ActiveClean: Iteratively Improving Convex-Loss Models With Data Cleaning

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 which can be sensitive to dirty data. Dirty data is often expensive to repair, and naive sampling solutions are not suited for training high dimensional models. In this paper, we propose ActiveClean, an anytime framework for training Machine Learning models with budgeted data cleaning. Our framework updates a model iteratively as small samples of data are cleaned, and includes numerous optimizations such as importance weighting and dirty data detection. We evaluate ActiveClean 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 industrial surveys have shown that dirty data are prevalent [27], and there is now a growing industry around cleaning data at scale [2]. Recent research and commercial interest is driven by increasing analytics from inherently error-prone processes such as extracting structured data from websites and synthesizing data from noisy sensors. New methodologies for scalable and reliable analysis in the presence of errors are required.

Data analysis pipelines are also growing in complexity. 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 [33]. Increasingly, analytics stacks are moving beyond traditional SQL analytics and natively supporting Machine Learning (e.g. Berkeley Data Analytics Stack [1], Hadoop, Stratosphere [4]). Predictions made by Machine Learning models can be significantly affected by systematic dirtiness i.e., corruption that is correlated with the data. Consider a music recommender system, we may find that due to a software bug all users from Europe have an incorrect age attribute defaulted to "18-24". A recommendation model trained on this data may spuriously "learn" a correlation relationship between the "18-24" age group and music liked by European users. While there is an extensive literature on robust Machine Learning, this work largely focuses on resilience to atypical outliers (i.e., age of "1000") and not systematically corrupted data.

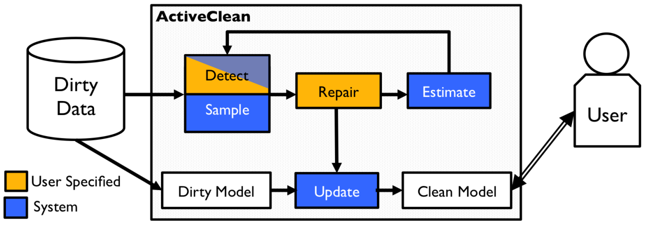

Figure 1: ActiveClean presents an architecture where data cleaning is integrated with model training. The user specifies a data repair method, and we provide a framework for sampling, model update, and feedback through estimation.

Since new analytics frameworks encapsulate the entire data processing pipeline from raw data to features to model training, there is an opportunity to address this problem from an end-to-end data management perspective. The database community's response to systematic corruption is data cleaning, which is an extensively studied field (see Rahm and Do [25] for a survey). Cleaning works by repairing (or approximately repairing) the corruption in the base data. However, data cleaning can be a very time consuming process. Programming data transformations to fix all problems manifest in the data requires substantial developer effort [17]. As errors increase in complexity, scripted transformations and automated rules may not suffice. Works such as Corleone [14] and CrowdFill [23] address this problem by using crowdsourcing. While more it is more accurate, even moderately sized datasets are very challenging to clean with the crowd.

An emerging solution to the growing costs of data cleaning is sampling as proposed in the SampleClean work [32]. The analyst cleans a small sample of $k \ll N$ records and can estimate the results of aggregate queries based on cleaning this sample. Adjusting the sample size k provides a flexible tradeoff between cleaning cost and query result accuracy for aggregate queries. The case for sampling in data cleaning is analogous to arguments for Approximate Query Processing (AQP) [3], where a timely approximate answer is more desirable than an exact slow answer.

Unfortunately, a naive application of sampling will not work well for the high-dimensional models learned in Machine Learning. These applications are far more sensitive to sample size (training data) than aggregate queries. Machine Learning applications can require a large amount of training data to learn correlations between possibly hundreds of features and labels. A small sample of clean data may not be able to support a viable Machine Learning model. The predictions from this model may be inaccurate and the problems introduced by the small sample size may dominate any gains from data cleaning.

In this paper, we propose ActiveClean, an anytime framework for training Machine Learning models with budgeted data cleaning (Figure 1) that iteratively corrects a dirty model rather than retraining it on a sample of clean data. ActiveClean is initialized with a dirty model and users specify a repair method. Our system will provide a framework to return an accurate model given a repair budget. ActiveClean selects a sample of data, applies the repairs, then updates the model. After model update, the repairs are fed back to adaptively select the next sample of data. We can optionally integrate user specified dirty data detection rules (such as constraints) into the framework to further narrow our sampling. ActiveClean supports a broad class of models, called convex-loss models, which includes linear regression, logistic regression, generalized linear models, and support vector machines.

ActiveClean presents a novel framework that tightly integrates data cleaning with model training. As a result, there are numerous new opportunities for optimization. We can "push down" the model training to the data cleaning and select data to clean that are most valuable to the model. We can also "push up" the data cleaning to the model training by intelligently batching together model updates from already cleaned data. In summary, our contributions are

- We propose ActiveClean which given dirty data, a model, a data cleaning procedure, and a budget, we can return a highly accurate model for a fraction of the cleaning cost.
- A gradient-based incremental model correction technique, that uses cleaned data to update a dirty model.
- A linearization-based estimation technique that estimates which data are most valuable to the model to guide data sampling.
- Integration with existing dirty data detection techniques for estimation, update, and sampling.
- We evaluate ActiveClean on real and synthetic datasets to show that non-uniform sampling achieves improved performance in comparison to uniform sampling, and Active Learning.

2. PROBLEM SETUP

2.1 Motivating Scenario

We introduce the concepts in this paper with a running example scenario that is inspired by our experimental dataset. A medical analyst is interested in understanding the correlation between neurological data (EEG data), patient predispositions, and the onset of seizures.

Example 1. *An analyst is given a 15-dimensional feature vector $x \in \mathbb{R}^{15}$ of electrical signals and three binary features indicating risk factors including family history, age, and gender. The observation is a label $y \in \{0, 1\}$ of whether the patient is having a seizure. The analyst wants to trains a linear regression model to predict seizures. This will allow her to understand how doctors can suggest diagnostics for new patients.*

Medical records are often dirty due various problems in digitization and data integration [2]. After model training, the analyst is informed of corruption in the patient data. With some preliminary data exploration, she finds out that older patients whose records were manually digitized are more likely to have missing values. In addtion, cccurances of certain types of seizures are mislabled based on a changing medical definition. However, the procedure to determine which patients' data are corrupted requires querying source medical records which are in a semi-structured format.

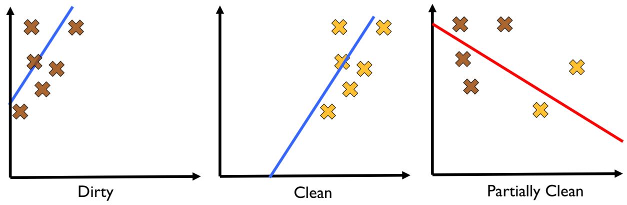

Figure 2: When there are systematic biases, mixing dirty and cleaned data can lead to unreliable models. In this example, a best fit line after cleaning two data points reverses its trend.

Cleaning With a Budget: The best scenario is if the dirtiness is not systematic (i.e., corruption affects all patients randomly), then she can just discard the corrupted data which would be equivalent to random sampling. However, she knows that certain records are more likely to be affected by the corruption and possibly making her linear regression model inaccurate. Let us suppose our analyst wants evaluate whether the problems affect her model, and she wants to do this without having to manually validate every record.

The first solution to this problem is to clean a sample of data, write this sample back, and then retrain the model on the partially cleaned data. However, this solution can give unreliable results since we are mixing dirty and clean data. In Figure 2, we illustrate the dangers of this approach on a very simple dirty dataset and model. Suppose, we are trying to find a best fit line (marked in blue) for two variables. One of the variables is systematically corrupted with a translation in the x-axis, and we illustrate the dirty, clean, and partially cleaned data. If we clean two of the data points, we find that the trend of the partially cleaned model is the opposite of what it is supposed to be. This is a well-known phenomenon called Simpsons paradox, where mixtures of different populations of data can result in spurious relationships.

