# ActiveClean: Training Models on Dirty Data Under a Cleaning Budget

## 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 ActiveClean (ActiveClean), 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 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.

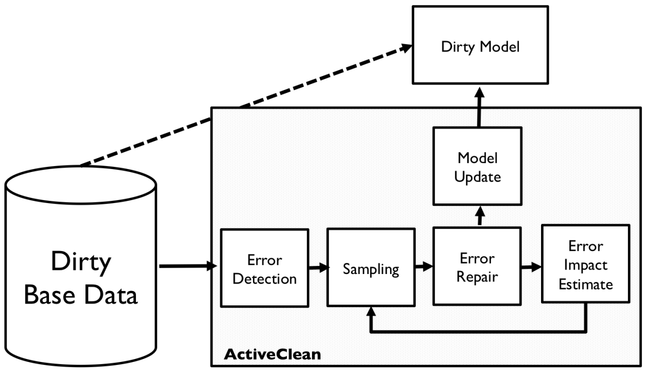

**Figure 1: ActiveClean is an anytime framework for training models on dirty data. Expensive data transformations prior to model training are budgeted with a non-uniform sampling that prioritizes examples that are most likely to change the model.**

## 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 [23], and there is now a growing industry around cleaning 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 from disparate data sources. As these data grow, new methodologies for scalable and reliable analysis in the presence of errors are required.

Increasingly, data analysis pipelines involve Machine Learning. The endpoint of such 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 [27]. 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. Discarding or ignoring this problem can make predictions for the affected subpopulation untrustworthy. While there is an extensive literature on robust Machine Learning, this work largely focuses on resilience to atypical outliers and not systematically corrupted data.

The database community's response to systematic corruption is data cleaning, which is an extensively studied field (see Rahm and Do [21] for a survey). Cleaning works by repairing (or approximately repairing) the corruption in the base data. 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 [13]. In some applications, scripted data transformations may not be reliable necessitating the use of even costlier crowdsourcing techniques [11,19].

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. Sampling 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) [2], where a timely approximate answer is more desirable than an exact slow answer.

Unfortunately, a naive application of sampling does not generalize to Machine Learning. Machine Learning is far more sensitive to sample size (training data) than aggregate queries. Suppose we have a budget of cleaning 50 records. This might be sufficient data to approximate some basic aggregate queries such sums and counts, but in any moderately difficult Machine Learning application, 50 examples is not sufficient to train an accurate model. In other words, in small samples, the error mitigation by data cleaning is dominated by the error introduced by sampling.

In this work, we explore how we can exploit the structure of a Machine Learning problem to train accurate models under a budget. Errors are often sparse and affect a small number of examples and features. This means that a dirty model trained on the full dirty data may be relatively "close" to the true clean model. So instead of attempting to retrain a model from scratch, we can clean batches of data and iteratively correct the old model. To make each iteration as valuable as possible, we can select data that if cleaned will most significantly affect the model.

In this paper, we propose ActiveClean, an anytime framework for training Machine Learning models with data cleaning. Users of ActiveClean have to specify a model, an error detection procedure, and an error repair method. Our system will provide an estimation framework to return an accurate model given a repair budget. In Figure 1, we illustrate our system architecture. ActiveClean supports a class of models called regularized-convex loss problems which includes linear regression, logistic regression, generalized linear models, and support vector machines. We start with a model trained on the dirty data. Then, an *error detection* module selects a set of candidate dirty data. An *error sampling* module chooses what data from the candidate dirty data to clean. We apply the repairs to this small sample with the *error repair* module, and then update the dirty model based on our cleaning. Once we clean data, we maintain a running estimate of how cleaning impacts the model and feed that estimate back to adaptively set the sampling distribution.

The key technical insight of ActiveClean is that even in clean data sampling is naturally part of large-scale Machine Learning. Stochastic optimization methods, such as Stochastic Gradient Descent, start at an arbitrary initialization and calculate updates by drawing training examples at random. In very large datasets, it is well known that substantial progress is made in less than one epoch (i.e., a full pass through the entire dataset) [?]. When errors are relatively sparse, we argue that we often start of with a very good initialization. This insight inspires the key idea of this paper, namely, that we can lazily materialize the clean data only when needed by the optimization. Since we know that we are going to be limited to a single pass of data, we try to make the initial iterations as valuable as possible. There is a growing consensus in Machine Learning research that all training data are not created equal and some data are more informative than others [8,22]. 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.

In summary, our contributions are

- We propose ActiveClean which given dirty data, a convex loss model, a data cleaning procedure, and a budget, we can return a highly accurate model for a fraction of the cleaning cost.
- ActiveClean relies on two key optimizations, importance sampling and error partitioning, that result in substantial emprical gains in accuracy in comparison to Active Learning.
- ActiveClean is implemented with an non-uniform importance sampling approach that prioritizes points in the clean model that are most informative.
- ActiveClean partitions dirty and clean data to reduce the variance of the model update at each iteration.
- 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 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 occurrence of seizures based on Electroencephalogram (EEG) and patient data.

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 trains an L1-Loss SVM on this data.*

*Medical records are often dirty due various problems in digitization and data integration [1]. After model training, the analyst is informed of inconsistencies in the three binary features representing patient data. However, the procedure to determine which patients' data are corrupted requires querying source 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. These solutions pose a dichotomy where either the analyst has to face expensive data cleaning, complete retraining, or cope with the unreliable analysis. In ActiveClean, 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 successive 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* $\theta$ 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 minimize an loss function $\phi$ at each point parametrized that is parametrized by $\theta$.

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

For example, in a linear regression $\phi$ is:

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

$\phi$ 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

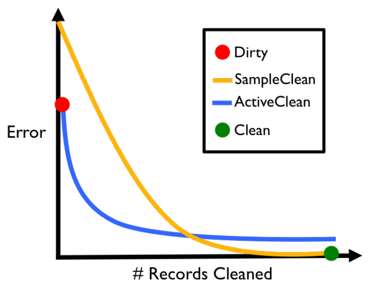

**Figure 2: ActiveClean starts with a dirty model and tries to iterative correct the error in batches. In many datasets, this leads to an improved convergence rate over naive uniform sampling.**

high or low values of feature weights in $\theta$ 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.

