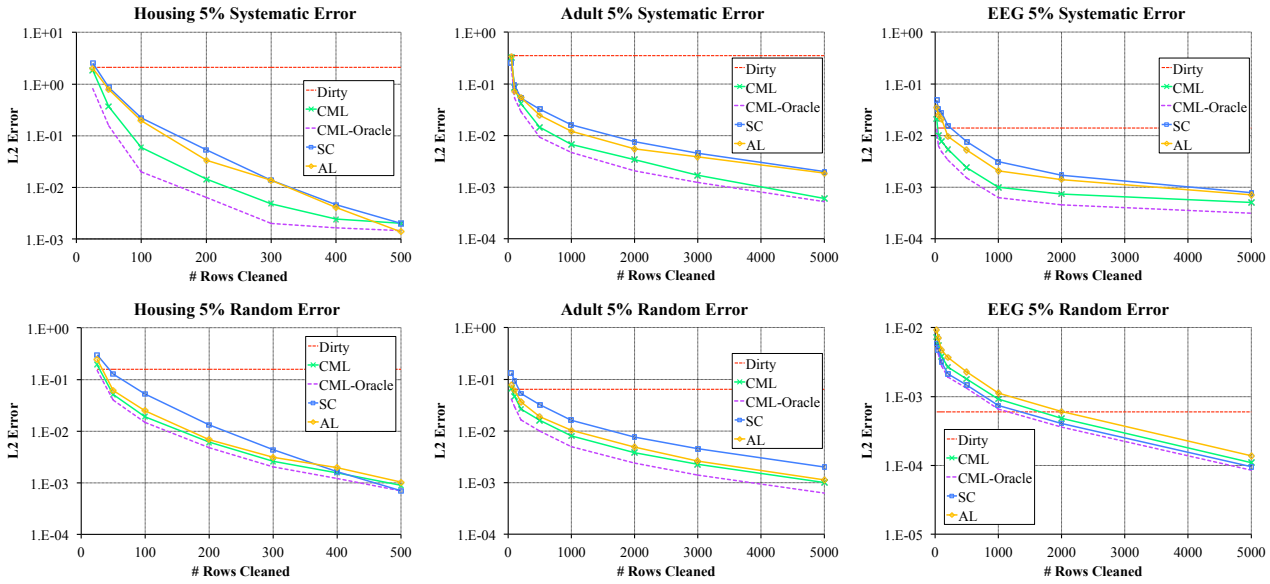

Figure 4: TODO

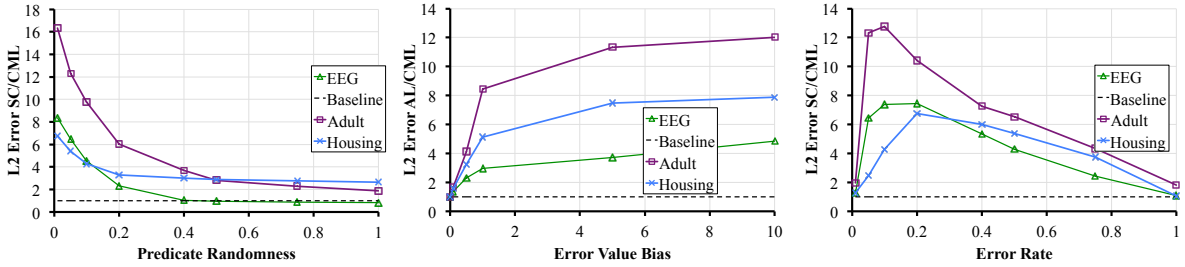

Figure 5: TODO

7. RELATED WORK

Bringing together data cleaning and machine learning presents us with several exciting new research opportunities that incorprates results from both communities. We highlight some of the key relevant work in this field and how this relates to our proposal.