The next solution is avoiding the dirty data altogether. She can take a sample of data, apply data cleaning, and train a model only on that sample of data. This approach is called SampleClean [32] and was proposed to approximate the results to aggregate queries by applying them to a clean sample of data. We discussed the problem with this approach in our introduction. Machine Learning models are highly senstive to sample size when high dimensional. The result is that the models will not be useful (i.e., cannot accurately predict test data) when the sample size is too small.

The Need for ActiveClean: In designing ActiveClean, we try to get the best of both worlds. We want the correctness on intermediate results of SampleClean, but we also want to avoid the strong dependence on sample size. In Figure 3, we illustrate the our ideal tradeoff space of sampling

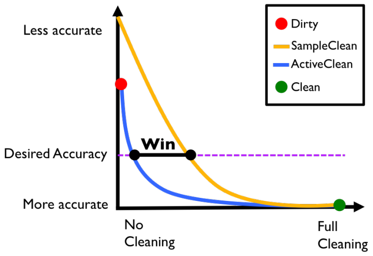

Figure 3: Our goal with ActiveClean is to converge to an accurate model with fewer cleaned records than a naive uniform sampling approach (SampleClean).

and data cleaning. At two extremes we have no cleaning (just using the dirty data) and full cleaning. We argue that the naive approach of SampleClean is not suited for Machine Learning (SampleClean) because it struggles at small sample sizes. Our goal is to optimize our system for faster convergence to an accurate model at these small sizes.

2.2 Preliminaries

In this paper, when we refer to "Machine Learning", we are refering to a class of Supervised Learning problems called Empirical Risk Minimization (ERM) (see Friedman, Hastie, and Tibshirani [13] for an introduction). In ERM, the goal is to learn a set of model *parameters* θ from training examples. We start with a set of training examples $\{(x_i, y_i)\}_{i=1}^N$ on which we minimize a loss function ϕ (a penalty for getting the prediction wrong) at each point 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:

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

Typically, a *regularization* term $r(\theta)$ is added to this problem. $r(\theta)$ 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. The loss function specifies a problem independent of the optimization used to calculate the optimal parameter θ^*.

A very important class of problems is when ϕ and r are convex in θ. For example, convex problems are amenable to a variety of different optimization methodologies and have strong theoretical guarantees on convergence rates. This class of problems includes all generalized linear models (including linear and logistic regression), and all variants of support vector machines. In this work, we focus on this class of problems as it is very broad yet we can prove guarantees about the results.

2.2.1 Types of Errors

When discussion "errors" in Machine Learning models, we precisely refer to the following metrics:

Model Error. Let θ be the model trained on the dirty data, and let θ^* be the model trained on the same data if it was cleaned. Then the model error is defined as $\|\theta - \theta^*\|$.

Testing Error. Let θ be the model trained on the dirty data, and let θ^* be the model trained on the same data if it was cleaned. Let $T(\theta)$ be the out-of-sample testing error when the dirty model is applied to the clean data, and $T(\theta^*)$ be the test error when the clean model is applied to the clean data. The testing error is defined as $T(\theta^*) - T(\theta)$

The *error* in the model and in testing is caused by two underlying problems:

Data Error. Error caused by dirtiness in the base data.

Sampling Error. Error introduced by sampling the data i.e., the difference between a model trained on $p\%$ of the data and 100% of the data.

In this paper, when we use the term *error*, we are refering to **model error**. We will be explicit when we are using the word error to describe problems regarding dirtiness in the underlying data (e.g. "data corruption").

2.3 ActiveClean Problem

Now, we introduce the basic problem that we address in this work.

Problem 1 (ActiveClean Problem). *Let R be a dirty relation and each row $r \in R$ is turned into a feature vector and label tuple $F(r) \mapsto (x, y)$. We are given a convex regularized loss model parametrized by θ trained on the set of features and labels $\{(x, y)\}$. The user specifies a data repair step which cleans a record $C(r) \mapsto r_{clean}$. Optionally, the user can provide a predicate that selects a set of candidate dirty data R_{dirty}. With a budget of applying cleaning i.e., $C(\cdot)$, only k times, we return an estimate $\hat{\theta}$ of the clean model.*

Essentially, this problem describes how can we use the information from a statistical model to inform data cleaning on samples, and how we can use the information from clean samples to udpate a statistical model. The tight feedback loop between model training and data cleaning poses a new algorithmic challenge. Data cleaning and model training happen at very different time scales, we have to carefully budget our effort to ensure that any optimizations actually address rate-determining steps in the workflow.

This framework is optimized around problems with expensive data cleaning. It is well known that most constraint-based repair problems are NP-Hard. As the complexity of dirtiness increases, so does the amount of computation needed to resolve them. The consequence is that automated repair is either not scalable or relies on greedy repairs that can be unreliable. Increasingly, human effort is seen as unavoidable in data cleaning [14,23,31,32]. When humans are involved, per record latencies for data repair are orders of magnitude larger than the CPU time needed for model training. We can compare recent results in data cleaning to a model training framework like CoCoA implemented on Spark [16]. Per record, BigDansing, a highly optimized automated Spark-based data cleaning system is 15.5x slower[1]. Crowd based techiques like CrowdFill [23] and CrowdER [31] are over 100,000x slower per record. Here is an example application of ActiveClean with our running example:

Example 2. *The analyst first trains her linear model on the dirty data ignoring the effects of the errors returning a model $\theta^{(d)}$. She sets a desired cleaning batch size of $b = 10$ and number of iterations $t = 4$. She initializes ActiveClean with $\theta^{(d)}$. ActiveClean samples an intial batch of 10 records and She manually cleans those records by the medical record source data. After each batch, the model is updated with the most recent cleaning results $\theta^{(t)}$. The model improves after each iteration. After $t = 4$ of cleaning, the analyst has an accurate model trained with 40 cleaned records but still utilizes the entire dirty data.*

3. SYSTEM ARCHITECTURE

Now, we overview the ActiveClean architecture, and setup the basic algorithmic framework. We will address the individual components in the subsequent sections.

3.1 Overview

We revist the architecture diagram (Figure 1) in more detail. The first step of the system is *initialization*. In this step, the user provides a dirty dataset R, a repair technique $C(\cdot)$, and a dirty model $\theta^{(d)}$ trained on the dirty dataset. Optionally, the user can provide a dirty data detector $D(\cdot)$ which selects the set of corrupted records when applied to R. This is possible for some types of data errors such as constraint violations and missing values. If one is not provided, we will start by assuming all of the data is dirty and try to learn a detector as we clean more data. At initialization, there are two hyperparameters to set, the batch size (how much data per iteration to repair) and the number of iterations.

After initialization, ActiveClean begins the cleaning and model update iterations. The first iterative step is *sampling*. ActiveClean selects a sample of dirty data based on the batch size. At this step, ActiveClean can use the detector D to narrow the sample to select only dirty data. Once a sample is selected, we apply the user-specified data cleaning operation to the sample in the *repair* step. Then, after the sample is cleaned, the dirty model is updated in the model *update* step.

The next two steps in the archictecture are feedback steps where we update the sampling for the next iteration. We have an *estimation* step which estimates the change on the model if a dirty record is cleaned based on prior cleaned data. This information can be used to guide sampling towards more valuable records. After estimation, we also update our detector $D(\cdot)$ based on the data that we have cleaned. After all of the iterations are complete, the system returns the cleaned model.

To summarize the architecture in pseudocode:

```
1. Init(dirty_data, dirty_model, batch, iter)

2. For each i in $\{1, ..., T\}$
    (a) dirty_sample = Sample(dirty_data, sample_prob,
        detector, batch)
    (b) clean_sample = Repair(dirty_sample)
    (c) dirty_model = Update(dirty_model, sample_prob,
        clean_sample)
    (d) sample_prob = Estimate(dirty_data, cleaned_data,
        detector)
    (e) detector = UpdateDetector(detector)
```

3.2 Challenges and Formalization

We highlight the important components and formalize the research questions that we explore in this paper.

Featurization. Given a relation R with a set of attributes A. There is a featurization function F which maps every row in $\mathcal{R}$ to a d dimensional feature vector and a l dimensional label tuple:

$$F(r \in R) \rightarrow (\mathbb{R}^l, \mathbb{R}^d)$$

The result of the featurization are the data matrices X and Y.

$$F(R) \rightarrow (X, Y)$$

We consider problems in which the training examples (i.e., rows in the data matrix) have a one-to-one relationship with rows in the base data (R).

