ActiveClean: Interactive Data Cleaning For Modern Machine Learning

Sanjay Krishnan, Jiannan Wang, Eugene Wu [†], Michael J. Franklin, Ken Goldberg
UC Berkeley, [†]Columbia University
{sanjaykrishnan, jnwang, franklin, goldberg}@berkeley.edu
ewu@cs.columbia.edu

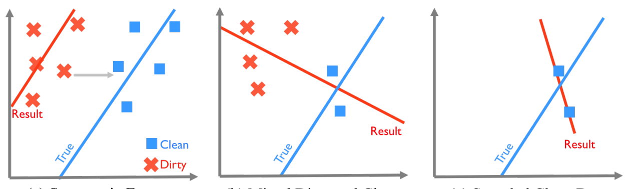
ABSTRACT

Data cleaning is often an important step to ensure that predictive models, such as regression and classification, are not affected by errors such as inconsistent, out-of-date, or outlier data. Identifying dirty data is often a manual and iterative process, and can be challenging on large datasets. However, many data cleaning workflows can introduce subtle biases into the training processes due to violation of independence assumptions. We propose ActiveClean, a progressive cleaning approach where the model is updated incrementally instead of re-training and can guarantee accuracy on partially cleaned data. ActiveClean supports a popular class of models called convex loss models (e.g., linear regression and SVMs) and record-by-record user-defined data cleaning operations. ActiveClean also leverages the structure of a user's model to prioritize cleaning those records likely to affect the results. Evaluation on four real-world datasets suggests that for a fixed cleaning budget, ActiveClean returns more accurate models than uniform sampling and Active Learning when corruption is systematic and sparse.

1. INTRODUCTION

Building distributed frameworks to facilitate model training on large and growing datasets is a key data management challenge with significant interest in both industry and academia [1, 2, 4, 5]. While these frameworks abstract much of the difficult details of distributed Machine Learning (ML), they seldom offer the analyst any support in terms of constructing the model itself, such as which features to use or how to represent their data. The model construction process is still highly iterative, where through trial-and-error an analyst makes these choices eventually converging onto a model with the desired accuracy. To further complicate matters, data often arrives *dirty*, including missing, incorrect, or inconsistent attributes, due to faulty sensors, software, time delays, or hardware. Thus, part of the iterative model construction process involves identifying potentially dirty data, understanding how they affect the model, and applying techniques to mitigate their effects. While data cleaning is an extensively studied problem, the high dimensionality of many models can amplify even a small amount of erroneous records [7], and the relative complexity (in comparison to SQL analytics) can make it difficult to trace the consequnces of an error.

In prior work, we have noted the choice of data cleaning algorithm can significantly affect results even when using robust ML techniques [3, 6]. In one fraud prediction example, we found that simply applying Entity Resolution before model training improved true positive detection probabilities from 62% to 91%. Despite this importance, in theory and in practice, the academic community has decoupled the data cleaning problem from featurization and ML. This is problematic because many ML techniques often make assumptions about data homogeneity and the consistency of sampling, which can be easily violated if the analyst applies data cleaning in an arbitrary way.



(a) Systematic Error (b) Mixed Dirty and Clean (c) Sampled Clean Data

Figure 1: (a) Systematic corruption in one variable can lead to a shifted model. (b) Mixed dirty and clean data results in a less accurate model than no cleaning. (c) Small samples of only clean data can result in similarly inaccurate models.

To understand how this may happen, consider an anlyst training a regression model on dirty data. At first, she may not realize that there are outliers and train an initial model directly on the dirty data. As she starts to inspect the model, she may realize that some records have a large residual value (not predicted accurately). Once she confirms that those records are indeed dirty, she has to design data cleaning rules or scripts to fix or remove the offending records. After cleaning, she re-trains the model–iterating until she no longer finds dirty data. This iterative process is the de facto standard, and in fact encouraged by the design of the increasingly popular interactive "notebook" ML development environments (e.g., IPython), but makes the implicit assumption that model training commutes with incremental data cleaning. This assumption is wholly incorrect; due to the well-known Simpson's paradox, models trained on a mix of dirty and clean data can have very misleading results even in simple scenarios (Figure 1).

In a parallel trend, the dimensionality of the features used in ML models is also rapidly increasing. It is now common to use 100,000s of features in image processing processing problems with techniques such as Deep Learning. Empirically, such feature spaces have facilitated breakthroughs in previously hard classification tasks such as image classification, robot actuation, and speech recognition. However, the pitfall is that the standard approaches for debugging and reasoning about data error may lose their intuition for higher dimensions. In other words, it is often not obvious how an analyst should select which records to clean.

As it stands, there are two key problems in interactive model construction, (1) correctness, and (2) dirty data identification. We address these to problems in a system called ActiveClean which facilitates interactive training-cleaning iteration in a safe way (with expected monotone convergence guarantees) and automatically selects the most valuable data for the analyst to inspect even in the complex models popular in modern ML pipelines. The selection technique applied in ActiveClean uses pointwise gradients to generalize the outlier filtering heuristics to select potentially dirty data even in complex models. The analyst initializes an ActiveClean with an ML model, a featurization function, and the base data, and the ActiveClean initially returns the model trained on the dataset. ActiveClean also returns an array of data sampled from the model that are possibly dirty. The analyst can apply any value transformations to the data and then prompt the system to iterate.

