ActiveClean: Training Models on Dirty Data Under Time Constraints

ABSTRACT

Databases are susceptible to various forms of corruption, or *dirtiness*, such as missing, incorrect, or inconsistent values. Increasingly, modern data analysis pipelines involve Machine Learning for predictive models. In this paper, we propose 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.

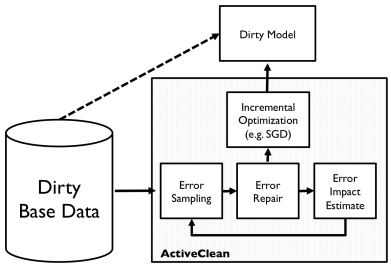

Figure 1: ActiveClean (ActiveClean) is an anytime framework from training models on dirty data. Expensive data transformations prior to model training are budgeted with a non-uniform sampling that prioritizes examples that are most likely to change the model.

1. INTRODUCTION

Databases are susceptible to various forms of corruption, or *dirtiness*, such as missing, incorrect, or inconsistent values. Numerous surveys of industry have shown that dirty data are prevalent [22], and there is now a growing industry around processing dirty data at scale [1]. Analysts are increasingly deriving data from inherently error-prone processes such as extracting structured data from websites, synthesizing data from multiple networked sensors, and linking entities in disparate data sources. As these data grow, new methodologies for scalable and reliable analysis in the presence of errors are required.

Increasingly, modern data analysis pipelines involve Machine Learning for predictive models. The endpoint of these pipelines can be any number of "data products", such as recommender systems, spam detectors, and forecasting models, all of which can be very sensitive to data quality [26]. When data error is systematic, or correlated with the data, errors can significantly bias predictions by a model. For example, in a recommender system, we may find that all users from one region have a missing age attribute. In this setting, discarding corrupted data or ignoring the problem can make predictions for the affected subpopulation untrustworthy. A more sophisticated approach is to apply robust statistical methods, but these are designed to mitigate the effect of high-magnitude random outliers, and not compensate for systematic errors.

For some types of systematic error, we can apply data cleaning, which is an extensively studied field (see Rahm and Do [20] for a survey). Instead of avoiding the problem, cleaning works by repairing (or approximately repairing) the corruption. However, cleaning large data can be expensive, both computationally and in human effort, as an analyst has to program repairs for all errors manifest in the data [12]. In some applications, scripted data transformations may not be reliable necessitating the use of even costlier crowdsourcing techniques [10,18].

An emerging solution to the growing costs of data cleaning is sampling [25] where the analyst cleans a small sample of data and can estimate the results of aggregate queries.Analysts can sample a large dataset, prototype changes on the sample, and evaluate whether these changes have the desired affect. The case for sampling is analogous to arguments for Approximate Query Processing (AQP) [2], where a timely approximate answer is more desirable than an exact slow answer. Sampling provides a flexible tradeoff between cleaning cost and query result accuracy for aggregate queries, but the question we explore in this paper is how this tradeoff extends to Machine Learning.

Machine Learning is far more sensitive to sample size (training data) than aggregate queries. However, there are a few observations that make this problem tractable. Sampling is naturally part of large-scale Machine Learning, as stochastic optimization methods start at an arbitrary initialization and calculate updates by drawing training examples at random. Even with clean data, 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) [**?**]. In our setting, we do not have to start with an arbitrary initialization. Errors often happen in batches, often tightly clustered, and affect a small number of features. This means that a dirty model may be relatively "close" to the clean model albeit with bias in a few features. Hypothetically, if we had a clean dataset, and were to apply stochastic optimization using the dirty model as an initialization, we can get a highly accurate result with processing far less than an entire epoch. This insight inspires the key idea of this paper,

namely, that we can clean data as and when required by the optimization allowing us to clean far less than the entire dataset.