Detect (Section 4). The first challenge in ActiveClean is dirty data dection. This is key part of our sampling framework. In this step, we select a candidate set of dirty records $R_{dirty} \subseteq R$. We will discuss two techniques to do this: (1) an a priori case, and (2) and adaptive case. In the a priori case, we know which data is dirty in advance. In the adaptive case, we have to train a classifier based on data we have already cleaned to select the dirty data. This serves two purposes: (1) it reduces the variance of our updates because we can cheaply scan over data we know that is clean, and (2) it increases the fraction of actually dirty records in the candidate batch.

Sample (Section 6). We take a sample of the records $S_{dirty} \subseteq R_{dirty}$. This is a non-uniform sample where each record r has a sampling probability p_r. We will show that we can derive the optimal sampling distribution (choice of p_r that minimizes the variance of our updates) theoretically, however, it will depend on knowing the cleaned data before we actually clean. This is why we introduce an estimation step to approximate this cleaned value.

Update (Section 5). Then, we update the model $\theta^{(}(i))$ based on the newly cleaned data $F(S_{clean})$. The result is the updated model $\theta^{(}(i+1))$. We will present the model update before we derive the optimal sampling distribution. This is because that distribution will naturally fall out of our update procedure.

Estimate (Section 7): The estimation step approximates the optimal distribution derived in the Sample step. Based on the change between $F(S_{clean})$ and $F(S_{dirty})$, we direct the next iteration of sampling to select points that will have changes most valuable to the next model update.

4. DETECTION

An important part of ActiveClean is dirty data detection. When we sample data to clean, we want to ensure that the

[1] For CoCoA to reach a precision of 1e-3

data that we are sampling is actually dirty. Therefore, before we can sample, we have to detect whether the record is dirty.

4.1 Goals

The detection component needs to give us two important pieces of information about a record. First, it needs to tell us if a record is dirty or clean. In this framework, we favor false positives (record is clean but marked as dirty) over false negatives as the false negatives will never get sampled and cleaned. Next, the detection also needs to tell us what is wrong with the record. This is important so we can estimate how valuable cleaning will be for this record. There are two cases that we explore in ActiveClean: *a priori* and adaptive. In the a priori case, we recognize that for many data cleaning methodologies, we can efficiently select the set of dirty records without repair. In the adaptive case, we relax this assumption, and explore how we can learn which records are dirty and clean with a classifier.

4.2 A Priori Case

For many types of dirtiness, it is possible to efficiently enumerate a set of corrupted records and enumerate what is wrong with these records.

Definition 1 (A Priori Detection). *Let r be a record in R. An a priori detector is a detector is a detector that returns a Boolean of whether the record is dirty and a set of columns e_r that are dirty.*

$$D(r) = (\{0,1\}, e_r)$$

From the set of columns that are dirty, we can find the corresponding features that are dirty f_r.

We highlight example use cases of this definition using data cleaning methodologies proposed in the liteature.

Constraint-based Repair: One model for handling errors in database declaring a set of constraints on a database and iteratively fixing records such that the constraints are satisfied [11,18,34].

Detection. Let Σ be a set of constraints on the relation $\mathcal{R}$. In the detection step, we select a subset of records $\mathcal{R}_{dirty} \subseteq \mathcal{R}$ that violate at least one constraint. The set e_r is the set of columns for each record which have a constraint violation.

Example 3. *Consider our EEG data running example, for an example of constraint-based cleaning. We can add a constraint that one of the electrical signals cannot be greater than $4V$. For all records whose value is above $4V$, we would select them in the error detection step.*

Entity Resolution: Another common data cleaning task is Entity Resolution [14,19,31]. A common pattern in Entity Resolution is to split up the operation into two steps: blocking and matching. In blocking, attributes that should be the same are coarsely grouped together. In matching, those coarse groups are resolved to a set of distinct entities.

Detection. This is the matching step. Let S be a similarity function that takes two records and returns a value in $[0,1]$ (1 most similar and 0 least similar). For some threshold t, S defines a similarity relationship between two records r and r':

$$r \approx r' : S(r,r') \geq t$$

In the detection step, R_{dirty} is the set of records that have at least one other record in the relation that satisfy $r \approx r'$. The set e_r is the attributes of r that have entity resolution problems.

Example 4. *An example of an Entity Resolution problem is where the patient's gender is inconsistently represented (e.g., "Male", "M", "Man"). We can define a similarity relationship $(first\ char =' M')$ and select all records that satisfy this condition.*

4.3 Adaptive Detection

Next, we explore how we can handle the case where the detection is learned as we clean data. The challenge in formulating this problem is that we not only need to know whether or not a record is dirty, but also how it is dirty (e.g. e_r in the a priori case). We try to formulate a technique that generalizes what we did before. Instead of assuming that we know which features are corrupted, let us say that we know that there are u classes of data corruption. As the analyst cleans data, she tags dirty data with one of the u classes. Then, the detection problem reduces to a multiclass classification problem. To address this problem, we can use any multiclass classifier, and we use an all-versus-one SVM in our experiments.

When an example (x, y) is cleaned, the repair step also has to provide a label of which of the clean, $1, 2, ..., u$ classes it belongs. It is possible that u increases each iteration as more types of dirtiness are discovered. Thus, we after cleaning k records, we have a dataset of k records labeled with $u + 1$ classes (including not dirty).

Definition 2 (Adaptive Case). *To select R_{dirty}, we select the set of records for which κ give a positive error classification (i.e., one of the u error classes). After each sample of data is cleaned, the classifier κ is retrained. So the result is:*

$$D(r) = (\{1,0\}, \{1, ..., u+1\})$$

As we mentioned before, we favor false positive errors over false negative errors when detecting dirty data. Many types of classifiers allow users to tradeoff precision and recall. In other words, we can also select any record within some level of confidence of the classification margin. 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%.

Interactive Data Cleaning with OpenRefine.

Example 5. *Consider our analyst using an interative tool such as OpenRefine [28] to clean her data. She takes a sample of data from the entire dataset. She then uses the tool to determine which records are corrupted and dirty. As she marks more records as dirty, the system learns what features determines a dirty record.*

5. UPDATE

Before we discuss sampling, we will first discuss how to update a model. We will assume that our detection step in the previous section has selected a set of candidate records R_{dirty} and our sampling step has sampled from this set of candidate records. We will show that this model update procedure can be interpreted as a Stochastic Gradient Descent (SGD) algorithm, which gives us a theoretical framework to analyze convergence and bound the error at each step.

5.1 Update Problem

The challenge is that we want a model update technique that takes advantage of the entire dirty data and a small sample of cleaned data. However, we want to avoid the Simpson's paradox problem discussed previously. Let us describe the abstract problem. Suppose we have a random sample of data $S_{dirty} \subseteq R_{dirty}$ from the results of the detection in the previous section. We also know the sampling probabilities of each record $p(r)$. We apply our repair to the sample and get S_{clean}, and then apply our featurization to get features and labels (X_{clean}, Y_{clean}). In the model update problem, we update the dirty model $\theta^{(d)}$ with some function $f(\cdot)$ i.e.,:

$$\theta^{new} \leftarrow f(X_{clean}, Y_{clean}, \theta^{(d)})$$

Our goal is that these updates should minimize the error of the updated model and the true model $\theta^{(c)}$ (if we cleaned and trained over the entire data):

$$error(\theta^{new}) = \|\theta^{new} - \theta^{(c)}\|$$

5.2 The Geometry Of The Problem

We will present the update algorithm intuitively by describing it in terms of the convex geometry of the problem. Let us consider this problem in one dimension (i.e., the parameter θ is a scalar vale), so then the goal is to find the minimum point (θ) of a curve $l(\theta)$. Convexity means, as we vary the model θ the loss is essentially bowl-shaped (visualized in Figure 4a).

The way that dirty data affects this problem, is that we are optimizing the wrong function. We are optimizing the function $l'(\theta)$, when we really want to optimize $l(\theta)$. Figure 4a, illustrates the consequence of this optimization. If we train a model on the dirty data, the model is not the optimal model for the cleaned data. The resulting dirty model is a suboptimal point on clean curve.