Stochastic Optimization: Zhao and Tong recently proposed using importance sampling in conjunction with stochastic gradient descent [27]. However, calculating the optimal importance sampling distribution is very expensive and is only justified in our case because data cleaning is even more expensive. Zhao and Tong use an approximation to work around this problem. This work is one of many in an emerging conensus in stochastic optimization that not all data are equal (e.g., [20]). This line of work builds on prior results in linear algebra that show that some matrix columns are more informative than others [**?**], and Active Learning which shows that some labels are more informative that others [22].

Active vs. Transfer Learning: While it is natural to draw the connection between CML and Active Learning, which is widely used in data cleaning, they differ in a few crucial ways. Active Learning largely studies the problem of label acquisition [22]. This can be seen as a narrower problem setting than our problem (missing data in the label attribute), and in fact, our proposed approach can be viewed as an Active Learning algorithm. CML has a stronger link to a field called Transfer Learning [18]. The basic idea of Transfer Learning is that suppose a model is trained on a dataset D but tested on a dataset D'. In transfer learning, the model is often weighted or transformed in such a way that it can still predict on D'. Transfer Learning has not considered the data cleaning setting, in which there is a bijective map between $D \mapsto D'$ that is expensive to compute. Much of the complexity and contribution of our work comes from efficiently cleaning the data.

Secure Learning: Another relevant line of work is the work in private machine learning [11,25]. Learning is performed on a noisy variant of the data which mitigates privacy concerns. The goal is to extrapolate the insights from the noisy data to the hidden real data. Our results are applicable in this setting in the following way. Imagine, we were allowed to query k true data points from the real data, which points are the most valuable to query. This is also related work in adversarial learning [17], where the goal is to make models robust to adversarial data manipulation.

Data Cleaning: There are also several recent results in data cleaning that we would like to highlight. Altowim et al. proposed a framework for progressive entity resolution [7]. As in our work, this work studies the tradeoff between resolution cost and result accuracy. This work presents many