## 2.3 Systematic Biases

As in our example, we consider the case when data is systematically biased. If we train a model with respect to the incorrect data, while we might have achieve a good out-of-sample accuracy on the incorrect data, the classifications are incorrect in a semantic sense. 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. The problem that we are interested in is when models are being trained on dirty data but are deployed or tested on clean data (or predicting "real world" trends). Two characterize this problem there are two metrics that we look at:

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

**Testing Error.** Let $\theta$ be the model trained on the dirty data, and let $\theta^*$ 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 $T(\theta^*) - T(\theta)$

## 2.4 Data Cleaning on a Budget

Suppose, we have a budget of cleaning $k \ll N$ records of data. The naive solution is to take a uniform random sample of $k$ records, and apply data cleaning. This approach was proposed in SampleClean [26] for aggregate query processing. This has a two limitations: (1) most of the sampled data is probably clean so a lot of "cleaning" effort is wasted and (2) training an accurate model from only $k$ records may be infeasible. In Figure 2, we illustrate the tradeoff space of sampling 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 uniform sampling is not suited for Machine Learning (SampleClean) because it struggles at small sample sizes. We design ActiveClean to make greater progress at these small sample sizes using the dirty model as an initialization. ActiveClean potentially sacrifices asymptotic performance if the dirty model is not a good initialization, we may have to take a full cleaning pass through the data to train an accurate model. We evaluate these tradeoffs extensively in our experiments.

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 $\theta$ trained on the set of features and labels $\{(x, y)\}$. The user specifies two data cleaning components: (1) error detection which selects a set of candidate dirty data $R_{dirty}$, and (2) error repair which cleans a record $C(r) \mapsto r_{clean}$. With a budget of applying cleaning i.e., $C(\cdot)$, only $k$ times, we return an estimate $\hat{\theta}$ of the optimal clean model.*

## 2.5 Two perspectives on error

When faced with such errors there are two contrasting perspectives from the Machine Learning and the Database communities.

**Existing Database Literature.** Traditionally, cleaning is agnostic to the queries and analysis that happens downstream. This perspective breaks down when cleaning is so expensive that we can only clean a small number of records. Ideally, we should clean the records that are most valuable to the downstream analysis.

**Existing Machine Learning Literature.** The Machine Learning community has focused on designing models that are robust to outliers (i.e., values far away from the typical value) 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). Robust techniques are best suited for random errors that look significantly different the rest of the examples. When errors are systematic, the Machine Learning answer has been to design features in such a way that they are robust to some systematic bias. For example, in image processing, scale-invariant feature transforms (SIFT) are widely applied that allow for image models invariant to pose or scaling issues.

**The ActiveClean Contribution.** We try to bring two perspectives together in this work to address the problem of expensive to clean systematic errors, namely the Database idea of data cleaning and the Machine Learning formalization of empirical risk minimization. Some errors require expensive cleaning procedures, increasingly using the crowd, and we joint have a time budget on cleaning and analysis. ActiveClean prioritizes cleaning with respect to an estimated impact on the clean model.

## 3. SYSTEM ARCHITECTURE

### 3.1 Overview

We revisit the architecture diagram in Figure 1, and formalize the problems addressed in each one of the stages.

**Featurization:** Formally, 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:

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

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

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

The basic assumption is that training examples (i.e., rows in the data matrix) have a one-to-one relationship with rows in the base data ($R$).

**Initialization:** To initialize ActiveClean, the user gives us the base dirty relation $R$, the featurization $F$, and a model $\theta$ that is trained on this dirty relation.

**Error Detection:** The first step in ActiveClean is error detection. In this step, we select a candidate set of dirty records $R_{dirty} \subseteq R$.

**Error Sampling:** The second step in ActiveClean is sampling. Since we cannot clean all of the dirty records, we take a sample of the records $S_{dirty} \subseteq R_{dirty}$.

**Error Repair:** Next, we apply a repair procedure $C$ to the sample of data $S_{dirty}$ to get a clean sample $S_{clean}$.

**Model Update:** Then, we update the model $\theta$ based on the newly cleaned data $F(S_{clean})$.

**Error Impact Estimate:** 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 model update.

### 3.2 User Specified Steps

ActiveClean is a data cleaning methodology that iteratively corrects an incorrect result, and we formalize the conditions on the data cleaning operations that can fit in our framework. The user has two specify two components: error detection and error repair.

#### 3.2.1 *Error Detection*

We have to select a set of candidate records to be cleaned from the base data. In our analysis, we assume that this candidate set is a superset of those records that are actually dirty. Later, in Section 6, we will relax this assumption. Formally, given a relation $R$, we select a subset of records to be cleaned $R_{dirty}$. Associated with each $r \in R_{dirty}$ is a set of errors $e_r$ which tells us which features are corrupted. Therefore, the error detection step gives us the following tuple: $(R_{dirty}, E_r)$.

#### 3.2.2 *Error Repair*

Given a dirty record $r$, the error repair module applies a repair $C$ to the record, to return $C(r) \mapsto r_{clean}$.

### 3.3 Examples

We illustrate how data cleaning processes proposed in the literature can apply in our model. In particular, we describe how we can select a set of candidate records for cleaning and apply cleaning to the sample.

#### 3.3.1 *Constraint-based Errors*

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 [10,14,28]. In this setting, it is relatively inexpensive to query the set of records that violate some constraint, however, automatically repairing the errors is often NP-Hard [10]. Therefore, recently proposed systems like Guided Data Repair [28], use human input to validate suggested fixes. The challenge with using human input is that it is expensive and is often with a time or a cost budget. One way to characterize this problem setting is that error detection is relatively inexpensive but error repairing is expensive.

Formally, constraint-based error detection and repair can be incorporated into ActiveClean in the following way.