This integrated model training and data cleaning architecture leads to new optimization opportunities. Since data cleaning adds a very significant cost to each model update, we can run pre-computations at each iteration to select which examples to clean because the additional overhead is offset by reductions in cleaning cost. 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 [7,21]. In Active Learning, we often sequentially select the most important points to label [21]. However, the key new challenge in data cleaning is that features and examples that may look unimportant in the dirty data may be important in the clean data and vice versa. We partition data that we believe is clean from the dirty data, and estimate the effects of cleaning on the dirty data based on prior examples. The resulting algorithm selects examples valuable to the clean model with higher probability and as an analyst cleans data we further guide the cleaning.

In this paper, we propose ActiveClean (ActiveClean), an anytime framework for training Machine Learning models with data cleaning. 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. An *error sampling* module selects a small sample of data from the corrupted dataset to clean. We apply the repairs to this small sample with the *error repair* module, and use an incremental optimization method (e.g., Gradient Descent) to update the dirty model. 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.

In summary, our contributions are

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

2. BACKGROUND

2.1 Motivating Example

To motivate our solution, we present a running example scenario inspired by one of our experimental datasets. In this problem, an analyst is training a classifier to predict the occurrence of seizures based on 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* θ from training examples. A common theoretical framework in Machine Learning is empirical risk minimization with linear predictors. We start with a set of training examples $\{(x_i, y_i)\}_{i=1}^N$ on which we minimize an loss function ϕ at each point parametrized that is parametrized by θ.

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

For example, in a linear regression ϕ is:

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

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

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

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

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

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

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

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 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 $T(\theta^*) - T(\theta)$

2.4 Two perspectives on error

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

Database Viewpoint. The Database literature treats data error as a data management problem. 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.

Machine Learning Viewpoint. 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. Some errors require expensive cleaning procedures, increasingly using the crowd, and we joint have a time budget on cleaning and analysis. ActiveClean has four main components: Error Sampling, Error Repair, Incremental Model Update, and Error Impact Estimation. We clean small samples of candidate dirty data and update the model accordingly. As we clean more data, we prioritize sampling to those data that are most valuable to the model if cleaned. This differentiates ActiveClean from the Active Learning frameworks that are widely applied in the data cleaning literature. ActiveClean prioritizes cleaning with respect to an estimated impact on the clean model.

3. SYSTEM ARCHITECTURE

3.1 Problem Formulation

We revisit the architecture diagram in Figure 1, and formalize the problems addressed in each one of the stages. ActiveClean is a data cleaning methodology that inlines data cleaning with model training, and we formalize the conditions on the data cleaning operations that can fit in our framework. 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.

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

Every feature and label in (X, Y) should be derived from exactly one column in the base data (i.e., a many-to-one relationship). We have a data cleaning operation $\mathcal{D}$, that cleans R:

$$F(\mathcal{D}(R)) \to (X_{clean}, Y_{clean})$$

3.1.1 Error Detection

The first step of ActiveClean is error detection. We have to select a set of candidate records to be cleaned from the base data. Formally, given a relation R, we select a subset of records to be cleaned R_{dirty}. R_{dirty} must contain at least all of the records that are actually dirty. Associated with each $r \in R_{dirty}$ is a set of errors e_r which tells us which columns are corrupted.

3.1.2 Error Sampling

Given a set of candidate dirty records, the error sampling step selects a sample to clean. Records have to be sampled independently, i.e., sampling record i does not affect the probability of sampling record j. The input to this step is a sample size K and the output is tuple of the a sample of K records and their sampling probabilities. The research problem that we explore in this work is how to construct this sampling distribution and feedback updates from the latest iteration.

PROBLEM 1 (SAMPLING PROBABILITY). *Given the current best model estimate θ, data matrices (X, Y), for each record i calculate a sampling probability p_i.*

3.1.3 Error Repair

Given a set of sampled records, we apply data cleaning to this sample. We define data cleaning as transformations that happens to the base data that result in changes to the feature vector. There is at most a one-to-one relationship between features in the cleaned data and the dirty data. We require the following conditions on the data cleaning:

1. Single Operation. A dirty record is completely cleaned after applying the Error Repair module.
2. Persistence. Once cleaned there is no future operation that can invalidate the cleaning rendering the record dirty.
3. Sample-Local Write. The data cleaning must only modify the records in the sample.