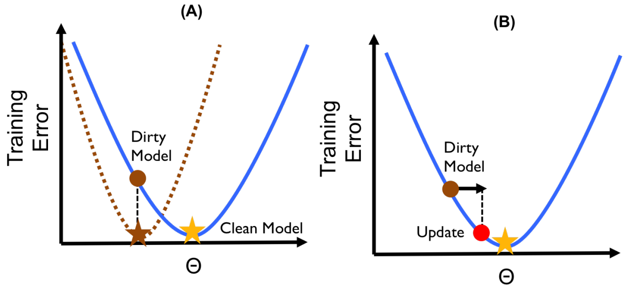

Figure 4: In ActiveClean, we explore an important class of Machine Learning problems where the loss function (i.e., training error) varies convexly. An incorrect model can be thought of as a sub-optimal point.

We want to get to the optimal clean model $\theta^{(c)}$ which is visualized as a yellow star. The first question is which direction do we move theta. Without cleaning the data, we do not know whether we need to move left or right.

First, consider if we had all of the clean data we could compute $\theta^{(c)}$. For this class of models, given a suboptimal point, we can find the direction to the the global optimum. Mathematically, this direction corresponds to gradient of the loss with respect to θ, and we need to move some distance γ along this direction (Figure 4b):

$$\theta^{new} \leftarrow \theta^{(d)} - \gamma \cdot \nabla\phi(\theta^{(d)})$$

In our visualization, this corresponds to whether we should move left or right. At the optimal point, the expected gradient will be zero. So intuitively, this approach iteratively moves the model downhill, or corrects, the dirty model until the budget is reached.

However, in reality, we do not have the all of the clean data and have to approximate this gradient from a sample. The intuition, which we will formalize in Section 5.5, is that if we are on average in the right direction the algorithm is guaranteed to converge with analytics bounds on the convergence rate.

5.3 Average Gradient From a Sample

To derive a sample-based update rule, the first property that we should recognize is that sums commute with derivatives and gradients. Our class of Machine Learning models are based on loss minimization, that is they are a sum of losses. So to calculate the average gradient from a sample, we simply calculate the gradient at every data point and average over them. Let S be a sample of data, where each $i \in S$ is drawn with probability p_i. We can approximate the global gradient:

$$\nabla\phi(\theta) \approx g_S(\theta) = \frac{1}{n \mid S \mid} \sum_{i \in S} \frac{1}{p_i} \nabla\phi(x_i^{clean}, y_i^{clean}, \theta)$$

Then for every batch of data cleaned, we apply the update to the current best model estimate:

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

The dection in the previous step adds a small complication, since the sample S_{dirty} is not representative of all of the data only R_{dirty}. We also have to compensate for this bias by averaging this estimate with the gradient with respect to the clean data:

$$g_C(\theta) = \frac{1}{\mid R - R_{dirty} \mid} \sum_{i \in R - R_{dirty}} \nabla\phi(x_i^{clean}, y_i^{clean}, \theta)$$

Then, for weights α, β which we will subsequently discuss how to select:

$$g(\theta) = \alpha \cdot g_C(\theta) + \beta \cdot g_S(\theta)$$

So then our final gradient update becomes:

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

5.4 Model Update Algorithm

Following from this geometric analysis, we propose the iterative model correction algorithm. We initialize the algorithm with $\theta^{(d)}$ which is the dirty model. At each iteration $i = \{1, ..., t\}$, we clean a batch of data b selected from the set of candidate dirty records R_{dirty}. Then, starting with $\theta^{(d)}$, we apply an averaged gradient update to get $\theta^{(i)}$. We iterate until our budget of cleaning $k = t \cdot b$ record is reached.

We present the model update algorithm here:

1. Calculate the gradient over the sample of clean data and call the result $g_S(\theta^{(i-1)})$
2. Calculate the average gradient over all the data in R_{clean}, and call the result $g_C(\theta^{(i)})$
3. Apply the following update rule:

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \lambda \cdot (\alpha \cdot g_S(\theta^{(i)}) + \beta \cdot g_C(\theta^{(i)}))$$

5.4.1 Selecting the Parameters

The proposed update policy makes intuitive sense. However, we have still have to set the parameters λ, α, β. We will show that theory from the stochastic optimization literature can allow us to understand this iterative algorithm. First, we will review all of the parameters that need to be set.

Step Size γ : The first problem is that we have not explained how to pick the step size γ. In other words, how far should we travel in the gradient direction.

Step Size α, β : The next problem is deriving the proportions with which we should combine $g_S(\theta)$ and $g_C(\theta)$. It turns out that $\alpha = \frac{R_{clean}}{R}$ and $\beta = \frac{R_{dirty}}{R}$, and we will show a derivation below.

Choosing S: As we mentioned, the optimal clean model depends on *all* the clean data, not just a sample. So g_S is really an approximation of the the true gradient g^* with some error $g^* \pm \epsilon$. The quality of the update depends on how well we can approximate g^* using g_S. The problem is how should we construct the sample of data to clean S to get the most accurate update. In particular, how should we choose the sampling probabilities p_i for each $i \in S$ such that the error is minimized.

5.5 Analysis with Stochastic Gradient Descent

This update policy can be formalized as a class of very well studied algorithms called Stochastic Gradient Descent. This gives us a theoretical framework to understand and analyze our update rule, bound the error, and choosing points to clean. Mini-batch stochastic gradient descent (SGD) is an algorithm for finding the optimal value of θ, given the convex loss, and data.

Setting γ: There is extensive literature in machine learning for choosing γ appropriately. γ can be set either to be a constant or decayed over time. Many machine learning frameworks (e.g., MLLib, Sci-kit Learn, Vowpal Wabbit) automatically set learning rates or provide different learning scheduling frameworks. In our experiments, we use a technique called inverse scaling where there is a parameter $\gamma_0 = 0.1$, and at each iteration we reduce it to $\gamma_t = \frac{\gamma_0}{|S|t}$.

Convergence: The next property of concern is convergence. Convergence properties of batch SGD formulations has been well studied [9]. The conditions on this convergence are essentially that at each step the estimate of the gradient g_S has to be unbiased, that is on average correct.

Proposition 1. *For an appropriately chosen learning rate γ_t, batch stochastic gradient descent will converge if $\mathbb{E}(g_S) = g^*$.*

A direct result of convergence is a guarantee on statistical consistency. Another interesting consequence of this result is that we can sample S in any way that we want as long as our estimate in unbiased.

Deriving α and β: We first select a set of records before starting we select a set of dirty records $R_{dirty} \subseteq R$. We construct batch S_{dirty} only from the dirty records and apply cleaning to this batch. Now the problem is that this sample (and the resulting gradient) is possibly biased since we are excluding some data. However, since we know that the set $R_{clean} = R - R_{dirty}$ is clean, we can compute the average gradient over those records without cleaning:

$$g_C(\theta^t) = \frac{1}{\mid R - R_{dirty} \mid} \sum g_i(\theta^t)$$

Since we know the partition sizes, we can combine the two estimates g_C and g_S togther:

$$g(\theta^t) = \frac{\mid R_{dirty} \mid \cdot g_S + \mid R_{clean} \mid \cdot g_C}{\mid R \mid}$$

Therefore,

$$\alpha = \frac{R_{clean}}{R}, \beta = \frac{R_{dirty}}{R} \blacksquare$$

Lemma 1. *The gradient estimate $g(\theta^t)$ is unbiased if g_S is an unbiased estimate of:*

$$\frac{1}{\mid R_{dirty} \mid} \sum g_i(\theta^t)$$

Proof Sketch. This result follows directly from the linearity of expectation, since we are adding a deterministic result with an unbiased result. □

Convergence Rate: The convergence rates of SGD are also well analyzed [7,9,35]. Suppose, a user cleans a batch size of b examples at each iteration. This allows us to bound the error of intermediate models and understand the expected number of steps before a model within a certain error.