**Error Detection.** Let $\Sigma$ be a set of constraints on the relation $\mathcal{R}$. The allowed constraint classes are Matching Dependencies [5], Conditional Functional Dependencies [10], and Denial Constrains [14]. In the error detection step, we select a subset of records $\mathcal{R}_{dirty} \subseteq \mathcal{R}$ that violate at least one constraint. The set $e_r$ the set of columns for each record which have a constraint violation.

**Error Repair.** ActiveClean supports both automated and human validated data repair for constraints. For automated techniques, we can apply the record-level "local" repair proposed in [10] or we can use human corrections as in [28]. Repair operations must satisfy the local property; that is, a repair to a record cannot cause additional constraint violations.

EXAMPLE 2. *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. Then, in the error repair step, we could apply a repair that sets the erroneous signal value to its most likely value.*

#### 3.3.2 *Value Filling*

Value filling is another data cleaning operation that is well studied in the literature. Park and Widom proposed the system called CrowdFill [19], where human workers fill in missing values in a database.

Formally, value filling error detection and repair can be incorporated into ActiveClean in the following way.

**Error Detection.** Let $\phi$ be the null-symbol (signifying a missing value), and $A$ be the set of attributes of $\mathcal{R}$. In the error detection step, we select a subset of records $\mathcal{R}_{dirty} \subseteq \mathcal{R}$ where $\mathcal{R}_{dirty} = \{r \in \mathcal{R} : \exists r(a \in A) = \phi\}$. The set $e_r$ is the attributes of $r$ that have missing values.

**Error Repair.** Value filling can be expressed in ActiveClean if the operation fixes an entire record; that is, for all $a$ such that $r(a) = \phi$, a value is filled.

Example 3. *Suppose, some of the patient information data is missing from dataset in our running example. In the error detection step, we would select all of the records such there is a missing value. In the error repair step, we could confirm that data with other records to fix the missing information.*

### *3.3.3 Entity Resolution*

Another common data cleaning task is Entity Resolution [11,15]. In Entity Resolution, there is an attribute a in R such that $r(a)$ and $r'(a)$ refer to the same real-world entity but their representations are different $r(a) \neq r'(a)$. Entity resolution typically consists of two steps: blocking and matching. In the blocking step, similar records are grouped together, and in the matching step, they are resolved. This process can be expressed in terms of a type of constraint called Matching Dependencies (described above). A Matching Dependency (MD) is a constraint that if this pair is similar according to some similarity relationship (e.g., edit distance and a threshold) then their attribute have to be the same

$$r \approx r' \implies r(a) \doteq r'(a)$$

Thus, this is an important special case of the constraint-based cleaning described before.

**Error Detection.** 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 error 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.

**Error Repair.** ActiveClean supports both automated and human validated entity resolution.

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. In the error repair step, a human can fix the inconsistencies setting all records that meet the condition to the canonical value.*

## 4. ITERATIVE MODEL UPDATE

In this section, we describe how we iteratively update a dirty model based on a cleaned batch of data. 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. This theoretical framework will also inform how we should

## 4.1 Model Update Problem

Suppose we have a cleaned batch of data $S_{clean}$ with features and labels $(X_{clean}, Y_{clean})$, and a dirty model $\theta^{(d)}$. In the model update problem, we update the dirty model 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 the updated model and the true model $\theta^{(c)}$ (if we cleaned and trained over the entire data):

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

### *4.1.1 Naive Solutions*

We will explain why this problem neccesitates a new approach as existing techniques will not do what we want. If we look at the progressive data cleaning problem in the abstract. 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. In fact, if we were to use current data cleaning architectures unmodified this is what they would do. While this solution is guaranteed to be asympotically consistent (i.e., if we clean all the data we are guaranteed the correct solution), the performance of intermediate models can be very unreliable. In statistics, there is a well-known phenomenon called Simpsons paradox, where mixtures of different populations of data can actually result in reversed trends.

So if mixing dirty and clean data is not the correct solution, the next approach would be to neglect the dirty data altogether. We could recompute the model on only the cleaned data. This solution is also guaranteed to be consistent, and intermediate results only reflect the clean data. However, the problem here is that high-dimensional models are very sensitive to the sample size. If our cleaning budget is small, sampling errors will dominate any reductions in data error. We summarize these two approaches in Figure 3.

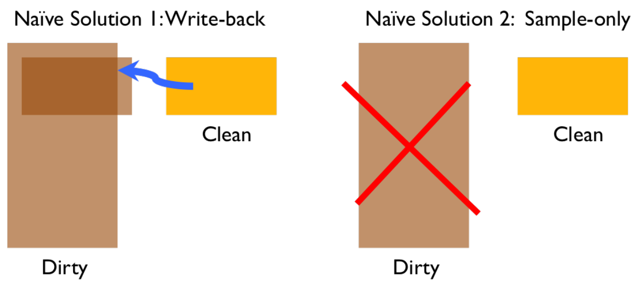

**Figure 3: The two naive solutions to the problem: writing-back the cleaned data, and training only on the sample. Writing back to the cleaned data potentially creates unreliable mixtures of data, and training only on the sample creates an issue of dimensionality and sample size.**

## 4.2 Exploiting the Model's Geometry

ActiveClean addresses the update problem by using the model's geometry. We restrict the class of Machine Learning models to convex regularized loss problems. That is, as we vary the model $\theta$ the loss is bowl-shaped (visualized in Figure 4a). The model update problem can be visualized as follows. We have a model $\theta^{(d)}$ (visualized in red) that is suboptimal with respect to the clean data. We want to get to the optimal clean model $\theta^{(c)}$ which is visualized as a yellow star.