The output of this step are cleaned set of features vectors $\{(x_{clean}, y_{clean})\}_i^K$.

3.1.4 Incremental Model Update

Given the sample of cleaned examples $\{(x_{clean}, y_{clean})\}_i^K$ and the sampling probabilities, the next problem that we address is to update the model.

PROBLEM 2 (INCREMENTAL UPDATE). *Given the current best model estimate θ, a clean sample $\{(x_{clean}, y_{clean})\}_i^K$, and a set of sampling probabilities $\{p\}_i^K$, we update the best model estimate from $\theta \to \theta'$.*

3.1.5 Error Impact Estimate

Once we compute θ', we feed this information back to update and guide the sampling distribution. We want to calculate our sampling distribution with respect to the cleaned data, so for each remaining uncleaned record, we have an estimate of its cleaned value $(\hat{X}, \hat{Y})$. This estimate need not be very accurate as it is only used to select the sampling probabilities.

PROBLEM 3 (ERROR IMPACT ESTIMATE). *Let (X_{dirty}, Y_{dirty}) be the data submatrices of the dirty data, E_r be the error identifying set, and (X_{clean}, Y_{clean}) be the submatrices for the cleaned data. Given (X_{clean}, Y_{clean}), we learn a function f that maps an $(x, y) \in (X_{dirty}, Y_{dirty}) \mapsto (\hat{x}, \hat{y})$.*

3.1.6 Summary

ActiveClean explores a different data cleaning architecture where data is progressively cleaned while a model is being trained. We can feed results back to guide and prioritize data cleaning adaptively.

3.2 Detection and Repair Models

There are two steps, Error Detection and Error Repair, that are dependent on the data cleaning operation. 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.

3.2.1 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 [9,13,27]. 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 [9]. Due to the hardness of automatic repair, heuristics are typically used so the fixes can be unreliable. Therefore, recently proposed systems like Guided Data Repair [27], 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 Σ be a set of constraints on the relation $\mathcal{R}$. The allowed constraint classes are Matching Dependencies [5], Conditional Functional Dependencies [9], and Denial Constrains [13]. We additionally require that Σ is transitively closed and the constraints are on disjoint attributes. 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 is a subset of Σ for each record that is violated.

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 [9] or we can use human corrections as in [27]. 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.*

3.2.2 Value Filling

Value filling is another data cleaning operation that is well studied in the literature. Park and Widom proposed the system called CrowdFill [18], 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 ϕ 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.*

3.2.3 Entity Resolution

Another common data cleaning task is Entity Resolution [10,14]. 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. SAMPLING AND MODEL UPDATE

In this section, we describe how we combine the different components of our system. If we formulate the model update step in ActiveClean as a Stochastic Gradient Descent step, then it provides us with a framework to derive an update policy (Problem 2), and in turn how that informs our sampling distribution (Problem 1).

4.1 Stochastic Gradient Descent

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

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

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

We treat the model update problem problem as a SGD problem. 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. We update the model using an SGD step (see [6] for a survey of Incremental optimization methods).

SGD Model Update (Problem 2):

1. For a sample of data S from R_{dirty}, calculate the expected gradient with respect to the sampling distribution (Section 4.2). Call this value $g_d^{(t)}$.
2. For the clean data $C = R - R_{dirty}$, calculate the population mean gradient $g_c^{(t)}$.
3. Apply gradient descent using the gradient $g = \frac{|C| g_d + |S| g_c}{|C| + |S|}$

We stop when the budget is reached returning the best model at that time. We argue that if the initial dirty model is relatively close the true model, a small number of gradient steps with clean data can compensate for the errors.

