Training Models on Dirty Data Under Time Constraints

ABSTRACT

{{TODO}}

1. INTRODUCTION

Large and growing data are often susceptible to various forms of corruption, or *dirtiness*, such as missing, incorrect, or inconsistent values. These corruptions can negatively affect subsequent analysis in subtle but significant ways. Increasingly, analytics consists of Machine Learning "data products", such as recommender systems, spam detectors, and forecasting models, which be very sensitive to data quality. Robust statistical methods have been extensively studied over the last 40 years, and trade off statistical efficiency and/or model bias for reduced sensitivity to the corruptions. While utilizing robust estimates can significantly reduce the generalization error of a statistical model, they essentially learn models that avoid the corruptions. As a result, certain populations of frequently corrupted data may be systematically mispredicted.

A data cleaning approach, complementing existing robust statistical techniques, can mitigate this concern for some types of error. Instead of avoiding the problem, cleaning works by repairing (or approximately repairing) the corruption. Data cleaning is an established line of research and there are numerous recent proposals of systems and techniques [4,8,13,14]. 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, simple data transformations may not be reliable necessitating the use of even costlier crowdsourcing techniques [11,16].

An emerging solution to the growing costs of data cleaning is sampling [3,19,20]. Analysts can sample a large dataset, prototype changes on the sample, and evaluate whether these changes have the desired affect. The case for sampling is analagous to arguments for Approximate Query Processing (AQP) [5], where a timely approximate answer is more desirable than an exact slow answer. Our goal is an *anytime* framework for training Machine Learning models on dirty data. An analyst can clean as much of the data as possible within an allocated time-budget, and then the framework returns a best-effort model. While studied in aggregate query processing (e.g., BlinkDB [5] and Online Aggregation [9,12]), Machine Learning is far more sensitive to sample size (i.e., training data). The key problem is returning a result that is accurate enough for the analyst to judge the significance of their cleaning operations.

There are a few important observations that can allow us to address this problem. First, we might be able to use knowledge about the application (Machine Learning model) to improve efficiency of data cleaning. Increasingly, the consensus in Machine Learning research is that all training data are not created equal and some data are more informative than others [10,18]. This would imply that using the model to guide sampling towards informative data can mitigate the sensitivity to sample size. On the surface, this seems like an Active Learning problem [18], however, most Active Learning approaches only consider label acquisition for unlabeled data. Our second observation is that errors are often happen in batches are often tightly clustered, that is they affect similar data. This motivates an adaptive approach where as an analyst cleans data we learn how data is cleaned to further guide the cleaning.

In this paper, we propose CleanML (CML), an anytime framework for training Machine Learning models with data cleaning. CML supports a class of models called regularized-convex loss problems which includes linear regression, logistic regression, generalized linear models, and support vector machines. Algorithmically, we treat the analysts actions as part of a Stochastic Gradient Descent (SGD). The basic idea of SGD is to draw a data point at random, calculate the gradient at that point, and then update a current best estimate with that gradient. In this work, we start with an initialization (the dirty model) and iteratively update this initialization as we get more clean data. SGD and its variants are well-studied and there are lower-bounds on the convergence rates using these techniques. So we will use SGD as the scaffolding to build and analyze an adpative algorithm. We also apply several techniques from database systems to optimize the model training. For example, if we intelligently partition data we believe is clean and dirty data, we can avoid expensive scans of this data. Next, when we do have to scan the data, we can amortize these scans using shared cursors to do multiple operations at each datum.

In summary, our contributions are

- We propose CleanML which given dirty data, a convex loss model, and a time budget, returns a model trained with respect to the clean data.
- CleanML is implemented with an non-uniform importance sampling approach that priortizes points in the clean model that are most imformative, and we provide analysis to identify the key tradeoffs.
- {{Systems Contributions}}
- We evaluate CleanML on real and synthetic datasets to show that non-uniform sampling achieves improved

performance in comparison to uniform sampling, and dirtiness-agnostic prioritization.

2. BACKGROUND

2.1 Machine Learning and Loss Minimization

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

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

For example, in a linear regression ϕ is:

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

ϕ is often designed to be *convex*, essential meaning bowl-shaped, to make the training this model tractable. Typically, a *regularization* term $r(\theta)$ is added to this problem. The regularization term acts as penalty to avoid overfitting to the training data.

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

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

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

This class of problems includes all generalized linear models and support vector machines.

2.2 Training on Clean Data

The Theoretical Statistics community and Machine Learning community have extensively studied the problem of robustness to statistical outliers and random noise. Various *robust* methods have been proposed to reduce a loss function's sensitivity to outliers. 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:

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

How exactly does a data cleaning approach differ from a robust method? We often in the problem setting where we are training on dirty data, but need to test or deploy the model on clean data. Consider an image classification task with incorrect labeling. If we train a model with respect to the incorrect labels, while we might have achieve a good out-of-sample accuracy on the incorrect labels, the classifications are incorrect in a semantic sense. Likewise, consider the case where we are predicting future product demand based on corrupted historical data. Training a model with respect to the corrupted data might have a low R^2 cross-validation error, but is incorrect at predicting the future trends.