Proposition 2. *For a general convex loss, a batch size b, and t iterations, the convergence rate is bounded by $O(\frac{\sigma^2}{\sqrt{bt}})$. σ^2 is the variance in the estimate of the gradient at each iteration:*

$$\mathbb{E}(\|g_S - g^*\|^2)$$

There is an interesting tradeoff between batch size an convergence rate. Increasing the batch size reduces the variance σ^2, however it does mean at each iteration you are doing more work. In a data cleaning context, this is a problem since the bottlneck is the work at each iteration not the number of iterations.

Laziness Argument: The convergence and the error bounds show that the technique is theoretically justified, but the remaining question is whether the constant factors make this technique practical. It is well known that even for arbitrary initializations SGD makes significant progress in less than one epoch (a pass through the entire dataset) [8]. In our case, we often have a decent initialization which is the dirty model, which means that we can get close processing far less than the full data. Our algorithm is essentially an arguement for lazy materialization, where we only clean the data when needed.

6. SAMPLING

In the previous section, we assumed that our model update recieved a sample with probabilities $p(r)$ In this section, we derive an optimal sampling problem that directly follows from our analysis of our update rule via SGD. It will turn out that the solution to the optimal sampling problem is not realizable in practice (as it depends on knowing the cleaned value), but we can use this to inform the next section where we estimate the cleaned value.

6.1 Goals and Challenges

In the Machine Learning and Optimization literature, SGD algorithms are optimized to avoid scanning the entire data. Uniform sampling is cheap so it is the preferred solution. However, data cleaning costs can be many orders of magnitude higher than model training. As a result, uniform sampling may not be the most efficient option. We can sacrifice computational overhead by precomputing some results over the entire data for savings during the data cleaning phase. We formulate this problem as an optimal sampling problem where we want to compute the sampling probabilities $p(r)$ that maximize the accuracy of our updates.

6.2 Optimal Sampling Problem

Recall that the convergence rate of an SGD algorithm is bounded by σ^2 which is the variance of the gradient.

DEFINITION 3 (OPTIMAL SAMPLING PROBLEM). *Given a set of candidate dirty data R_{dirty}, $\forall r \in R_{dirty}$ find sampling probabilites $p(r)$ such that over all samples S of size k it minimizes:*

$$\mathbb{E}(\|g_S - g^*\|^2)$$

In other words, we want to most accurately (in the mean squared error sense) estimate the gradient.

To construct these sampling probabilities, we first need the following lemma:

LEMMA 2. *Given a set of real numbers $A = \{a_1, ..., a_n\}$, let $\hat{A}$ be a sample with replacement of A of size k. If μ is the mean $\hat{A}$, the sampling distribution that minimizes the variance of μ, i.e., the expected square error, is $p(a_i) \propto a_i$.*

PROOF SKETCH. This proof follows from [21], as it is a straight forward importance sampling result. It is easy to verify that for sampling probabilities $p(a_i)$, that the unbiased estimate of the mean is:

$$\mu = \frac{1}{nk} \cdot \sum_{i \in \hat{A}}^{k} \frac{a_i}{p_i}$$

If we minimize the variance $Var(\mu)$, we can find that:

$$p_i = \frac{\mid a_i \mid}{\sum_i \mid a_i \mid}$$

□

Based on Lemma 2, we now derive an optimal sampling distribution. Lemma 2 shows that when estimating a mean of numbers with sampling, the distribution with optimal variance is where we sample proportions to the values. This is insight leads to a direct higher-dimensional generalization, where at iteration t we should sample the records in R_{dirty} with probabilities:

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

However, in our case, it leads to a chicken-and-egg problem. The optimal sampling distribution requires knowing $(x_i^{(clean)}, y_i^{(clean)})$, however, we have to sample and clean those points to get those values. In the next section, we discuss how to inexpensively approximate this optimal distribution. As our technique can work with *any* distribution, we are guaranteed convergence no matter how inaccurate this approximation. However, a better approximation will lead to an improved convergence rate.

7. ESTIMATION

In this section, we make the sampling result of the previous section practical, and estimate the cleaned data.

7.1 Challenges and Goal

The optimal sampling distribution is dependent on a value that we cannot know without data cleaning $\nabla\phi(x_i^{(clean)}, y_i^{(clean)}, \theta^t)$. One way to approximate this distribution is to learn a function $e(\cdot)$ mapping (x_{clean}, y_{clean}), based on the data that we have cleaned. Suppose, we use a regression technique based on the previous data. This is a high-dimensional regression problem which may have to learn a very complicated relationship between dirty and clean data. The biggest challenge with such an estimator is the cold start problem, where if we have cleaned a small amount of data, the estimator will be inaccurate. In ActiveClean, we want to be able to make as much progress as possible in the early iterations so this technique may not work. We take an alternative approach, where we try to exploit what we know about data cleaning to produce an estimate for groups of similarly corrupted records. To define "similarly corrupted", we are going to show how we can use the detection step to make this problem tractable.

7.2 Estimation For A Priori Detection

EXAMPLE 6. *Consider our EEG dataset with multiple types of errors. Suppose the electrical signals for some records are corrupted with missing data and the patient data for other records are corrupted with inconsistency problems. Each training example will have a set of corrupted features (i.e., $\{1, 2, 6\}$, $\{1, 2, 15\}$).*

Suppose that we have just cleaned the records r_1 and r_2 represented as tuples with their corrupted feature set: (r_1,$\{1, 2, 3\}$), (r_2,$\{1, 2, 6\}$). Then, we have new record (r_3,$\{1, 2, 3, 6\}$). We want to be able to use the cleaning results from r_1, r_2 to estimate the gradient in r_3.

If most of the features are correct, it would seem like the gradient is only incorrect in one or two of its components. The problem is that the gradient $\nabla\phi(\cdot)$ can be a very nonlinear function of the features that couple features together. For example, let us look at the gradient for linear regression:

$$\nabla\phi(x, y, \theta) = (\theta^T x - y)x$$

We see that it is not possible to isolate the effect of a change of one feature on the gradient. Even if one of the features is corrupted, all of the gradient components will be incorrect.

7.2.1 Error Decoupling

We can try to approximate the gradient in a way that the effects of features on the gradient are decoupled. Recall, that when we formalized the a priori detection problem, we ensured that associated with each $r \in R_{dirty}$ is a set of errors f_r which is a set that identifies a set of corrupted features. We will show how we can use this property to construct a coarse estimate of the clean value. The main idea is that if we can calculate average changes for each feature, then given an uncleaned (but dirty) record, we can add these average changes to correct the gradient.

Let us formalize this intuition. Instead of computing the actual gradient with respect to the true clean values, let us compute the conditional expectation given that a set of features and labels f_r are corrupted:

$$p_i \propto \mathbb{E}(\nabla\phi(\theta_{(t)}^T x_{clean}, y_{clean}) \mid e_r)$$

What we mean by corrupted features is that:

$$i \notin f_r \implies x_{clean}[i] - x_{dirty}[i] = 0$$

$$i \notin f_r \implies y_{clean}[i] - y_{dirty}[i] = 0$$

The needed approximation represents a linearization of the errors. We will show that the sampling distribution can be estimated in the form:

$$p_r \propto \|\nabla\phi(x, y, \theta^{(t)}) + M_x \cdot \Delta_{rx} + M_y \cdot \Delta_{ry}\|$$

where M_x, M_y are matrices and Δ_{rx} and Δ_{ry} are average change vectors for the corrupted features in r. Without decoupling, if we were to calculate the expected value conditioned on e_r, we would have to condition on all the combinatorial possibilities.