Here, we can draw an analogy to Materialized View maintenance, since after all, a model parameterized by θ 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 [16].

However, it is important to note, that this analogy only goes so far. Most incremental view maintenance architectures can propagate their updates in a single pass of the data. In our setting, this is not guaranteed. If the dirty model greatly differs from the clean model a single epoch may not be enough to reconcile the differences. In this case, to avoid cleaning the entire data, it may be better to take a clean sample of data and train the model completely rather than doing it in an iterative fashion. We explore this tradeoff in our experiments {{[?]}}.

4.2 Sampling Distribution

In the Machine Learning and Optimization literature, SGD algorithms are optimized to avoid scanning the entire data. However, in the data cleaning setting, there is a new bottleneck. Data cleaning is often the most expensive step in the workflow, so optimizing for scan cost may lead to neglible overall improvements. Instead, in ActiveClean, we 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.

To construct this distribution, we have to analyze a single SGD iteration. In essence, the key property that allows SGD to converge is that the gradient step $g^{(t)}$ defined as:

$$g^{(t)} = \frac{1}{\mid S^{(t)} \mid} \sum_{i \in S^{(t)}} \nabla \phi$$

is an unbiased estimate of the "true" gradient (i.e., over all training examples):

$$\mathbb{E}(g^{(t)}) = \frac{1}{N} \sum \nabla \phi$$

With uniform sampling this basic SGD algorithm has performance lower bound [1]:

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

where $\mathbb{E}(\|\nabla \phi\|_2^2) = \sigma^2$. If we choose our sampling technique carefuly, we can reduce the σ^2 term in the error bound. We preserve the same expected gradient at each step $\mathbb{E}(g^{(t)})$ but possibly have a lower variance.

The technique that we will apply is called importance sampling. Importance sampling is a technique that allows us to draw a batch from a non-uniform distribution but reweight the result to give us an unbiased estimate of the gradient. It is established [28] that the optimal sampling distribution (lowest variance) is:

$$p_i \propto \|\nabla \phi(x, y, \theta^{(t)})\|$$

There is immediate a chicken-or-egg problem with this optimal distribution, namely it is optimal if we know the gradient; which is exactly what we are trying to estimate. This problem is further complicated by the fact that we do not know the actual value of the record until we clean it. In our setting, we are actually calculating the gradient of the cleaned data:

$$\phi(x_{clean}, y_{clean}, \theta^{(t)})$$

To address these problems, we will show how we can use the structure of the data cleaning specifications in the previous section to compute low cost estimates of the cleaned value. In other words, let $e(\cdot)$ be an estimator function, then, we calculate the sampling distribution:

$$p_i \propto \|\nabla \phi(e(x, y), \theta^{(t)})\|$$

The gradient update becomes:

$$g_d^{(t)} = \frac{1}{\mid S^{(t)} \mid} \sum_{i \in S^{(t)}} \frac{1}{p_i} \nabla \phi(x_i, y_i, \theta^{(t)})$$

In our analysis, we will describe the conditions under which this distribution has lower variance than uniform sampling.

4.3 Summary

In summary, while we still have to discuss how we find the estimator function $e(\cdot)$, we can describe the basic processing workflow of ActiveClean.

1. Initialize with $\theta^{(0)}$ as the dirty model, T iterations, with a batch size B
2. For rounds i=1...T
   (a) Sample B candidate dirty data points with probabilites as described.

[1] there is a technical condition that this bound holds for strongly convex functions

(b) Apply data cleaning to the sample of data.

(c) Apply weighted gradient descent to update the model.

3. Return $\theta^{(T)}$

5. APPROXIMATING THE OPTIMAL DISTRIBUTION

This section describes our solution to problem of constructing and maintaining an inexpensive estimate of clean value $e(\cdot)$. To do this accurately, is a challenging regression problem. However, we only use this estimate to construct a sampling distribution, so a coarse-grained estimate suffices. We show how we can exploit the structure of the data cleaning operations to find this coarse-grained estimate.