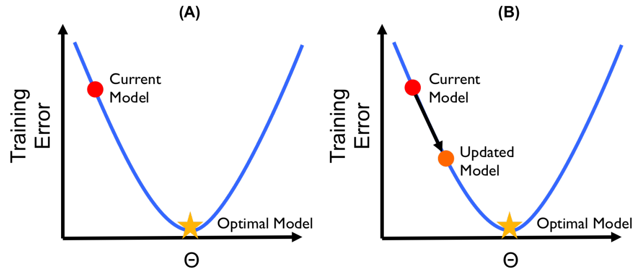

**Figure 4: In ActiveClean, we explore an important class of Machine Learning problems where the loss function (i.e., training error) varies convexly. A stale model can be thought of as a sub-optimal point and we want to move towards the optimal point.**

Ideally, we want an update that goes in the right direction (one that points to the optimum) along the loss curve (Figure 4b). Mathematically, this direction corresponds to gradient of the loss with respect to $\theta$, and we need to move some distance $\gamma$ along this direction:

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

However, since we can only clean a sample of data, we only approximate the globally optimal direction. Our class of Machine Learning models are based on loss minimization, that is they are a sum of losses, and we know that sums commute with derivatives.

Let $S$ be a sample of data, where each $i \in S$ is drawn with probability $p_i$. We can approximate the global gradient:

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

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

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

In other words, we are moving approximately in the right direction. The intuition, which we will formalize in Section 4.4, is that if we are on average in the right direction the algorithm is guaranteed to converge with analytics bounds on the convergence rate. 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.

The tricky part is that only a subset of the records of the records are dirty. In our architecture, we suggested that we want to clean only the records that are actually dirty $R_{dirty}$. For example, as data are cleaned, there is no need to resample this data. The consequence of this is that the estimate $g_S(\theta)$ is now biased. We also have to compensate for this bias by averaging this estimate with the gradient with respect to the clean data:

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

Then, for weights $\alpha, \beta$ 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:

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

Here, we can draw an analogy to Materialized View maintenance, since after all, a model parameterized by $\theta$ is just a table of floating point numbers. Krishnan et al. proposed a technique called sample view cleaning, in which they take a clean sample of data and propogate the updates to a Materialized View. Similarly, in this work, we take the information from a sample of cleaned data and propagate an update with the gradient. The insight that many mathematical problems are analogous to view maintenance has recently been noted others as well [17].

## 4.3 Iterative Model Correction 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 algorithm here:

1. Initialize with $\theta^{(0)} = \theta^{(d)}$ as the dirty model, $t$ iterations, with a batch size $b$, $R_{dirty}$, and $R_{clean} = R - R_{dirty}$.
2. Hyperparamters $\gamma$, $\alpha$, $\beta$.
3. For rounds i=1...t
   (a) Draw a sample of $b$ records from $R_{dirty}$ called $S_{dirty}$.
   (b) Apply data cleaning to the sample of data call the result $S_{clean}$.
   (c) Calculate the gradient over the sample of clean data and call the result $g_S(\theta^{(i-1)})$
   (d) Calculate the average gradient over all the data in $R_{clean}$, and call the result $g_C(\theta^{(i-1)})$
   (e) Apply the following update rule:
   $$\theta^{(i)} \leftarrow \theta^{(i-1)} - \lambda \cdot (\alpha \cdot g_S(\theta^{(i-1)}) + \beta \cdot g_C(\theta^{(i-1)}))$$
   (f) Update the sets $R_{dirty} = R_{dirty} - S_{dirty}$, $R_{clean} = R_{clean} \cup S_{clean}$
4. Return $\theta^{(T)}$

### 4.3.1 Selecting the Parameters

The proposed update policy makes intuitive sense. However, we have still have to set the parameters $\lambda$, $\alpha$, $\beta$. 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 $\gamma$ :** The first problem is that we have not explained how to pick the step size $\gamma$. In other words, how far should we travel in the gradient direction.

**Step Size $\alpha, \beta$ :** 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 gradient 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.

#### *4.3.2 Physical Design Considerations*

From a systems perspective, the important step in the update algorithm is step 3a. We have to query a sample of candidate data points from $R_{dirty}$ which can be expensive. As we clean data, we can maintain an index on which points are dirty and clean.

### 4.4 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 $\theta$, given the convex loss, and data. In SGD, we start with an abitrary initialization and iterate to convergence. In our case, we start with an initialization which is the dirty model, the model trained completely on the dirty data. Then, at each iteration we sample a mini-batch of data on which we apply our expensive data cleaning operations. The update rules in SGD mirror our update policy:

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

**Setting $\gamma$:** There is extensive literature in machine learning for choosing $\gamma$ appropriately. $\gamma$ 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 [7]. 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 $\gamma_t$, 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.

**Convergence Rate:** The convergence rates of SGD are also well analyzed [6,7,29]. 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 $O(\frac{\sigma^2}{\sqrt{bt}})$. $\sigma^2$ 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 $\sigma^2$, 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.

#### *4.4.1 Reducing the Variance $\sigma^2$*

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, there are a few observations about the data cleaning problem that suggest that we can do better if we carefuly select which points to clean. The key property that we need to preserve is unbiasedness (Proposition 1).

**Observation 1. Detection vs. Cleaning:** In many applications, enumerating the set of corrupted records is much easier than cleaning them. For example, we may be able to select the set of rows that have missing values but actually filling those missing values is expensive. Likewise, in the constraint literature, selecting a set of rows that have a violated constraint can be done in polynomial time, however, fixing the constraints is NP-Hard. In other words, cleaning a batch of data can be far more expensive than a scan. We can use this to our advantage to reduce the variance of the gradient without affecting our batch size.

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. $\square$

**Observation 2. Pre-processing is relatively cheap:** Data cleaning is often the most expensive step in the workflow, so optimizing for scan cost may lead to neglible overall time improvements. We can sacrifice a small overhead in pre-computation for each data point to determine its value to the model and select a sampling distribution accordingly. Intuitively, while each iteration has an increased cost, it also makes more progress towards the optimum.