We demonstrate ActiveClean with a visual interface allows analysts to debug complex ML models and understand the effects of dirty data. This interface will allow the analyst to specify the desired model and featurization. It will then visualize a sample of potentially dirty records and allow the analyst to clean the data appropriately. In our demonstration, we will present three experimental scenarios where the models are affected by dirty data:

EXAMPLE 1 (EVENT DETECTION WITH SVMS).

EXAMPLE 2 (VIDEO SEGMENTATION WITH CNNS).

EXAMPLE 3 (TOPIC MODELING WITH LDA).

2. ARCHITECTURE AND OVERVIEW

We will first describe ActiveClean and overview the entire framework, and a detailed description of the research challenges and algorithms can be found in [3].

2.1 Problem Setup and Formalization

There is a relation R and we wish to train a model using the data in R. We assume that there is a featurizer $F(\cdot)$ that maps every record $r \in R$ to a feature vector x and label y. This work focuses on a class of well-analyzed predictive analytics problems; ones that can be expressed as the minimization of loss functions. For labeled training examples $\{(x_i, y_i)\}_{i=1}^{N}$, the problem is to find a vector of *model parameters* θ by minimizing a loss function ϕ over all training examples:

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

Where ϕ is a convex function in θ. For example, in a linear regression ϕ is:

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

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

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

In this work, without loss of generality, we will include the regularization as part of the loss function i.e., $\phi(x_i, y_i, \theta)$ includes $r(\theta)$.

2.2 Required User Input

More concretely, the analyst provides the following user-defined function to use ActiveClean:

Model: The user provides a predictive model (e.g., SVM) specified as a loss optimization problem $\phi(\cdot)$ and a featurizer $F(\cdot)$ that maps a record to its feature vector x and label y.

Gradient: The user provides a gradient function $\nabla\phi(\cdot)$ that returns the gradient of the loss. For popular convex models such as SVMs and Linear Regression these functions are known and easy to express analytically. For more complex problems such Topic Modeling or Neural Network learning, we assume that this function (or an approximation of it) is expressed programatically. There are a number of frameworks such as Torch, Theano, and TensorFlow, which can return such programs using symbolic differentiation.

Batches: Data are cleaned in batches of size b and the user can change these settings if she desires more or less frequent model updates. We empirically find that a batch size of 50 performs well across different datasets and use that as a default. A cleaning budget k can be used as a stopping criterion once $C(\dot{})$ has been called k times, and so the number of iterations of ActiveClean is $T = \frac{k}{b}$. Alternatively, the user can clean data until the model is of sufficient accuracy to make a decision.

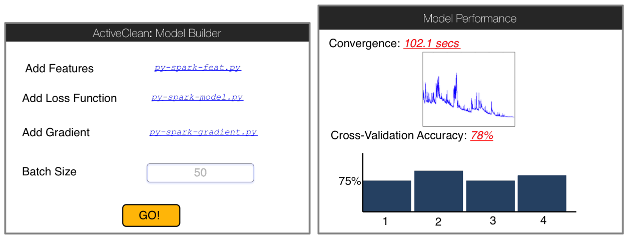
Cleaning Function: We represent this operation as $Clean(\cdot)$ which can be applied to a record r (or a set of records) to recover the clean record $r' = Clean(r)$. Formally, we treat the $Clean(\cdot)$ as an expensive user-defined function (implemented by code or by manual inspection) composed of deterministic schema-preserving map and filter operations applied to a subset of rows in the relation.

2.3 Basic Data Flow

The system first trains the model $\phi(\cdot)$ on the dirty dataset to find an initial model $\theta^{(d)}$ that the system will subsequently improve. The *sampler* selects a sample of size b records from the dataset and passes the sample to the *cleaner*, which executes $Clean(\cdot)$ for each sample record and outputs their cleaned versions. The *updater* uses the cleaned sample to update the weights of the model, thus moving the model closer to the true cleaned model (in expectation). Finally, the system either terminates due to a stopping condition (e.g., $C(\cdot)$ has been called a maximum number of times k, or training error convergence), or passes control to the *sampler* for the next iteration.

To summarize in pseudocode:

```
1. Init(dirty_data, cleaned_data, dirty_model, batch, iter)
2. For each t in $\{1, ..., T\}$
   (a) dirty_sample = sampler(dirty_data, sample_prob, detector, batch)
   (b) clean_sample = Cleaner(dirty_sample)
   (c) current_model = Update(current_model, sample_prob, clean_sample)
   (d) cleaned_data = cleaned_data + clean_sample
   (e) dirty_data = dirty_data - clean_sample
   (f) sample_prob = Estimator(dirty_data, cleaned_data, detector)
```


ActiveClean: Model Builder

Add Features py-spark-feat.py

Add Loss Function py-spark-model.py

Add Gradient py-spark-gradient.py

Batch Size 50

Model Performance

Convergence: *102.1 secs*

Cross-Validation Accuracy: *78%*

Figure 2: Initial Run. The analyst loads user-defined model functions into ActiveClean and then trains an initial model on the dirty data

```
    (g) detector = DetectorUpdater(detector, cleaned_data)

3. Output: current_model
```

2.4 Optimizations

We overview some of the research contributions and the details can be found in [3].