In such scenarios, we see data cleaning as complementary to robust statistics. Data cleaning gives us information about the "true" data distribution, which is highly beneficial when errors have systematic biases. Without cleaning, certain subpopulations of data might be frequently mispredicted.

2.3 SampleClean Project

In our previous work, we studied the relationship between approximate query processing, data cleaning, and sampling [15,20]. Traditionally, data cleaning has explored expensive, up-front cleaning of entire datasets for increased query accuracy. We proposed the SampleClean problem, in which an analyst cleans a small sample of data, and then estimates the result to an aggregate query e.g., `sum`, `count`, or `avg`. The main insight from the SampleClean project is that highly accurate answers for aggregate queries does not require cleaning the full dataset. Generalizing this insight, there is a deep relationship between the application (i.e., the query) and how an analyst should budget their effort in data cleaning. In fact, `avg` and `sum` queries are a special case of the convex loss minimization discussed in the previous section:

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

In SampleClean, we explored a "one-size fits all" approach to sampling and query processing. We take a random sample of data and then answer any aggregate query on this sample. However, in many important Machine Learning applications, there is a single model of interest at the end of the pipeline. In this work, we explore how we can exploit the structure of that model to guide data cleaning towards the most important points. This new setting is very different from the problem previous studied for the following reasons: (1) the new setting considers multidimensional data and has to account for correlations between attributes, (2) while `avg` and `sum` queries can be answered with a single pass of the data, general convex loss optimization requires iteration, and (3) guiding the data cleaning requires an estimate of how valuable the cleaning is to the model.

2.4 Stochastic Gradient Descent

Our goal is to design an anytime framework for training models on dirty data. Our main idea is to introduce data cleaning into the iterative algorithms that train the models. The problems described in the previous subsections are often trained using a technique called Stochastic Gradient Descent (SGD) or one of its variants. The basic idea of SGD is to draw a data point at random, calculate the gradient at that point, and then update a current best estimate with that gradient.

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

SGD can also be applied in a "mini-batch" mode, where we draw a subset of data at random and update with the average gradient.

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

At a high-level, this problem mirrors the problem explored in this work. We start with an initialization (the dirty model) $\theta^{(0)}$ and iteratively update the existing best model as we get more clean data. SGD and its variants are well-studied and there are lower-bounds on the convergence rates using these techniques.

Recently, a number of works have explored non-uniform sampling distributions for stochastic optimization [17,21]. The main insight is that non-uniform distributions may on average estimate the gradient accurately. The technique that is applied is called importance sampling. Importance sampling preserves the exepected value of a parameter while trying to sample from a distribution that results in lower variance.

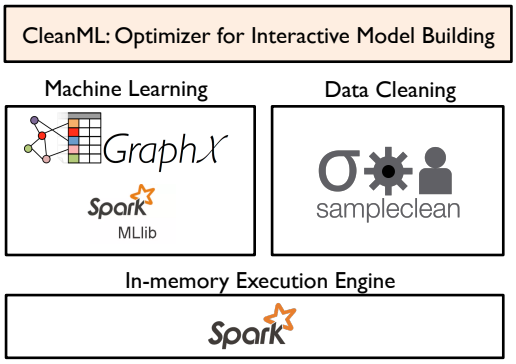

Figure 1: CleanML (CML) is an anytime framework from training models on dirty data. Expensive data transformations prior to model training are budgeted with a non-uniform sampling that priortizes examples that are most likely to change the model. As more data is cleaned the sampling distribution is updated.

3. SYSTEM ARCHITECTURE

3.1 System Architecture

The SampleClean problem inspired an eponymous system that implements sampling, data cleaning, and approximate query processing on the Apache Spark stack [4]. Also included in the Apache Spark stack are Machine Learning libraries including MLlib [2] and GraphX [1]. The in-memory architecture of the Apache Spark stack allows for increasingly interactive analysis [6,7]. Analysts can prototype data processing workflows on samples to evaluate performance before running expensive batch processing jobs on entire datasets. With data cleaning and machine learning libraries in the same software ecosystem, we see a new opportunity for joint optimization for interactive model building.

We believe that interactive model building problem is best addressed by an anytime approach, where an analyst has a time budget and wants a best effort result given this time budget. CML optimizes the convergence of the model by adaptively changing the sampling distribution. In Figure 1, we illustrate our system archictecture. CML is an optimizer that wraps around the SampleClean data cleaning library and Machine Learning libraries in Spark.