In our problem, we are trying to select a sample $S_{dirty}$ from $R_{dirty}$. If we sample every element $i \in R_{dirty}$ with probability $p_i$, the question is how can we compute a set of $\{p_i\}$ such that variance of the gradient $\sigma^2$ is minimized. 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 $\mu$ is the mean $\hat{A}$, the sampling distribution that minimizes the variance of $\mu$, i.e., the expected square error, is $p(a_i) \propto a_i$.*

Proof. This proof follows from [**?**], we present it here for intuition. It is easy to verify that for sampling probabilities $p(a_i)$, that the unbiased estimate of the mean is:

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

The variance of this estimate is given by:

$$Var(\mu) = \mathbb{E}(\mu^2) - \mathbb{E}(\mu)^2$$

Since the estimate is unbiased, we can replace $\mathbb{E}(\mu)$ with the average of $A$:

$$Var(\mu) = \mathbb{E}(\mu^2) - \bar{A}^2$$

Since $\bar{A}$ is deterministic, we can remove that term during minimization. Furthermore, we can write $\mathbb{E}(\mu^2)$ as:

$$\mathbb{E}(\mu^2) = \sum_{i}^{N} \frac{a_i^2}{p_i}$$

Then, we can solve the following optimization problem over the set of weights $P = \{p(a_i)\}$:

$$\min_P \sum_{i}^{N} \frac{a_i^2}{p_i}$$

$$\text{subject to: } P > 0, \sum P = 1$$

Applying Lagrange multipliers, an equivalent unconstrained optimization problem is:

$$\min_{P>0,\lambda>0} \sum_{i}^{N} \frac{a_i^2}{p_i} + \lambda \cdot (\sum P - 1)$$

If, we take the derivatives with respect to $p_i$ and set them equal to zero:

$$-\frac{a_i^2}{2 \cdot p_i^2} + \lambda = 0$$

If, we take the derivative with respect to $\lambda$ and set it equal to zero:

$$\sum P - 1$$

Solving the system of equations, we get:

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

$\square$

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. We visualize this intuition in Figure 5, and essentially, this lemma shows that when estimating a function of a random variable $\mathbb{E}(f(X))$ the optimal sampling distribution is to align the samples as closely as possible with the function. This is insight leads to a direct higher-dimensional generalization, where at iteration $t$ we should sample the records in $R_{dirty}$ with probabilities:

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

This is an insight which has recently been of great interest in the Machine Learning community [29]. 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 $(x_i^{(clean)}, y_i^{(clean)})$. 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.

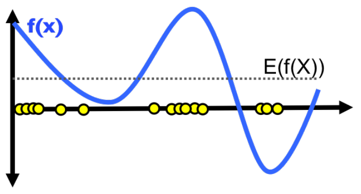

**Figure 5: The minimum variance estimate of an expected value of a function of a random variable $\mathbb{E}(f(X))$ is found by placing more samples (marked in yellow) on the "spikes" of the function.**

## 5. APPROXIMATING THE OPTIMAL DISTRIBUTION

This section describes our solution to problem of constructing and maintaining an inexpensive estimate of the optimal distribution.

### 5.1 Avoiding Regression

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. However, learning this function can be very expensive and is not guaranteed to be reliable. This is a high-dimensional regression problem which may have to learn a very complicated relationship between dirty and clean data. Such an estimator has a cold start problem, where if we have cleaned a very small number of errors, the estimator will be inaccurate. This problem becomes increasingly worse in higher dimensions as we need more data for an accurate estimate. 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.

## 5.2 Error Decoupling

We come back to the insight of the last section, where error detection is often much easier than error repair. For many types of data error, we can return a subset of corrupted attributes and in turn a subset of features that are corrupted; all without cleaning the data. Recall, that when we formalized the error detection problem, we ensured that associated with each $r \in R_{dirty}$ is a set of errors $e_r$ which is a set that identifies a set of corrupted columns. 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:

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

What we mean by corrupted features is that:

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

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

So basically, 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 non-linear 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. To understand why we care so much about the decoupling consider the following example:

EXAMPLE 5. *Suppose that we have just cleaned the following records 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 $r_3$.*

Decoupling allows us to treat errors conditioned on each feature independently. Alternative techniques such as taking the average change the gradient, without this property, would require to condition on every distinct set of corrupted features which can be combinatorially large.

## 5.3 Linear Approximation

We can approximate the gradient in such a way that we can do this. This approximation represents a linearization of the errors. 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. $\Delta x$ is a $d$ dimensional vector where each component represents a change in that feature and $\Delta y$ is an $l$ dimensional vector that represents the change in each of the labels.

Let us return to the linear regression example, where the gradient is:

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

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 **{{TR}}**.

## 5.4 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:*

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

*Similarly, for a label $i$:*

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

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 $\Delta_r$ 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:

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

## 6. RELAXING THE ENUMERATION ASSUMPTION

The crucial assumption of the previous sections is that we can enumerate a superset of the dirty records. An important aspect of our gradient update is partitioning the dirty and clean data. We aggregate an average gradient from both subpopulations when making our update. When our partitioning is perfect, at least all of the dirty data is selected in $R_{dirty}$, this estimate in unbiased. However, in some cases, characterizing this partioning can be difficult and it may be impossible to ensure this condition unless $R_{dirty} = R$ causing a loss in efficiency if errors are sparse. In this section, we explore the case when the partitioning is imperfect.

### 6.1 Why do we partition?

Partitioning serves two purposes: (1) it reduces the variance in the gradient update, and (2) it increases the fraction of actually dirty records in the candidate batch. The first objective follows directly from the analysis in the previous section (Section **??**). A good example of why we need the second objective is seen in the context of crowdsourcing. If we have a crowdworker clean records, we will have to pay them for the task whether or not the record required cleaning. Partitioning the data ensures that the only records cleaned are dirty records. Without partitioning, cleaning rare errors might be very wasteful.

### 6.2 Using a classifier

We retain the assumption that error detection is easier than repair. In particular, we argue that it is more feasible to train a classifier to predict error occurance than to train a regression to fix the consequences of error. Let there be $u$ types of errors in the database. Suppose, we have a multiclass classifier $\kappa$ (e.g. SVM) that classifies every record $r$ into $u + 1$ classes (the $u$ error types or not dirty).