5.1 Error Decoupling

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. Since there is a one-to-many relationship between columns and features, each erroneous column affects a disjoint set of features. We will show how we can construct a decoupled estimate, one where we independently estimate for each colum and aggregate them together.

Property represents a linearization of the errors, and can be addressed with a first order approximation of the expected gradient. The expected gradient is defined as:

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

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 actually 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:

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

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

5.2 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.

LEMMA 1 (SINGLE FEATURE). *For a feature i, we average all records cleaned that have an error for that feature, weighted by their sampling probability:*

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

Similarly, for a label i:

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

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

THEOREM 1 (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 address the **Sampling Distribution and Impact Estimate (Problem 1 and 3):**

$$p_r \propto \|\nabla\phi(x, y, \theta^{(t)}) + \frac{\partial}{\partial X}\nabla\phi \cdot \Delta_r + \frac{\partial}{\partial Y}\nabla\phi \cdot \Delta_r\|$$

5.3 Algorithm

Now that we know how to feedback the error estimates $c(\cdot)$, we describe the entire workflow of ActiveClean:

1. Initialize with $\theta^{(0)}$ as the dirty model, T iterations, with a batch size B
2. Initialize all $\Delta = 0$
3. For rounds i=1...T
   (a) Sample B candidate dirty data points with probabilites as described.
   (b) Apply data cleaning to the sample of data.
   (c) Apply weighted gradient descent to update the model.
   (d) Update Δ for each feature.
4. Return $\theta^{(T)}$

5.4 Physical Design Considerations

From a systems perspective, the important step in this algorithm is step 3a. We have to query a sample of candidate data points from R_{dirty} which can be expensive. For eample, if we have a set of constraints, we would have to re-evaluate the violated constraints at each iterations. This is why we make assumptions about the persistence of data repairs in our problem formalization. If we make this assumption, we can run the query once at initialization and index the dirty records. Then, as we clean data, we can maintain the index.

5.5 Comparative Analysis

There is well-studied history of Machine Learning and Data Cleaning with budgets and prioritization. Suppose, we have a budget of cleaning $k \ll N$ records. Non-iterative techniques include the traditionally studied robust Machine Learning. These techniques handle only a small set of systematic error. If we want to handle a larger space of data systematic error, then we need some sort of a data cleaning approach. We could apply SampleClean [25] in a non-iterative fashion which takes a random sample of data, applies data cleaning, and trains the model to optimum on the sample.

An alternative iterative technique is Active Learning to prioritize which data to clean. Prior work in Active Learning is agnostic to data error so the prioritization will be with respect to the dirty data instead of the clean data. Active Learning may give us a benefit when the statistics of the dirty data are a good proxy for the clean

Technique	Systematic Errors	Preferred Regime	Cost per Iteration	Overall Cleaning Cost
Robust Machine Learning	Incomplete	High-Magnitude Random Outliers	-	None
SampleClean	Yes	Very Biased	-	Sample
ActiveLearning	Yes	Minimal Error	O(N)	Reduced
ActiveClean +Uniform	Yes	Sparse, high-variance errors	O(N)	Reduced
ActiveClean +Importance	Yes	Sparse, localized errors	O(N)	Greatly Reduced

data. In fact, expected gradient length as an Active Learning criterion has been studied before [21], but in this work we devise a technique to compensate for data error. We also leverage the structure of the data cleaning setting to make more progress at each iteration than a typical Active Learning workflow by averaging the gradients of newly cleaned batches of data with data that we know is clean. We summarize the salient features in Table(**??**).