7.2.2 Deriving M_x, M_y

We can take the expected value of the Taylor series expansion around the dirty value. If d is the dirty value and c is the clean value, the Taylor series approximation for a function f is given as follows:

$$f(c) = f(d) + f'(d) \cdot (d - c) + ...$$

If we ignore the higher order terms, we see that the linear term $f'(d) \cdot (c - d)$ decouples the features. We only have know the change in each feature to estimate the change in value. In our case the function f is the gradient $\nabla\phi$. So, the resulting linearization is:

$$\nabla\phi(\theta^T x_{clean}, y_{clean}) \approx \nabla\phi(\theta^T x, y) + \frac{\partial}{\partial X}\nabla\phi(\theta^T x, y) \cdot (x - x_{clean})$$

$$+\frac{\partial}{\partial Y}\nabla\phi(\theta^T x, y) \cdot (y - y_{clean})$$

When we take the expected value:

$$\mathbb{E}(\nabla\phi(\theta^T x_{clean}, y_{clean})) \approx \nabla\phi(\theta^T x, y) + \frac{\partial}{\partial X}\nabla\phi(\theta^T x, y) \cdot \mathbb{E}(\Delta x)$$

$$+\frac{\partial}{\partial Y}\nabla\phi(\theta^T x, y) \cdot \mathbb{E}(\Delta y)$$

So the resulting estimation formula takes the following form:

$$\approx \nabla\phi(\theta^T x, y) + M_x \cdot \mathbb{E}(\Delta x) + M_y \cdot \mathbb{E}(\Delta y)$$

Recall that we have a d dimensional feature space and l dimensional label space. Then, $M_x = \frac{\partial}{\partial X}\nabla\phi$ is an $d \times d$ matrix, and $M_y = \frac{\partial}{\partial Y}\nabla\phi$ is a $d \times l$ matrix. Both of these matrices are computed with respect to dirty data, and we will present an example. Δx is a d dimensional vector where each component represents a change in that feature and Δy is an l dimensional vector that represents the change in each of the labels.

7.2.3 Example M_x, M_y

Let us walk through an example, where we calculate M_x, M_y. In the linear regression example, where the gradient is:

$$\nabla\phi(x, y, \theta) = (\theta^T x - y)x$$

For a record, r, suppose we have a feature vector x. It we take the partial derivatives with respect to x M_x is:

$$M_x[i, i] = 2x[i] + \sum_{i \neq j} \theta[j]x[j] - y$$

$$M_x[i, j] = \theta[j]x[i]$$

Similarly M_y is:

$$M_y[i, 1] = x[i]$$

In the appendix, we describe the matrices for common convex losses.

7.2.4 More Accurate Early Error Estimates

We can give some intuition on why the linearization actually avoids amplifying estimation error. If we consider the linear regression gradient:

$$\nabla\phi(x, y, \theta) = (\theta^T x - y)x$$

We can rewrite this as a vector in each component:

$$g[i] = \sum_i x[i]^2 - x[i]y + \sum_{j \neq i} \theta[j]x[j]$$

We see that this function is already mostly linear in x except for the one quadratic term. However, this one quadratic term has potential to amplify errors. Consider two estimates:

$$f(x + \epsilon) = (x + \epsilon)^2 = x^2 + 2x\epsilon + \epsilon^2$$

$$f(x + \epsilon) \approx f(x) + f'(x)(\epsilon) = x^2 + 2x\epsilon$$

The only difference between the two estimates is the quadratic ϵ^2, if ϵ is highly uncertain random variable then the quadratic dominates. Since we are using an average which is a coarse grained estimate high error is unavoidable. The bottom estimate while a potentially biased approximation avoids amplifying this error. We evaluate our technique in Section 8.5 against alternatives, and we find that indeed we provide more accurate estimates for a small number of samples cleaned. When the number of cleaned samples increases the alternative techniques are comparable.

7.2.5 Maintaining Decoupled Averages

This linearization allows us to maintain per feature (or label) average changes and use these changes to center the optimal sampling distribution around the expected clean value. We know how to estimate $\mathbb{E}(\Delta x)$ and $\mathbb{E}(\Delta y)$.

Lemma 3 (Single Feature). *For a feature i, we average all records cleaned that have an error for that feature, weighted by their sampling probability:*

$$\bar{\Delta}_i = \frac{1}{NK}\sum_{j=0}^{K}(x[i] - x_{clean}[i]) \times \frac{1}{p_j}$$

Similarly, for a label i:

$$\bar{\Delta}_i = \frac{1}{NK}\sum_{j=0}^{K}(y[i] - y_{clean}[i]) \times \frac{1}{p_j}$$

Then, it follows, that we can aggregate the $\bar{\Delta}_i$ into a single vector:

Lemma 4 (Delta vector). *Let* $\{1..,i,...,d\}$ *index the set of features and labels. For a record r, the set of corrupted features is* f_r. *Then, each record r has a d-dimensional vector* Δ_r *which is constructed as follows:*

$$\Delta_r[i] = \begin{cases} 0 & i \notin f_r \\ \bar{\Delta}_i & i \in f_r \end{cases}$$

With the above theorem, we finally have an approximation to our sampling weights:

$$p_r \propto \|\nabla\phi(x, y, \theta^{(t)}) + M_x \cdot \Delta_{rx} + M_y \cdot \Delta_{ry}\| \blacksquare$$

7.3 Estimation For Adaptive Case

The same logic still holds in the adaptive setting, however, we have to reformulate what we mean by "similarly" corrupted. Here, we use u dirtiness classes. Instead of conditioning on the features that are corrupted, we condition the classes. So for each error class, we compute a Δ_{xu} and Δ_{yu}. These are the average change in the features given that class and the average change in labels given that class respectively.

$$p_{r,u} \propto \|\nabla\phi(x, y, \theta^{(t)}) + M_x \cdot \Delta_{ux} + M_y \cdot \Delta_{uy}\| \blacksquare$$

8. EXPERIMENTS

There are a number of different axes on which we can evaluate ActiveClean. First, we take real datasets and generate various types of errors to illustrate the value of data cleaning in comparison to robust statistical techniques. Next, we explore different prioritization and model update schemes for data cleaning samples. Finally, we evaluate ActiveClean end-to-end in a number of real-world data cleaning scenarios.

8.1 Experimental Setup and Notation

Our main metric for evaluation is a relative measure of trained model with ActiveClean (or an alternative technique) and the model if all of the data is cleaned.

Relative Model Error. Let θ be the model trained on the dirty data, and let θ^* be the model trained on the same data if it was cleaned. Then the model error is defined as $\frac{\|\theta-\theta^*\|}{\|\theta^*\|}$.

Testing Accuracy. Let θ be the model trained on the dirty data, and let θ^* be the model trained on the same data if it was cleaned. Let $T(\theta)$ be the out-of-sample testing accuracy when the dirty model is applied to the clean data, and $T(\theta^*)$ be the testing accuracy when the clean model is applied to the clean data. The testing error is defined as $T(\theta^*) - T(\theta)$

8.1.1 Scenarios

We apply ActiveClean to the following scenarios:

Adult: In this census dataset, our task is to predict the income bracket (binary) from 12 numerical and categorical covariates. There are 45552 data points in this dataset. We use a SVM classifier to predict the income bracket of the person.

EEG: In this dataset, our task is to predict the on set of a seizure (binary) from 15 numerical covariates. There are 14980 data points in this dataset. We chose this dataset since the classification task is hard with an accuracy in clean data of 65%. The model that we use is a thresholded Linear Regression.

MNIST: In this dataset, our task is to classify 60,000 images of handwritten images into 10 categories. The unique part of this dataset is the featurized data consists of a 784 dimensional vector which includes edge detectors and raw image patches. We use this dataset to explore how we can corrupt the raw data to affect subsequent featurization. The model is an one-to-all multiclass SVM classifier.

8.1.2 Alternative Algorithms

Here are the alternative methodologies that we consider:

Robust Logistic Regression [12]. Feng et al. proposed a variant of logistic regression that is robust to outliers. We chose this algorithm because it is a robust extension of the convex regularized loss model, leading to a better apples-to-apples comparison between the techniques.

Discarding Dirty Data. As a baseline we explore model accuracy when dirty data is discarded.

SampleClean (SC) [32]. In SampleClean, we take a sample of data, apply data cleaning, and then train a model to completion.

Active Learning (AL) [15]. There have been recent proposals of integrating Active Learning with stochastic optimization. Active Learning has been widely applied in the data cleaning literature [14], but never integrated with model training. We acknowledge the work in the Machine Learning literature and formulate an Active Learning that uses a prioritization fuction agnostic of the dirty data (no estimation and detection). We clean in batches of 25 records and we use this parameter setting for both AL and ActiveClean.