To make such a classifier useful, we have the modify our problem formulation:

**Error Detection:** To select $R_{dirty}$, we select the set of records for which $\kappa$ give a positive error classification (i.e., one of the $u$ error classes). 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%.

**Error Repair:** When an example $(x, y)$ is cleaned, the repair step also has to provide a label of to which of the $u + 1$ classes it belongs. As more data is cleaned, this data becomes training data for the classifier.

### 6.3 Error Estimate

Our model update framework still holds in this setting, however, our error estimation framework requires some modification. In our derivation, we showed that our error estimate was a linearization of the conditional expectation of the gradient. Instead of conditioning on the features that are corrupted, we condition the error classes. So for each error class, we compute a $\Delta_{xu}$ and $\Delta_{yu}$. These are the average change in the features given that class and the average change in labels given that class respectively.

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

## 7. 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.

### 7.1 Experimental Setup and Notation

Every experiment has two steps: data cleaning and model evaluation. We evaluate the data cleaning on one metric:

**Cleaning Efficiency.** Let $K$ be the number of samples processed by the algorithm, and $K'$ be the number of samples that were actually dirty. The cleaning efficiency is $\frac{K'}{K}$.

In our experiments, we explore three classification models: L1-Hinge Loss SVM, Logistic Regression, and Thresholded Linear Regression. We evaluate the trained models on the following metrics:

**Relative Model Error.** Let $\theta$ be the model trained on the dirty data, and let $\theta^*$ 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 $\theta$ be the model trained on the dirty data, and let $\theta^*$ 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 $T(\theta^*) - T(\theta)$

#### *7.1.1 Scenarios*

We apply these models in the following scenarios:

**Housing:** In this dataset, our task is to predict housing prices from 13 numerical and categorical covariates. There are 550 data points in this dataset. The model is a Logistic Regression classifier which predicts if the house price is greater than \$500k.

**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. This dataset is unique because the classification is hard with linear predictors. 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.

## 7.2 Experiment 1. Effect of Cleaning

Before we evaluate ActiveClean, we first evaluate the benefits of cleaning on our 4 example datasets. We first explore this problem without sampling to understand which types of errors are amenable to data cleaning and which are better suited for robust statistical techniques. We compare 4 schemes: (1) cleaning, (2) adding an L2 regularizer tuned to maximal accuracy with a grid search, (3) discarding the dirty data, and (4) baseline of no cleaning.

We corrupted 5% of the training examples in each dataset. We corrupted these data in two different ways.

**Random errors:** 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 errors:** We simulated innocuous looking (but still incorrect) systematic errors. We trained the model on the clean data, find the most important feature (highest weighted). We sort examples but this feature and corrupt the top 5% of examples with the mean value for that feature.

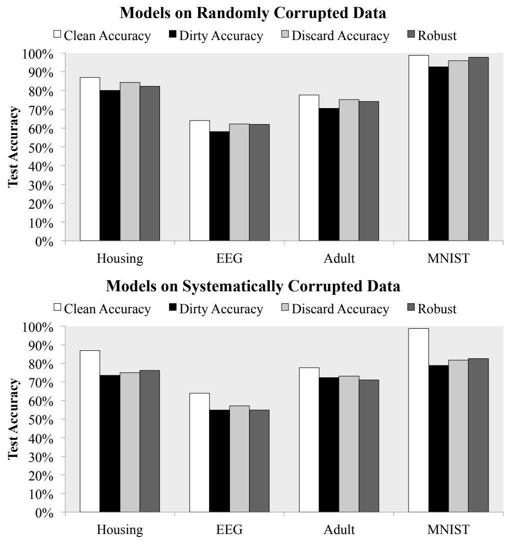

**Figure 6: Robust techniques work best when corrupted data are random and look atypical. Data cleaning can provide reliable performance in both the systematically corrupted setting and randomly corrupted setting.**

In Figure 6, we present the results of this experiment. As we argued in this paper, the robust method performs well on the random high-magnitude outliers, however, falters on the systematic corruption. Interestingly enough, in the random setting, discarding dirty data also performs well. However, when errors are systematic data cleaning is the most reliable option across datasets. In the MNIST dataset, we see a particularly significant effect of systematic corruption where the test accuracy drops from nearly 98% to 78%. Multiclass classification is particularly sensitive to systematic corruption when the corruptions can make classes ambiguous (e.g. reconizing a "4" and a "9"). The problem is that a priori, we do not know if data error is random or systematic. While data cleaning requires more effort, it provides benefits in both settings.

## 7.3 Experiment 2. Prioritization

The next set of experiments evaluate different approaches to cleaning a sample of data. In this set of experiments, we use the random errors generated above.

### 7.3.1 2a. Alternative Algorithms

In our first prioritization experiment, we evaluate the samples-to-error tradeoff between three alternative algorithms:

**SampleClean (SC):** In SampleClean, we do not use a gradient update and instead take a sample of data and train the model to completion on the sample.

**Active Learning (AL):** In Active Learning, we do not consider the effect of "data cleaning" and prioritze points by their dirty gradient value. We do, however, do this iteratively and update the model.

**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 error estimation.

In Figure 7, we present our results on Housing, Adult, and EEG. We find that ActiveClean gives its largest benefits for small sample sizes (up-to 12x). 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 impact estimates and error partitioning allow us to beat Active Learning on all three of the datasets.