{{**SK will flesh out analysis here next week.**}}

{{**Beats SampleClean in ideal case**}}

LEMMA 2. *In the oracular case, where we know the clean data in advance, importance sampling gives a strictly tighter error bound than uniform sampling.*

{{**Beats Active Learning in ideal case**}}

LEMMA 3. *In the oracular case, where we know the clean data in advance, the gradient step made by ActiveClean gives a strictly lower error bound than a naive random sample.*

{{**Gains are preserved in practice**}}

LEMMA 4. *In the real case, where we have to estimate the clean data, importance sampling gives a tighter error bound when the variance of the estimates e is less than the variance of the gradients.*

6. IMPERFECT PARTITIONING

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 5.5). The second objective is best understood 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 Revised Formulation

Even in this setting, our framework is still useful, however we have to modify the formulation of the problem. Instead of assuming that we know which records are dirty in advance, we make the assumption that we know which attributes are dirty in advance.

Error Repair: When an example (x, y) is cleaned, for each dirty attribute a cleaned becomes a training example for an error classifier κ_a. This classifier, e.g., Support Vector Machine, that learns which points are dirty and which points are cleaned based on the results of operation.

Error Detection: To select R_{dirty}, we select the set of records for which any of the error classifiers κ_a give a positive error classification. 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%.

6.3 Revised Algorithm

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

1. Initialize with $\theta^{(0)}$ as the dirty model, T iterations, with a batch size B
2. Initialize all $\Delta = 0$
3. Initialize $R_{dirty} = R$
4. For rounds i=1...T
   (a) Sample B candidate dirty data points with probabilites as described.
   (b) Apply data cleaning to the sample of data.
   (c) Apply weighted gradient descent to update the model.
   (d) Update Δ for each feature.
   (e) Update classifiers κ, and apply the classifier to prune the set R_{dirty} to records clean up to sufficent confidence.
5. Return $\theta^{(T)}$

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 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 $T(\theta^*) - T(\theta)$

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.

In Figure 2, 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

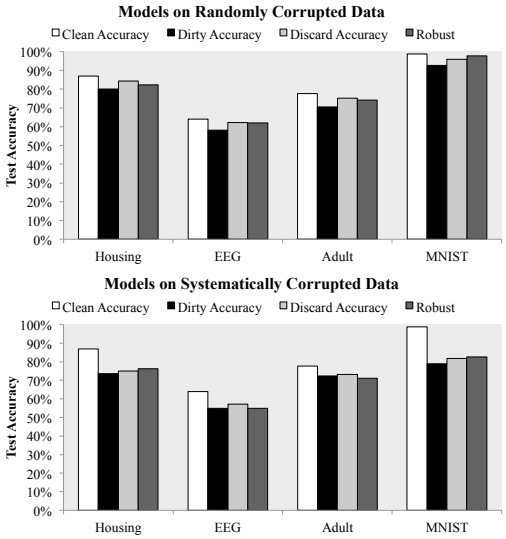

Figure 2: Robust techniques work best when corrupted data are random and look atypical. Data cleaning can provide reliable performance in both the systematically corrupted setting and randomly corrupted setting.

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 3, 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 3 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.

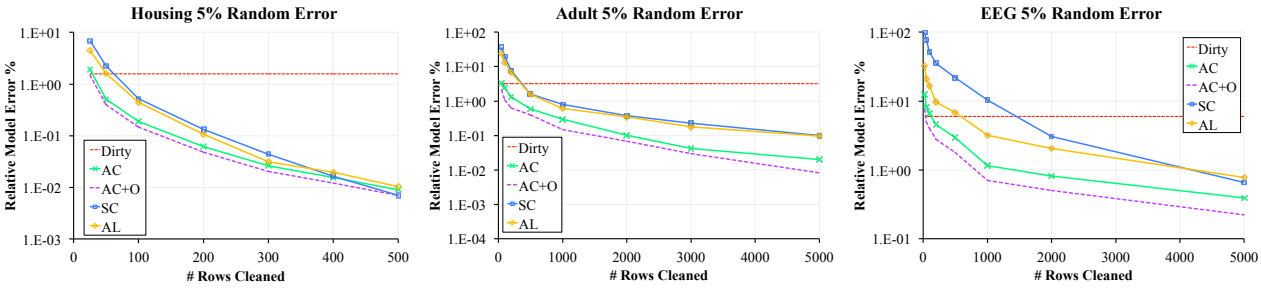

Figure 3: ActiveClean converges with a smaller sample size to the true result in comparison to Active Learning and SampleClean.