3.2 Example Use Case And API

Consider the following example problem. We are given the University of Texas Restaurant [1] dataset, which has the following schema:

```
Restaurant(name, address, city,, type)
```

We want to train a Support Vector Machine classifier that given the tokens in the name and the city, can predict the category of the restaurant (e.g., Chinese or Mexican). However, the challenge is that this dataset has numerous entity resolution problems where city names and categories are inconsistently represented.

```
rex il ristorante,617 s. olive st.,los angeles,
    italian
rex il ristorante,617 s. olive st.,los angeles,
    nuova cucina italian
21 club,21 w. 52nd st.,new york,american
21 club,21 w. 52nd st.,new york city,american(new)
```

Entity resolution can be relatively expensive if the inconsistent representations are very different from each other and are not amenable to similarity measure optimizations such as prefix filtering or sorting.

{{Maybe Describe API Here}}

The analyst wants to quickly understand how much better a clean data set would be for this task. Using our system, the analyst can construct a data cleaning workflow and specify a budget. For example, this is what the code would look like in CML.

```
restaurant.load()
.clean(EntityResolution.weightedJaccard('type'
    ,0.6))
.clean(EntityResolution.editDistance('city',10))
.featureView(List('type','label'),('city','one_hot
    '),('name','bag_of_words')))
.model(new SVMModel(_))
.budget(5000)
```

3.3 Algorithmic Problem Statement

To formalize the optimization problem:

Base Data and Featurization: We are given a base dataset $\mathcal{R}$ which is a relation with N rows. 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:

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

.

Data Cleaning: We define data cleaning as transformations that happens to the base data that result in changes to the feature vector. We do not consider data cleaning operations that change the number of rows in the relation such as row deletion or row deduplication. Allowed operations must result in the following changes to the feature vector: (1) feature transformation, (2) feature deletion, and (3) feature addition. Therefore, there is at most a one-to-one relationship beftween features in the cleaned data and the dirty data.

{{Formalize Better}}

Budget Specification: The analyst specifies two hyperparameters a cleaning budget B which is the number of tuples to clean, and the number of cleaning rounds T. In each round, $\frac{B}{T}$ rows are processed by the system, we will discuss the tradeoffs between batching and incremental processing in the subsequent sections.

{{Maybe exclude rounds}}

Model: The analyst specifies the model which we require to be solved via convex regularized loss minimization:

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

Output: The output of CML is at each cleaning round we draw the sample of $\frac{B}{T}$ rows from a sampling distribution that incorporates information about the value of cleaning that row to model. After the T cleaning rounds, the user is returned an approximately clean model $\theta^{(c)}$.

4. IMPORTANCE SAMPLING AND SGD

[1] http://www.cs.utexas.edu/users/ml/riddle/data/restaurant.tar.gz

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

4.1 Basic Algorithm: Oracular Case

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

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

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

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

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

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

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

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

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

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

It is established [21] that the optimal sampling distribution (lowest variance) is:

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

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

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

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

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

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

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

We have been purposefully vague about how to learn the estimator $\hat{C}$ to simplify the intuition and analysis of the algorithm. In the subsequent subsections, we will describe different settings and how we can efficiently learn $\hat{C}$. Essentially, all variants of the basic algorithm will modify step 2 in differing ways.

4.1.1 Oracular Case: Analysis

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

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

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

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

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

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

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

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

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

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

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

4.2 Error Predicate Case

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

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

Essentialy, we are computing the gradient at many points and then throwing that computation away when we sample.

We now explore how we can use this previous computation to further reduce the variance σ^2.

In Section 3.2, we described the basic API and user interface of our system. An analyst specifies multiple cleaning operations and then a model. Suppose, we decomposed the function C into its consituent cleaning operations C_i^k. To describe the parse errors, for each C_i there is some predicate ρ_i that selects the affected records. The next case that we analyze is when we know the perfectly ρ_i in advance. Unlike the oracular case, there are some real-world situations where this true. For example, if data is missing e.g., field misalignment it is often clear which rows are affected and which are not. Similarly, in the entire field of constraint-based data cleaning [**?**], we know in advance which records have constraint violations.

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

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

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

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

the resulting estimator is:

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

The oracular algorithm gets modified as follows:

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

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

4.2.1 Error Predicate Case: Analysis

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

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

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

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

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

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

4.3 General Case

Finally, in the general case, we do not know the error predicate in advance. Consider a crowd-based cleaning technique whose effects are hard to predict a priori. In this case, we have to continuously update a classifier to partition the data into clean and dirty. The challenge is that this potentially biases our result if our classifier is not perfect. Some dirty points may be classified as clean. The quadratic savings seen in the predicate case can often outweigh small biases due to classification error.

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

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

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

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

5. REFERENCES

[1] Graphx. `https://spark.apache.org/graphx/`.
[2] Mllib. `https://spark.apache.org/mllib/`.
[3] Trifacta. `http://www.trifacta.com`.
[4] Sampleclean. http://sampleclean.org/, 2015.
[5] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.
[6] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

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

[8] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. 2015.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In *SIGMOD Conference*, pages 1115–1118, 2010.

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

[11] 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] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD Conference*, pages 171–182, 1997.

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

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

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

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

[19] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[20] 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.

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