ActiveClean: An Interactive Data Cleaning Framework For Machine Learning

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

ABSTRACT

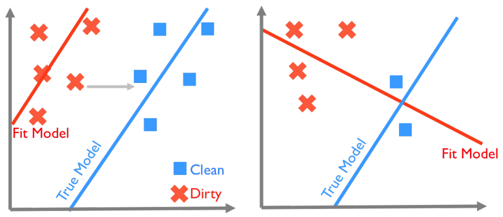
Databases can be corrupted with various errors such as missing, incorrect, or inconsistent values. Increasingly, modern data analysis pipelines involve Machine Learning, and the effects of dirty data can be difficult to debug. Dirty data is often sparse, and naive sampling solutions are not suited for high-dimensional models. We propose ActiveClean, an anytime framework for training Machine Learning models with data cleaning. Our framework updates a model iteratively as the analyst cleans small batches of data, and includes numerous optimizations such as importance weighting and dirty data detection. We designed a visual interface to wrap around this framework and demonstrate ActiveClean for a video classification problem and a topic modeling problem.

1. INTRODUCTION

Model training on large and growing datasets is a key data management challenge with significant interest in both industry and academia [1, 5, 7, 9]. While many scalable training frameworks abstract much of the difficult details of distributed Machine Learning (ML), they seldom offer the analyst 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 often through trial-and-error. To further complicate matters, data are often *dirty*, including missing, incorrect, or inconsistent attributes, due to faulty sensors, software, time delays, or hardware. Thus, although part of the iterative process is tweaking the model parameters and features, a significant portion involves identifying and cleaning potentially dirty data. In our work, we focus on this latter issue. While data cleaning is an extensively studied problem, the high dimensionality of many models can amplify even a small amount of erroneous records [15], and the relative complexity (in comparison to SQL analytics) can make it difficult to trace the consequences of an error.

To highlight the importance of data cleaning in modern ML pipelines, we have noted the choice of data cleaning algorithm can significantly affect results even when using robust statistical techniques [10, 11]. For instance, 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 independent and identically distributed data, which can be easily violated if the analyst applies data cleaning in an arbitrary way.

To understand how this may happen, consider an analyst training a regression model on dirty data. At first, she may not realize that there are outliers and train a preliminary model directly on the dirty data. As she starts to inspect the model and the data, she will soon realize that some records have a large residual error (i.e., not predicted accurately). Once she confirms that those records are indeed dirty, she has to design 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 is in fact encouraged by the design of the increasingly popular interactive "notebook" ML development environments (e.g., IPython). However, this makes the implicit assumption that model training commutes with incremental data cleaning, and the models trained on partially clean data have meaningful predictive value. This assumption is often 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).



(a) Systematic Error (b) Mixed Dirty and Clean

Figure 1: (a) Systematic corruption in one variable (axes) can lead to a shifted model (fitted lines). (b) Mixed dirty and clean data results in a less accurate model than no cleaning.

In a parallel trend, the advent of techniques such as Deep Learning and Non-Parametric Bayesian Methods has lead to an explosion in the number of model parameters. It is now common to use 100,000s of features in image processing processing problems with Convolutional Neural Networks. 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 higher dimensional models are harder to debug. Determining the effects and interaction between dirty data, model error, and counter-intuitive higher dimensional effects can be very challenging.

As it stands, there are two key problems in ML over dirty data, (1) correctness, and (2) dirty data identification. We address these two 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. ActiveClean is implements the algorithms proposed in our prior work [10]. 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 set 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.

Intuitively, ActiveClean prioritizes data cleaning by identifying records, which if cleaned, are likely to change the analyst's model predictions. ActiveClean applies to a large class of models which can be represented as loss minimization problems solved by gradient descent. This captures SVMs, Linear Regression, Neural Networks (Deep Learning), and some types of topic modeling problems such as LDA (for a formal description see [10]). In our demonstration, we will show how ActiveClean facilitates debugging complex ML models to understand the effects of dirty data in two experimental scenarios:

EXAMPLE 1 (VIDEO SEGMENTATION WITH CNNS). *The JHU JIGSAWS Dataset is a corpus of surgical training 120 videos from between 1-5mins long. These videos are annotated by expert surgeons describing the gestures that occur in the video. Classifying video frames is an important task for segmentation and summarization of future videos. We would like a classifier that can predict these annotations. This can be done using Convolutional Neural Networks to featurize frames of the video and then applying a standard classifier like an SVM after the features are extracted. However, sometimes the annotations are incorrect and ActiveClean can be used to determine when the annotations are incorrect (correspond to the wrong gesture) and inconsistent (multiple gestures simultaneously).*

EXAMPLE 2 (TOPIC MODELING WITH LDA). *We have a corpus of reviews from Yelp and we wish to learn a topic model from this dataset. However, a substantial number of the reviews are spam that typically are soliciting traffic for a fraudulent website. These spam reviews can affect the distribution of words and affect any models learned from the data. However, some spam reviews are hard to detect automatically and require human validation. We can apply ActiveClean to efficiently estimate the topic model without having to validate every review.*

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 [10].

2.1 What Is New?

Machine learning, specifically active learning, has been applied in prior work to improve the efficiency of data cleaning [16, 17, 8]. Human input, either for cleaning or validation of automated cleaning, is often expensive and impractical for large datasets. An ML model can be used to extrapolate repairs to not-yet-cleaned data, and the goal of these approaches is to provide the cleanest possible dataset–independent of the subsequent analytics or query processing. Prior work studies how to *use machine learning models* to improve data cleaning. In contrast, ActiveClean explores how to control the impact of data cleaning for downstream machine learning models.

This new problem setting leads a question of correctness – if I incrementally clean subsets of my data, is the model I then train even correct? It also leads to optimization opportunities – how can the data cleaning step be made aware of the subsequent data analysis (e.g., the model that is trained) in such a way that the model will most quickly converge to the correct model? Existing approaches, which are designed for homogeneous (all clean or all dirty) data, cannot be applied here. One of the primary contributions of this work is an incremental model update algorithm with correctness guarantees for the resulting mixture of dirty and clean data. The essence of the approach is to model the human's iterative process as a Stochastic Gradient Descent loop.

2.2 Problem Setup and Formalization

We assume that there is a featurizer $F(\cdot)$ that maps every record in a dataset $r \in R$ 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, that will be trained on the output of applying $F(\cdot)$ to the records in R. For these 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 ℓ over all training examples:

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

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

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

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.

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

In this work, without loss of generality, we will include the regularization as part of the loss function i.e., $\ell(x_i, y_i; \theta)$ includes $r(\theta)$. We note that designing ActiveClean for this form of machine learning model supports SVMs, Linear Regression, Logistic Regression, Neural Networks, Latent Dirichlet Allocation, and Gaussian Mixture Models.

2.3 Required User Input

In such a problem setting, ActiveClean takes as input four user-defined parameters. The first two are UDFs used in order to train models, and can be assumed to be readily available. The latter two consist of a user tunable parameters with reasonable defaults, and a generic record cleaning function implemented by script or by manual effort:

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

Gradient: The gradient function $\nabla\ell(\cdot)$ returns the gradient of the loss. For popular convex models such as SVMs and Linear Regression these functions are known and provided as part of the system. 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.

Stopping Criteria: 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 $C(\cdot)$ which can be applied to a record r (or a set of records) to recover the clean record $r' = C(r)$. Formally, we treat the $C(\cdot)$ as an expensive user-defined function (implemented as code or manual inspection) composed of deterministic schema-preserving map and filter operations applied to a subset of rows in the relation. This formulation supports common operations such as extraction, deletion, transformation, and find-and-replace.

2.4 Basic Data Flow

The following pseudocode summarizes how ActiveClean works:

```
1. Init(dirty_data, cleaned_data, dirty_model, batch,
   iter)
2. For each t in $\{1, \ldots, T\}$
   (a) dirty_sample = Sampler(dirty_data, sample_prob,
       detector, batch)
   (b) clean_sample = Cleaner(dirty_sample)
   (c) current_model = Updater(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)
   (g) detector = Detector(detector, cleaned_data)
3. Output: current_model
```

The system first trains the model $\ell(\cdot)$ 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 $C(\cdot)$ on the whole sample 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. ActiveClean assumes that *Init* and *Cleaner* are user-specified, and it implements all of the other functions.

2.5 Technical Details

We overview some of the research contributions and the details can be found in [10].