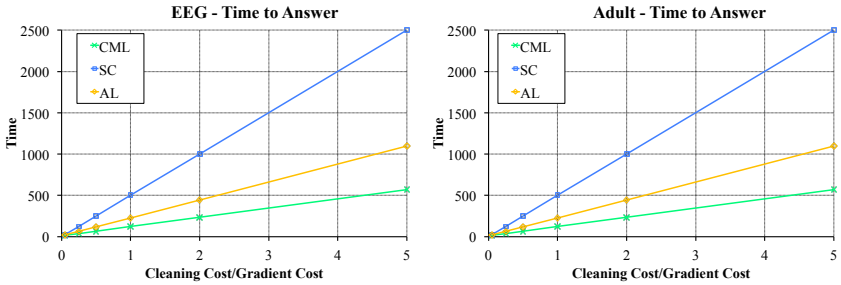

Figure 6: TODO

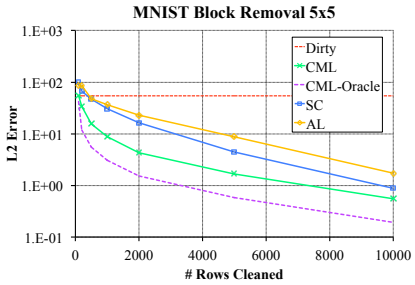

Figure 7: TODO

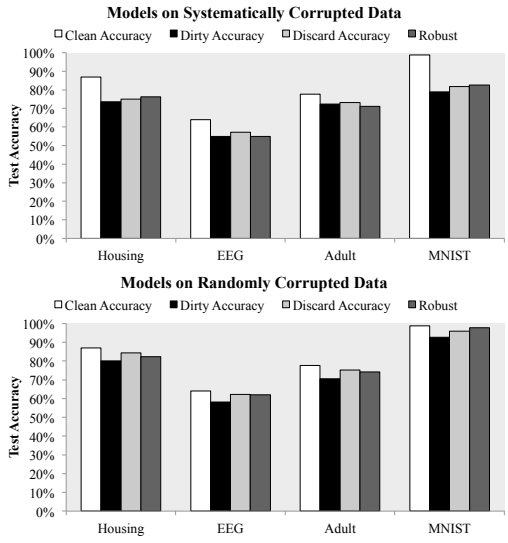

Figure 3: TODO

important ideas on which we build in our paper: (1) it recognizes that ER is expensive and some operations are more valuable than others, and (2) anytime behavior is desirable. We take these ideas one step further where we pushdown the model at the end of the pipeline to data cleaning and choose data that is most valuable to the model. Volkovs et al. explored a topic called progressive data cleaning [24]. They looked at maintaining constraint-based data cleaning rules as base data changes. Many of the database tricks employed including indexing and incremental maintenance were valuable insights for our work. Bergman et al. explore the problem of query-oriented data cleaning [9]. Given a query they clean data relevant to that query. Bergman et al. does not explore aggregates or the Machine Learning models studied in this work.

8. CONCLUSION

In this paper, we propose CleanML (CML), an anytime framework for training Machine Learning models with data cleaning. We implement CML on Apache Spark to jointly optimize cleaning and modeling pipelines. CML uses an importance sampling-based stochastic gradient descent framework to priortize data cleaning. We evaluate CleanML on real and synthetic datasets to show that non-uniform sampling achieves improved performance in comparison to uniform sampling, and dirtiness-agnostic prioritization. {{TODO}}

CML is only a first step in a larger integration of data analytics and data aquisition/cleaning. There are several exciting, new avenues for future work. First, in this work, we largely study how knowing the Machine Learning model can optimize data cleaning. We also believe that the reverse is true, knowing the data cleaning operations and the featurization can optimize model training. For example, applying Entity Resolution to one-hot encoded features results in a linear transformation of the feature space. For some types of Machine Learning models, we may be able to avoid retraining. {{**Sounds confusing rewrite shameless self citation ;)**}} The optimizations described in CML are not only restricted to stochastic gradient descent algorithms. We believe we can extend a variant of SDCA (Stochastic Dual Coordinate Ascent) [14] to extend this technique to kernelized methods.

9. REFERENCES

[1] Big data's dirty problem.
[2] Graphx. `https://spark.apache.org/graphx/`.
[3] Mllib. `https://spark.apache.org/mllib/`.

[4] Sampleclean. http://sampleclean.org/, 2015.

[5] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

[6] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

[7] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *Proceedings of the VLDB Endowment*, 7(11), 2014.

[8] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. SIGMOD, 2015.

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

[10] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

[11] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 429–438. IEEE, 2013.

[12] A. Ebaid, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiane-Ruiz, N. Tang, and S. Yin. Nadeef: A generalized data cleaning system. *VLDB*, 2013.

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

[14] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

[15] S. Kandel, A. Paepcke, J. Hellerstein, and H. Jeffrey. Enterprise data analysis and visualization: An interview study. *VAST*, 2012.

[16] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. `http://www.ocf.berkeley.edu/~sanjayk/pubs/svc-2014.pdf`, 2014.

[17] B. Nelson, B. I. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. Tygar. Query strategies for evading convex-inducing classifiers. *The Journal of Machine Learning Research*, 13(1):1293–1332, 2012.

[18] S. J. Pan and Q. Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[19] H. Park and J. Widom. Crowdfill: Collecting structured data from the crowd. In *SIGMOD*, 2014.

[20] Z. Qu, P. Richtárik, and T. Zhang. Randomized dual coordinate ascent with arbitrary sampling. *arXiv preprint arXiv:1411.5873*, 2014.

[21] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[22] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[23] N. Swartz. Gartner warns firms of 'dirty data'. *Information Management Journal*, 41(3), 2007.

[24] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 244–255. IEEE, 2014.

[25] M. J. Wainwright, M. I. Jordan, and J. C. Duchi. Privacy aware learning. In *Advances in Neural Information Processing Systems*, pages 1430–1438, 2012.

[26] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD Conference*, pages 469–480, 2014.

[27] P. Zhao and T. Zhang. Stochastic optimization with importance sampling. *arXiv preprint arXiv:1401.2753*, 2014.