### 7.3.2 2b. Source of Improvements

Throughout the paper, we proposed numerous optimizations. 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 7 that corresponds to 10% of the data cleaned (55 for Housing, 4555 for Adult, 150 for EEG) and compare the performance of ActiveClean with and without various optimizations. We denote ActiveClean without partitioning as (AC-P) and ActiveClean without partitioning and importance sampling as (AC-P-I). In Figure 8, we plot the relative error of the alternatives w.r.t to the optimized version of ActiveClean. Partitioning 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 partitioning we still see some improvements since our importance sampling relies on error impact estimates that judge

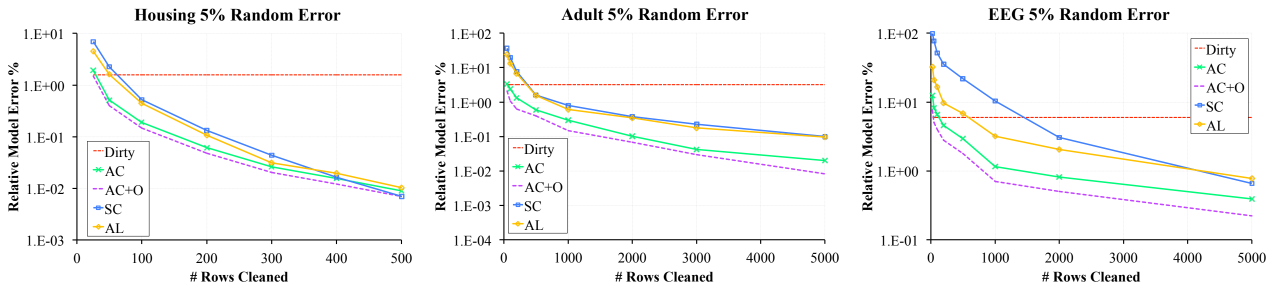

**Figure 7: ActiveClean converges with a smaller sample size to the true result in comparison to Active Learning and SampleClean.**

how valuable a point is to the clean model rather than the dirty model in Active Learning. Not surprisingly, when we remove both these optimizations, ActiveClean is comparable or slightly worse than Active Learning.

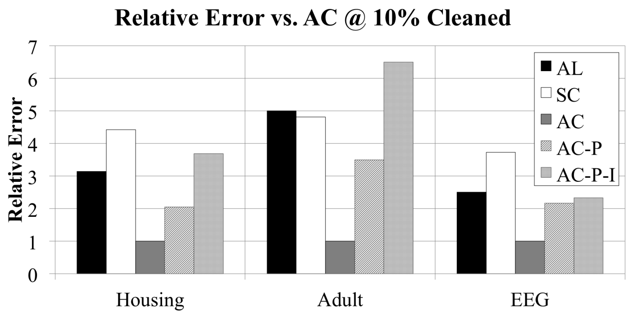

**Figure 8: We clean 10% of the data with the alternative algorithms and also include variants of ActiveClean with optimization removed. We plot the relative error w.r.t the optimized ActiveClean. Both partitioning and importance sampling lead to significant reductions in error.**

We evalue Active Learning and ActiveClean to better understand this relationship. In Figure 9, we vary the biasing effect of our random corruptions. That is, we start with zero mean noise and increase the mean value and variance of the noise. Since Active Learning uses the gradient, if there is zero mean noise, in expectation, the dirty data and clean data are the same. However, as the bias increases, the fact that Active Learning prioritizes w.r.t to the dirty data matters more and becomes increasingly erroneous w.r.t to ActiveClean.

### 7.3.3 2c. Error Dependence

Both Active Learning and ActiveClean outperform SampleClean in our experiments. In our next experiment, 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 rate of random error, thus making the initialization more and more arbitrary, and measure the relative performance between SampleClean and ActiveClean. Since SampleClean only acts on a clean sample of data, it is robust to data error. So at some point, the errors in the data are so significant that training a model on a small but clean sample of data is more efficient than iteratively updating the dirty model.

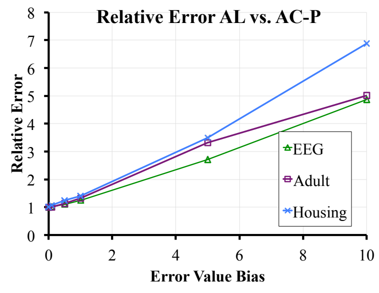

**Figure 9: As we increase the biasing nature of the corruption, Active Learning is increasingly erroneous w.r.t ActiveClean.**

In Figure 10, we present the results from this experiment. We corrupt entries from the data matrix of the Adult dataset at random (probability on plotted on the x-axis). Then, we measure the number of records we need to clean before we have a relative error of 0.1%. We find that at about 30% corruption rate, SampleClean is more accurate than ActiveClean. Since the Adult dataset has 12 features, a 30% corruption rate corresponds to each example with 3.6 features incorrect on average. We optimized ActiveClean for sparse and relatively small errors but it still shows reasonable performance even in this highly erroneous setting. At higher corruption rates, ActiveClean requires more than one epoch to converge to an accurate answer which requires cleaning almost all of the data.

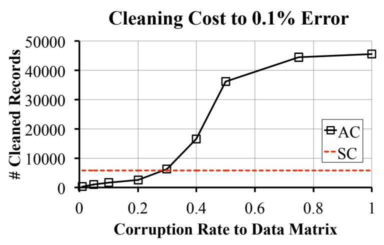

**Figure 10: We corrupt an increasing number of entries in the data matrix. At about 30% corrupted, ActiveClean is no longer more efficient than SampleClean.**

### 7.3.4 2d. Testing Accuracy

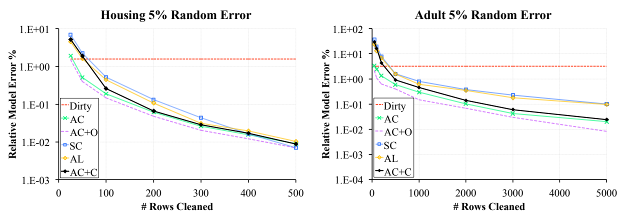

**Figure 12: While initially ActiveClean is comparable in performance to Active Learning, as the classifier improves, ActiveClean converges faster than Active Learning and SampleClean.**

In the previous experiments, 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 11, we present the results for the three datasets: Adult, Housing, and EEG. We find that in two of the datasets, Housing and Adult, ActiveClean converges to clean test accuracy faster than the alternatives.