Updater (Incremental Gradient Method): Rather than retraining, in ActiveClean, we start with a dirty model as an initialization θ_d, and then incrementally make an update using a gradient step.This process leverages the structure of the model rather than treating it like a black-box, and we apply convergence arguments from optimization theory.

Suppose, we model the cleaner $C(\cdot)$ as an oracle that maps a dirty example (x_i, y_i) to a clean example (x'_i, y'_i). Ideally, we would like to be able to solve the following problem:

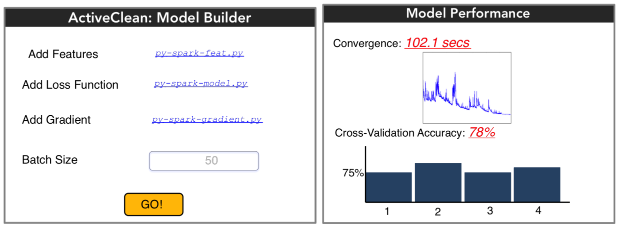
$$\arg\min_{\theta\in\Theta} \sum_{i=1}^{N} \ell(C(x_i, y_i); \theta)$$

This minimization problem can be solved with an algorithm called Stochastic Gradient Descent, which iteratively samples data, estimates a gradient, and updates the current best model. We can represent the iterative cleaning process similarly where the gradient is estimated from newly cleaned batches of data:

$$\theta^{(0)} \leftarrow \theta^*_d$$

$$\theta^{(t+1)} = \theta^{(t)} - \lambda \cdot \mathbf{w}$$

where $\mathbf{w}$ is an unbiased estimate of the gradient with $\mathbf{E}[\mathbf{w}] = \sum_i^N \nabla\ell(C(x_i, y_i); \theta^{(t)})$. If iterated to infinity, this procedure will converge to the correct value θ^*_c with rate $O(\frac{1}{t})$ for strongly convex functions and $O(\frac{1}{\sqrt{t}})$ for general convex functions and avoids the pitfalls of Simpson's paradox. There are a number of subtleties which we discuss in the technical report such as re-sampling already cleaned data and selecting batch sizes.


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

Sampler (Importance Sampling): We use an importance sampling technique to select a sample of likely dirty records. The necessary conditions for convergence are $\mathbf{E}[\mathbf{w}] = \sum_i^N \nabla\ell(C(x_i); \theta^{(t)})$ and there is nothing that enforces that we must uniformly sample the data. We can apply a technique called importance sampling which allows us to calculate expectations with respect to one distribution $q(\cdot)$ while sampling from another with the same support $p(\cdot)$.

$$\mathbf{E}_{X\ q}[X] = \sum_{x\in X} x\frac{q(x)}{p(x)}$$

What this means for our problem is that we can define any sampling distribution $p(\cdot)$ over X as long as $p(\cdot) > 0$:

$$\mathbf{w} = \frac{|X|}{b}\sum_{i=1}^{b}\frac{1}{p(i)}\nabla\ell(C(x_i, y_i); \theta^{(t)})$$

While all of the distributions will have the same expected values, their variances will differ. We can show that an approximation for the distribution that minimizes the variance is simply:

$$p(i) \propto \|\nabla\ell(x_i, y_i; \theta^{(t)})\|$$

3. THE INTERFACE

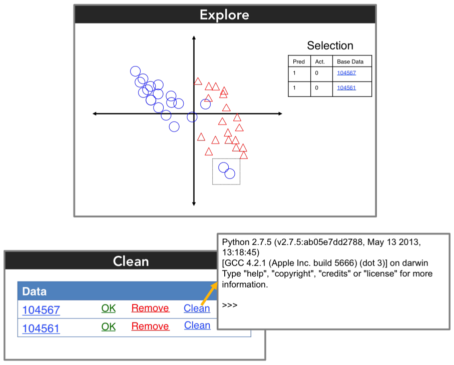
Next, we will describe the components of the ActiveClean interface used in this demonstration.

3.1 Initialization

The analyst first uses the Model Builder (Figure 2, left panel) to specify and initialized the problem. She loads three user-defined functions written in PySpark [12] based on the descriptions in the previous section, and optionally tune the stopping criteria. Once the model is trained, the right panel will show summary information. The top shows Performance information, and plots the model's convergence as a function of iteration, The bottom shows model accuracy information, and shows the cross-validation accuracy if it is a classification task and the hold-out residual error if it is a regression task.