ActiveClean Oracle (AC+O): In ActiveClean Oracle, we importance sample points by their clean gradient. This represents the theoretical best that our algorithm could hope to achieve given perfect estimation.

8.2 Effect of Cleaning

Before we evaluate ActiveClean, we first evaluate the benefits of cleaning on 2 of our example datasets (EEG and Adult). We first explore this problem without sampling to understand which types of data corruption are amenable to data cleaning and which are better suited for robust statistical techniques. We compare 4 schemes: (1) full data cleaning, (2) robust logistic regression, (3) discarding the dirty data, and (4) baseline of no cleaning. We corrupted 5% of the training examples in each dataset in two different ways:

Random Corruption: We simulated high-magnitude random outliers. We select 5% of the examples and features uniformly at random and replace a feature with 3 times the highest feature value.

Systematic Corruption: We simulated innocuous looking (but still incorrect) systematic corruption. We trained the model on the clean data, find the three most important feature (highest weighted). We sort examples by each these features and corrupt the top of examples with the mean value for that feature. At the end, we have corrupted 5% of the examples. It is important to note the some examples can have multiple corrupted features.

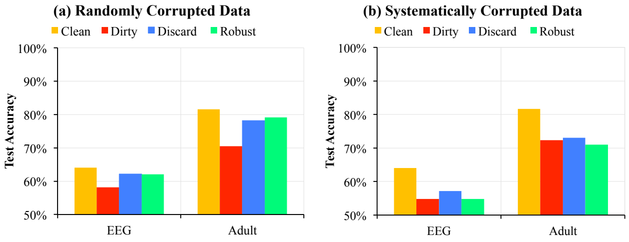

Figure 5: (a) Robust techniques and discarding data work when corrupted data are random and look atypical. (b) Data cleaning can provide reliable performance in both the systematically corrupted setting and randomly corrupted setting.

In Figure 5, we present the results of this experiment. We plot the test accuracy for models trained on both types of data with the different techniques, since it is a binary classification task with balanced class distributions we cut the plots at 50% (random accuracy). As we argued in this paper, the robust method performs well on the random high-magnitude outliers, however, falters on the systematic corruption. In the random setting, discarding dirty data also performs well. However, with systematic corruption, data cleaning is the most reliable option across datasets. The problem is that without cleaning we do not know if the corruption is random or systematic. While data cleaning requires more effort, it provides benefits in both settings. In the remaining experiments, unless otherwise noted, we use the systematic corruption.

8.3 Budgeted Cleaning

The next set of experiments evaluate different approaches to cleaning a sample of data. In this set of experiments, we use the *a priori* case for detection. We assume that we know all of the corrupted records in advance but do not know how to repair them.

8.3.1 Alternative Algorithms

In our first prioritization experiment, we evaluate the samples-to-error tradeoff between three alternative algorithms: ActiveClean (AC), SampleClean, Active Learning, and ActiveClean +Oracle (AC+O). In Figure 6, we present our results on Adult and EEG. We find that ActiveClean gives its largest benefits for small sample sizes (up-to 12x more accurate than SampleClean). ActiveClean makes significant progress because of its intelligent initialization, iterative updates, and partitioning. For example, the EEG dataset is the hardest classification task. SampleClean has difficulty on this dataset since it takes a uniform sample of data (only 5% of which are corrupted on average) and tries to train a model using only this data. ActiveClean and Active Learning leverage the initialization from the dirty data to get an improved result. However, ActiveClean's estimates and detection allow us to beat Active Learning on all three of the datasets. ActiveClean also is relatively close in performance to the oracular version (theoretical optimum).

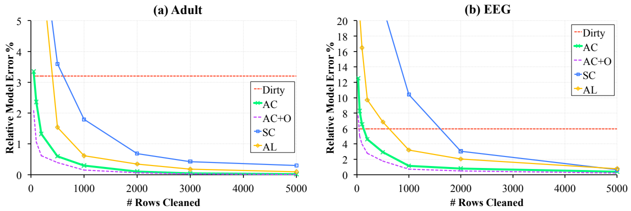

Figure 6: ActiveClean converges with a smaller sample size to the true result in comparison to Active Learning and SampleClean. We show the relative model error as a function of the number of examples cleaned.

8.3.2 Testing Accuracy

In the previous experiment, we studied the relative model error which measures the training loss. However, to an end user the metric that matters is test accuracy. In the next experiment, we try to understand how reductions in model error correlate to improvements in test error. In Figure 7, we present the results for the two datasets: Adult and EEG. We find that in both Adult and EEG, ActiveClean converges to clean test accuracy faster than the alternatives.

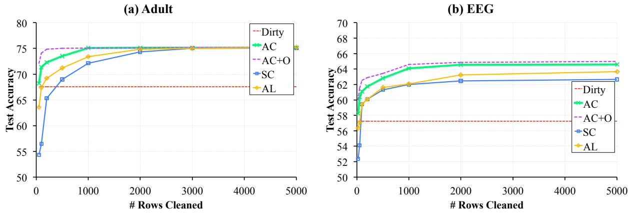

Figure 7: ActiveClean converges with a smaller sample size to the maximum test accuracy in comparison to Active Learning and SampleClean. The reductions in model error correlate well with increased test accuracy in (a). Due to the hardness of the task in (b), we small gains in terms of test accuracy.

8.3.3 Source of Improvements

Now, we try to understand the source of our improvements w.r.t Active Learning and SampleClean. We pick a single point on the curves shown in Figure 6 that corresponds to 500 records cleaned and compare the performance of ActiveClean with and without various optimizations. We denote ActiveClean without detection as (AC-D) (that is at each iteration we sample from the entire dirty data) and ActiveClean without detection and importance sampling as (AC-D-I). In Figure 8, we plot the relative error of the alternatives and ActiveClean with and without the optimizations. Detection significantly improves our results in all of the datasets, and accounts for a substantial part of the improvements over Active Learning. However, when we remove detection we still see some improvements since our importance sampling relies on error impact estimates. Not surprisingly, when we remove both these optimizations, ActiveClean is slightly worse (but still comparable to) than Active Learning.

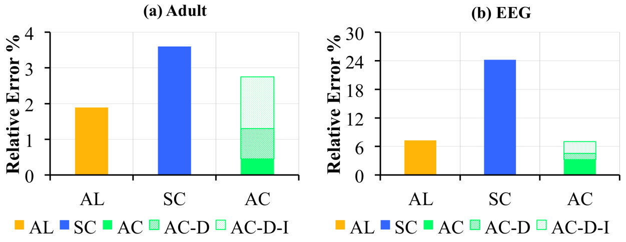

Figure 8: We clean 500 records and plot the relative error of ActiveClean with and without optimizations. -D denotes no detection, and -D-I denotes no detection and no importance sampling. Both optimization significantly help ActiveClean outperform SampleClean and Active Learning.

8.3.4 *Error Dependence*

Both Active Learning and ActiveClean outperform SampleClean in our experiments. In our next experiment (Figure 9), we try to understand how much of this performance is due to the initialization (i.e., SampleClean trains a model from "scratch"). We vary the systematic corruption rate and plot the number of records cleaned to achieve 1% relative error for SampleClean and ActiveClean. SampleClean does not use the dirty data and thus is not dependent on this rate. We find that SampleClean outperforms ActiveClean only when corruptions are very severe (45% in Adult and nearly 60% in EEG).

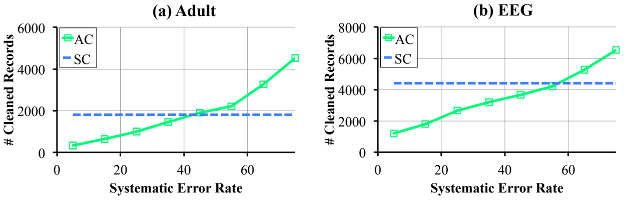

Figure 9: We corrupt an increasing number of entries in the data matrix. ActiveClean performs well until the corruption is so severe that the dirty model is not a good initialization.

8.4 Adaptive Detection

In this experiment, we explore how the results of the previous experiment change when we use an adaptive detector rather than the a priori detector. In our systematic corruption, we corrupted 3 of the most informative features at random, thus we group these problems into 3 classes. We use a all-versus-one SVM to learn the categorization.