Gradient-Based Updates: Rather than retraining, in ActiveClean, we start with a dirty model as an initialization, and then incrementally make an update using a gradient step.We can draw an analogy to Materialized View maintenance, since after all, a model parametrized by θ is just a table of floating point numbers. This process leverages the structure of the model rather than treating it like a black-box, and we apply convergence arguments from optimization theory.

Estimate-Driven Prioritization: We use an importance sampling technique to select a sample of likely dirty records. This is designed in a way so it still preserves convergence guarantees.

Partitioning Dirty and Clean Data: ActiveClean adaptively learns to partition dirty and clean data based on the analysts actions. Partitioning serves two purposes: (1) it reduces the variance of our updates because we can cheaply scan over data we know that is clean, and (2) it increases the fraction of actually dirty records in the candidate batch.

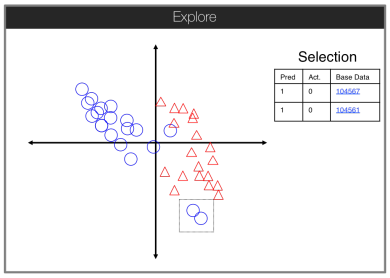
3. THE INTERFACE

Next, we will describe the components of the ActiveClean interface used in this demonstration.

3.1 The Intial Run

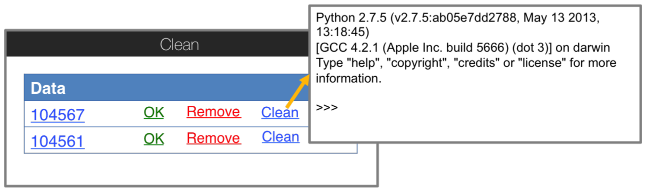
The first part of ActiveClean is called the Model Builder, this is an interface that the analyst uses to specify the problem. She loads three user-defined functions written in PySpark [?] based on the descriptions in the previous section. Optionally, she can change the batch size from our default setting of 50. Once the model is instantiated, then she can train the model on the dirty data. If there are any errors that are critical, i.e., causes the training to fail, ActiveClean will immediately error. However, if the training proceeds to completion, the analyst will see the Performance window which plots the model's convergence as a function of iteration. It also shows the cross-validation accuracy if it is a classification task and the hold-out residual error if it is a regression task. Both of these panels are visualized in Figure 2.

3.2 Diagnose Interface



Pred	Act.	Base Data
1	0	104567
1	0	104561

Figure 3: The diagnose inteface. The analyst can select and inspect suspect data.


Clean

Data			
104567	OK	Remove	Clean
104561	OK	Remove	Clean

```
Python 2.7.5 (v2.7.5:ab05e7dd2788, May 13 2013,
13:18:45)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.

>>>
```

Figure 4: Clean Eval

Suppose the analyst is unhappy with her model, and wishes to understand why her prediction accuracy is poor. She can then open the diagnose panel to understand why (Figure 3). When she opens the diagnose panel, ActiveClean applies the importance sampling algorithm to select a subset of examples from the dataset. Since these points are in general high dimensional, we apply T-SNE [?] to visualize the points in 2D. T-SNE is a non-linear dimensionality reduction technique which is widely used to visualize complex data distributions. In the diagnose interface, we use color coding to indicate class in the case of classification. The analyst can select examples from the diagnose interface for further inspection.

3.3 Clean Interface

3.4 Iteration and Updates

4. REFERENCES

[1] Berkeley data analytics stack. `https://amplab.cs.berkeley.edu/software/`.

[2] A. Alexandrov, R. Bergmann, S. Ewen, J. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. *VLDB J.*, 23(6), 2014.

[3] A. Arxiv. Activeclean: Arxiv. `http://arxiv.org`.

[4] A. Crotty, A. Galakatos, and T. Kraska. Tupleware: Distributed machine learning on small clusters. *IEEE Data Eng. Bull.*, 37(3), 2014.

[5] G. Inc. Tensorflow. `https://www.tensorflow.org/`.

[6] J. Mahler, S. Krishnan, M. Laskey, S. Sen, A. Murali, B. Kehoe, S. Patil, J. Wang, M. Franklin, P. Abbeel, and K. Y. Goldberg. Learning accurate kinematic control of cable-driven surgical robots using data cleaning and gaussian process regression. In *CASE*, 2014.

[7] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *ICML*, 2015.

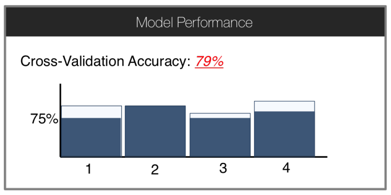


Figure 5: Clean Eval