3.2 Diagnose Interface

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 and visualize a subset of examples from the dataset. Since these points are in general high dimensional, we apply T-SNE [13], a non-linear dimensionality reduction technique that is widely used to visualize complex data distributions, to visualize the points in a 2-D plot. The points are color coded to indicate



Pred	Act.	Base Data
1	0	104567
1	0	104561

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 3: Diagnose Interface (above): The analyst can select and inspect suspect data. Cleaning Interface (below): The analyst can choose to remove data, write a custom cleaning operation, or automatically clean the data using an existing cleaning operation.

the label in the case of classification. The analyst can select examples from the plot for further inspection.

3.3 Cleaning Interface

If the analyst decides that the data are actually dirty, then she can open the Clean panel. This panel gives her the option to remove the dirty record, apply a custom cleaning operation (specified in Python), or pick from a pre-defined list of cleaning functions. Custom cleaning operations are added to the library to help taxonomize different types of errors and reduce analyst cleaning effort. This also provides a hint about which data are similarly corrupted (and thus fixed), we can guide future samples to draw similar data in future samples. We do this by training a classifier to learn the conditions for when the operation is applicable to a record, the details are described in our technical report [10].

Finally, after the data sample is cleaned, ActiveClean updates the current best model, and re-runs the cross-validation to visualize changes in the model accuracy. At this point, ActiveClean begins a new iteration by drawing a new sampling of records to show the analyst. Once ActiveClean satisfies the stopping conditions, the current best model parameters along with the (partially) cleaned dataset are returned to the analyst.

4. DEMONSTRATION PROPOSAL

In our demonstration, we will highlight both ActiveClean's overall efficacy on real large-scale scientific datasets, as well as provide the user with hands-on experience working in the clean-validate-retrain loop that is part of ActiveClean. Overall, the demon is intended to show how ActiveClean quickly improves the accuracy of the examples presented in the introduction.

To this end, we will first run a head-to-head comparison between ActiveClean and naive Active Learning in automatic mode – the participant will pick the example setting and see an animation as both algorithms proceed in real-time. The demo will automatically walk through all of the key panels of the interface, using full-scale instantiations of the models and datasets. As part of this process, both algorithms will select samples of dirty data points, automatically clean the points by replacing incorrect values with ground-truth, and retrain the model. The visualization of the cross-validation accuracy, as well as the diagnosis interface, will be updated in real time, so that participants can visually compare the progress of both ActiveClean and the Active Learning algorithms in a real scenario.

After the automatic mode is complete, participants will have hands-on experience with ActiveClean. We will present a simplified dataset and model with artificial errors that is small enough to fully clean in a minute or two. Participants will be able to train an SVM and manually clean samples of data using ActiveClean for a binary classification task. This will illustrate the tradeoffs and usability of the system.

5. CONCLUSION

Building and training high quality machine learning models is hard, and doing so in the context of dirty data is painful. When presented with large, dirty datasets, practitioners often complaint that it is difficult to even know where to start the cleaning process. Worse, this process is often manual and makes fully cleaning the entire dataset impractical. Our key insight is that an important and broad class of predictive models, called loss models, *can* be cleaned progressively *with guarantees* by embedding the process into an incremental optimization loop controlled by a system such as ActiveClean. We hope to convey to the participants that the design of the ML development environment can be used to facilitate proper methodology.

This research is supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Adatao, Adobe, Apple, Inc., Blue Goji, Bosch, C3Energy, Cisco, Cray, Cloudera, EMC2, Ericsson, Facebook, Guavus, HP, Huawei, Informatica, Intel, Microsoft, NetApp, Pivotal, Samsung, Schlumberger, Splunk, Virdata and VMware.

6. REFERENCES

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

[2] Informatica. `https://www.informatica.com`.

[3] Talend. `https://www.talend.com/solutions/etl-analytics`.

[4] Trifacta. `http://www.trifacta.com`.

[5] 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.

[6] Z. Chen and M. Cafarella. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.

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

[8] 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.

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

[10] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning while learning convex loss models. `http://sampleclean.org/activeclean.pdf`.

[11] 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.

[12] P. Spark. Pyspark. `https://spark.apache.org/docs/0.9.0/python-programming-guide.html`.

[13] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

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

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

[16] M. Yakout, L. Berti-Equille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD Conference*, 2013.

[17] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5), 2011.