However, there is a curious negative result with the EEG dataset that we would like to highlight. We find that even though ActiveClean has significantly lower model error (Figure 7), this does not correspond to as significant of an increase in test accuracy. We speculate this is due to the inherrent hardness of the EEG classification problem. ActiveClean may encourage overfitting at intermediate results for hard classification tasks. The solution to this problem may be to add additional regularization, thus actually changing the optimization problem. We hope to explore this problem in further detail in future work.

## 7.4 Experiment 3. Predicates vs. Classification

### 7.4.1 *3a. Basic Performance*

In the next set of experiments, we explore the error partitioning in more detail. We presented two models for error sampling, one where we are given a set of candidate dirty records through a predicate and one where we have to learn this predicate as we clean. In Figure 12, we overlay the convergence plots in the previous experiments with a curve (denoted by AC+C) that represents ActiveClean using a classifier instead of an error predicate. We use an SVM classifier to predict errors. We find an interesting tradeoff where initially ActiveClean is comparable to Active Learning (explanation for why seen in Figure 8), as our classifier becomes more effective the partitioning improves the performance.

### 7.4.2 *3b. Classifiable Errors*

Using a classifier to partition errors depends on errors that can be classified. For example, random errors that look like other data may be hard to learn. As errors become more random, the classifier becomes increasingly erroneous. We run an experiment where we start with the systematic errors described earlier. With probability $p$, we increasingly make these errors more random. We compare these results to AC-P where we do not partition the errors. In Figure 13, we plot the error reduction using a classifier. We find that when errors are about 40% random then we reach a break even point where the user is better of not partitiong the data since the errors introduced by incorrect classification are more than the error reductions.

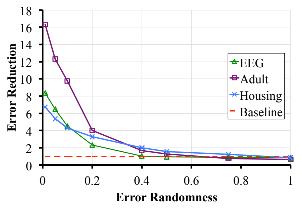

**Figure 13: Errors that are less random are easier to classify, and lead to more significant reductions in relative model error.**

## 7.5 Experiment 4. Price of a Scan

In our fourth set of experiments, we evaluate one of the computational premises of our work. We argue that the price of a scan of data is cheap compared to data cleaning. The overheads of partioning the dirty and clean data and calculating the sampling distribution should be small in comparison to the faster convergence of our model. **{{How do we quantify the costs of data cleaning?}}**

## 7.6 Real Scenarios

We evaluate ActiveClean in two real scenarios: base data replacement and constraint-based data repair.

### 7.6.1 *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.

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. In Figure 14, we visualize the corruptions. We constructed these features to mimic the random vs. systematic errors that we studied before. The 5x5 block removal behaves much more like a systematic error. Typical image processing features are based on edges and corrupting edges leads to ambiguities (e.g., is the image in Figure 14 a 4 or a 9). On the other hand, the making the image fuzzy is more like a random error.

In Figure 15, 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 error. This is because SampleClean does not depend on any quantity derived from the dirty data. Next, the gap between the theoretical AC+O and

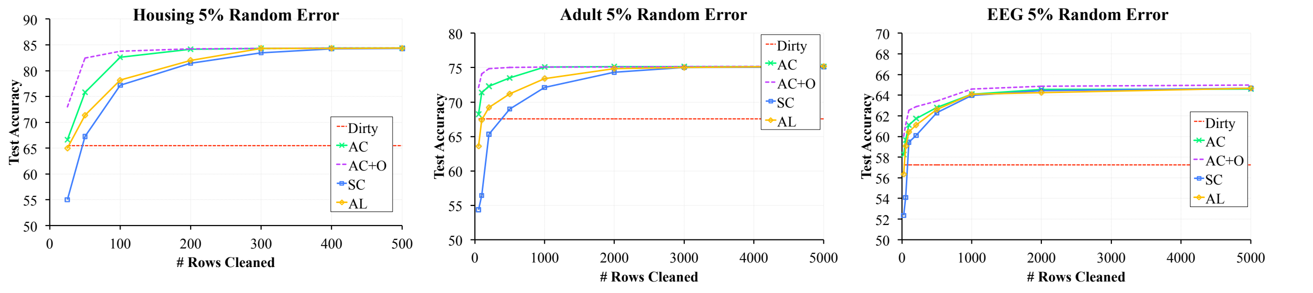

**Figure 11: ActiveClean converges with a smaller sample size to the maximum test accuracy in comparison to Active Learning and SampleClean.**

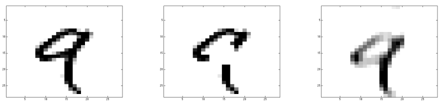

**Figure 14: We experiment with two forms of corruption in the MNIST image datasets: 5x5 block removal and making the images fuzzy.**

what we are able to achieve is much larger in this dataset. We speculate that this is because classifying errors in a 784 dimensional space is much harder than in our previous experiments, and this in turn leads to reduced accuracy in impact estimation.

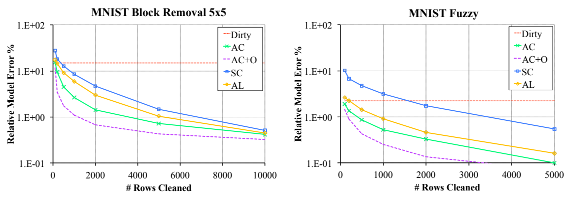

**Figure 15: For both types of corruption, ActiveClean converges faster than Active Learning and SampleClean.**

**{{CFD: Adult}}**

## 8. 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 [29]. 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., [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 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 [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. ActiveClean 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 [9,25]. 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 [16], 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 [3]. 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 [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 [4]. 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.

## 9. 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) [12] to extend this technique to kernelized methods.

## 10. REFERENCES

[1] Big data's dirty problem.

[2] 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.

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

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

[5] L. Bertossi, S. Kolahi, and L. V. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Theory of Computing Systems*, 52(3):441–482, 2013.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

[17] M. Nikolic, M. Elseidy, and C. Koch. Linview: incremental view maintenance for complex analytical queries. In *SIGMOD Conference*, pages 253–264, 2014.

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

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

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

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

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

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

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

[25] 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.

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

[27] 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.

[28] 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?

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

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