We denote ActiveClean without partitioning as (AC-P) and ActiveClean without partitioning and importance sampling as (AC-P-I). In Figure 4, 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 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.

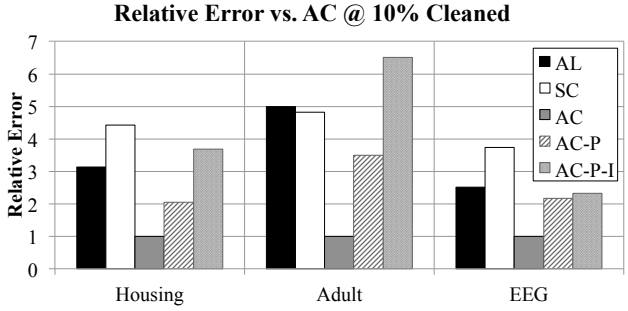

Figure 4: We clean 10% of the data with the alternative algorithms and also include variants of ActiveClean with optimization removed. We plot the relative error w.r.t the optimized ActiveClean. Both partitioning and importance sampling lead to significant reductions in error.

We evalue Active Learning and ActiveClean to better understand this relationship. In Figure 5, 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

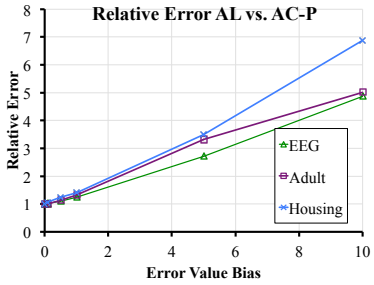

Figure 5: As we increase the biasing nature of the corruption, Active Learning is increasingly erroneous w.r.t ActiveClean.

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.

In Figure 6, 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.

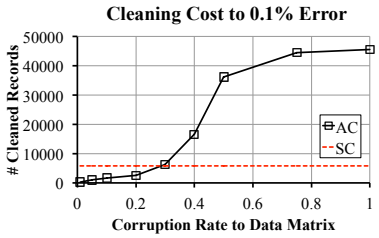

Figure 6: We corrupt an increasing number of entries in the data matrix. At about 30% corrupted, ActiveClean is no longer more efficient than SampleClean.

7.3.4 2d. Testing Accuracy

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 7, we present the results for the three datasets: Adult, Housing, and EEG.

7.4 Experiment 3. Error Predicates vs. Classification

{{Evaluate how much we lose}}

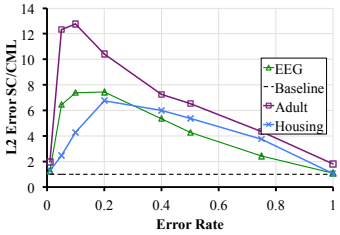

Figure 8: {{TODO}}

7.5 Error Types

Now, we evaluate the tradeoff space between random errors and systematic errors. In the first plot (Figure 9a), we show the relative accuracy between AC+U and AC as a function of predicate randomness. In other words, we start with the 5% systematic errors that we had before, and increasingly make them more random (i.e., corrupt a random point with ϵ probability). As the errors become more random, AC has less of a benefit in comparison to uniform sampling. Next, in Figure 9b, we explore the opposite tradeoff. We start with random errors and vary the magnitude of corruption. When errors are random with 0 mean, then AC is equivalent to Active Learning. As we increase the bias, the impact estimates from AC change the sampling distribution and there it is more accurate in comparison.