8.4.1 *3a. Basic Performance*

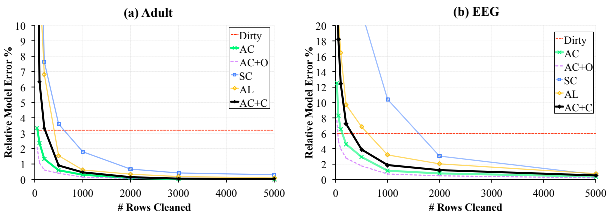

Figure 10: Even with a classifier ActiveClean converges faster than Active Learning and SampleClean.

In Figure 10, we overlay the convergence plots in the previous experiments with a curve (denoted by AC+C) that represents ActiveClean using a classifier instead of the a priori detection. We find an interesting tradeoff where initially ActiveClean is comparable to Active Learning, as our classifier becomes more effective the partitioning improves the performance.

8.4.2 *Classifiable Errors*

The adaptive case depends on being able to predict corrupted records. For example, random corruption that look like other data may be hard to learn. As corruption becomes more random, the classifier becomes increasingly erroneous. We run an experiment where we start with the systematic corruption described earlier. With probability p, we increasingly make these problems more random. We compare these results to AC-D where we do not have a detector at all. In Figure 11, we plot the relative error reduction using a classifier. We find that when the corruption is about 40% random then we reach a break even point.

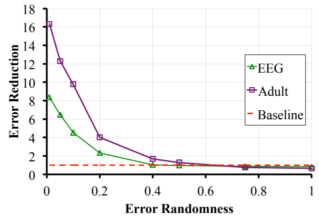

Figure 11: Data corruptions that are less random are easier to classify, and lead to more significant reductions in relative model error.

8.5 Estimation

In the next experiment, we evaluate our estimation technique. We explore how this technique performs compared to a few alternative estimation techniques: (1) "linear regression" where we train a linear regression model that predicts the clean gradient as a function of the dirty gradient, (2) "average gradient" change where we do not use the detection to inform how to apply the estimate, (3) "average feature change" where we do not linearize but use detection, and (4) our taylor series linear approximation. Using the EEG dataset, we measure the relative estimation error (relative L2 error with the true gradient) for different amounts of data cleaned. The Taylor series approximation proposed in this work gives more accurate for small cleaning sizes, which is exactly the regime in which we are interested in optimizing. The other techniques require more data to converge. The linear regression and the gradient do not exploit what we know about the data cleaning, and the gradients can amplify estimation errors in the average feature change approach at small sample sizes (Section 7.2.4).

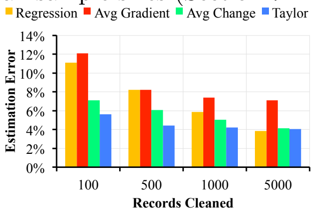

Figure 12: The Taylor series approximation gives more accurate estimates when the amount of cleaned data is small.

8.6 Real Scenarios

We evaluate ActiveClean in two real scenarios, one demonstrating the adaptive detection case and one demonstrating the a priori case.

8.6.1 Replacing Corrupted Data

We consider the following scenario with the MNIST handwritten digit recognition dataset. Suppose, our goal is to classify digits into a set of classes. However, we suspect that some of our raw images are of low quality. Typicaly image processing workflows are long with many different featurization steps. As a result, changes to the raw images may have very significant effects on some features. In this scenario, the analyst must inspect a potentially corrupted image and replace it with a higher quality one. We devise an "adaptive detection" ActiveClean scenario for this example.

The MNIST dataset consists of 64x64 grayscale images. We run two experiments, in which we have two types of corruptions: (1) 5x5 block removal where take a random 5x5 block from the image and set its pixel values to 0, and (2) Fuzzy where we run a 4x4 moving average over the entire image. We applied these corruptions to a random 5% of the images. We constructed these features to mimic the random vs. systematic corruption that we studied before. The 5x5 block removal behaves much more like a systematic corruption. Typical image processing features are based on edges and corrupting edges leads to ambiguities. On the other hand, the making the image fuzzy is more like a random corruption. We use a 9 class classifier (one for each digit) to detect the corruption.

In Figure 13, we present the results. As in our earlier experiments, we find that ActiveClean makes more progress towards the clean model with a smaller number of examples cleaned. This experiment highlights a few other interesting points. First, SampleClean converges at the same rate independent of the data corruption. This is because SampleClean does not depend on any quantity derived from the dirty data. Next, the gap between the theoretical AC+O and what we are able to achieve is much larger in this dataset. We speculate that this is because classifying corruption in a 784 dimensional space is much harder than in our previous experiments, and this in turn leads to reduced accuracy in impact estimation.

8.6.2 Replacing Corrupted Data

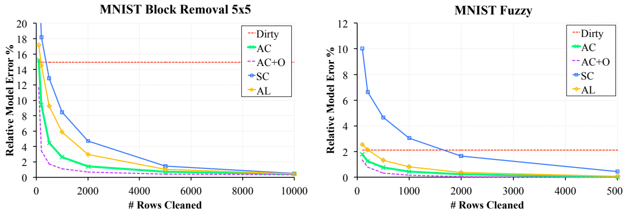

Figure 13: In a real adaptive detection scenario with the MNIST dataset, ActiveClean outperforms Active Learning and SampleClean. We simulate systematic corruptions with image block removal and random corruptions with image fuzzying.

{{TODO World Bank Data}}

9. RELATED WORK

Bringing together data cleaning and machine learning presents us with several exciting new research opportunities that incorporates 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 [35]. 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 consensus in stochastic optimization that not all data are equal (e.g., [24]). 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 [26].

Active vs. Transfer Learning: While it is natural to draw the connection between ActiveClean 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 [26]. 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. ActiveClean has a stronger link to a field called Transfer Learning [22]. 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 [10,30]. 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 [20], 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 [5]. As in our work, this work studies the tradeoff between resolution cost and result accuracy. This work presents many 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 [29]. 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 [6]. 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.

10. CONCLUSION

In this paper, we propose ActiveClean (ActiveClean), an anytime framework for training Machine Learning models with a data cleaning budget. Naive solutions to this problem can cause sampling errors to dominate any benefit of data cleaning, so instead we propose a gradient-based update to incrementally correct a dirty model. To make this update as impactful as possible, we exploit many properties we know about data cleaning such as the relative ease of error enumeration. Our solution is a linear approximation of the optimal sampling distribution which empirically shows significant improvements over alternative approaches. This formulation fits into a Stochastic Gradient Descent theoretical framework, which allows us to ensure convergence and statistical consistency under relatively mild conditions. The elegance of the SGD formulation is that we can use approximations of approximations and still have a methodology that gives bounded results.

ActiveClean 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 re-training. The optimizations described in ActiveClean are not only restricted to SGD algorithms. We believe we can extend a variant of SDCA (Stochastic Dual Coordinate Ascent) [16] to extend this technique to kernelized methods.

11. REFERENCES

[1] Berkeley data analytics stack.

[2] Big data's dirty problem.

[3] 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.

[4] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, et al. The stratosphere platform for big data analytics. *The VLDB JournalThe International Journal on Very Large Data Bases*, 23(6):939–964, 2014.

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

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

[7] D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010:1–38.

[8] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[9] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13(1):165–202, 2012.

[10] 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.

[11] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.

[12] J. Feng, H. Xu, S. Mannor, and S. Yan. Robust logistic regression and classification. In *Advances in Neural Information Processing Systems*, pages 253–261, 2014.

[13] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[14] 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.

[15] A. Guillory, E. Chastain, and J. Bilmes. Active learning as non-convex optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 201–208, 2009.

[16] 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.

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

[18] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. 2015.

[19] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.

[20] 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.

[21] A. B. Owen. *Monte Carlo theory, methods and examples*. 2013.

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

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

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

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

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

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

[28] R. Verborgh and M. De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.

[29] 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.

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

[31] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.

[32] 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.

[33] H. Xiao, T. DE, B. Biggio, D. UNICA, G. Brown, G. Fumera, C. Eckert, I. TUM, and F. Roli. Is feature selection secure against training data poisoning? 2015.

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

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