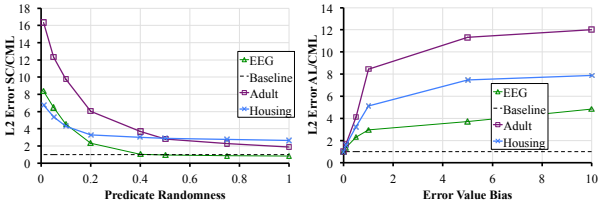

Figure 9: {{The labels are wrong and these plots are confusing}}

7.6 Incremental vs. Sample

{{How much bias before SC becomes useful?}}

7.7 Price of a Scan

{{With partitioning, without partitioning, with(out) indexing?}}

7.8 Real Scenarios

{{Value Filling: MNIST}}
{{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 [28]. 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., [19]). 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 [21].

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 [21]. 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 [17]. 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 [8,24]. 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 [15], 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 [23]. 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 data clean-

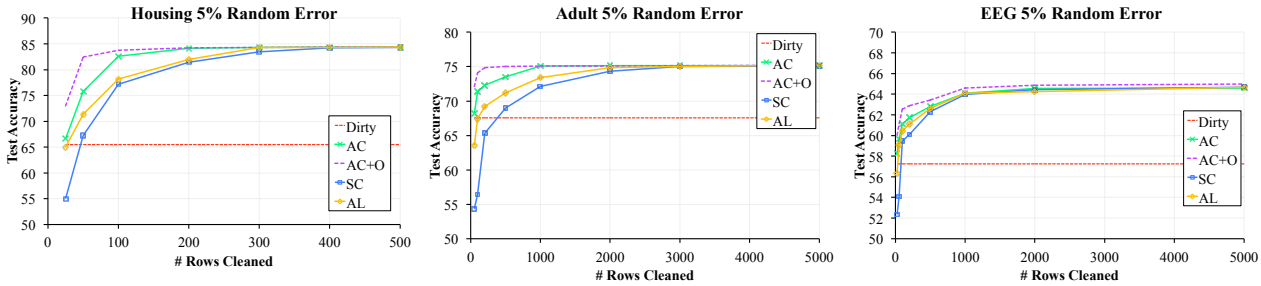

Figure 7: ActiveClean converges with a smaller sample size to the maximum test accuracy in comparison to Active Learning and SampleClean.

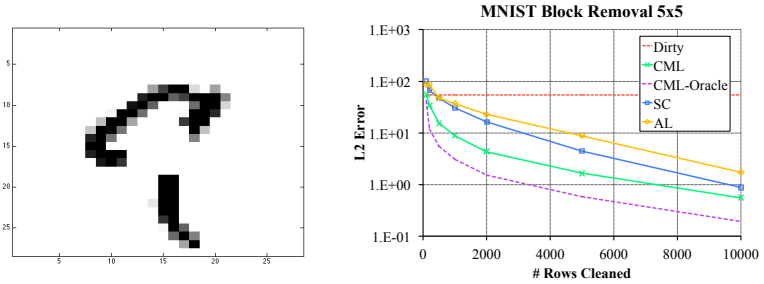

Figure 10: One experiment with MNIST

ing. We implement ActiveClean on Apache Spark to jointly optimize cleaning and modeling pipelines. ActiveClean uses an importance sampling-based stochastic gradient descent framework to prioritize data cleaning. We evaluate ActiveClean on real and synthetic datasets to show that non-uniform sampling achieves improved performance in comparison to uniform sampling, and dirtiness-agnostic prioritization. {{**TODO**}}

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. {{**Sounds confusing rewrite shameless self citation ;)**}} The optimizations described in ActiveClean are not only restricted to stochastic gradient descent algorithms. We believe we can extend a variant of SDCA (Stochastic Dual Coordinate Ascent) [11] 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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

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

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

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

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

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

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

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

[23] 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.

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

[25] 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.

[26] 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?

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

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