ActiveClean: Interactive Data Cleaning For Modern Machine Learning

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

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 which can be sensitive to dirty data. Dirty data is often expensive to repair, and naive sampling solutions are not suited for training high dimensional models. We propose ActiveClean, an anytime framework for training Machine Learning models with budgeted data cleaning. Our framework updates a model iteratively as small samples of data are cleaned, and includes numerous optimizations such as importance weighting and dirty data detection. In this demonstration, 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, 2, 3, 5]. While these 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, where through trial-and-error, an analyst makes these choices. Eventually she converges 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, although part of the iterative process is tweaking the model paratemers 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 [11], and the relative complexity (in comparison to SQL analytics) can make it difficult to trace the consequnces of an error.

To highlight the importance of data cleaning in modern machine learning, we have noted the choice of data cleaning algorithm can significantly affect results even when using robust statistical techniques [6, 7]. 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 data homogeneity and the consistency of sampling, which can be easily violated if the analyst applies data cleaning in an arbitrary way.

To understand how this may happen, consider an anlyst training

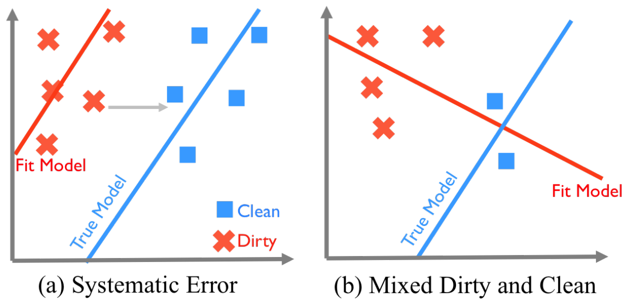


Figure 1: (a) Systematic corruption in one variable (axes) can lead to a shifted model (fitted lines). (b) Mixed dirty and clean data results in a less accurate model than no cleaning.

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 and the data, she will soon realize that some records have a large residual error (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).

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 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 higher dimensional models are harder to debug and reason about. Determining the differences between dirty data, model error, and counter-intuitive higher dimensional effects can be very challening. For an analyst, it is often not obvious how to identify a set of likely dirty records.

As it stands, there are two key problems in model construction, (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; 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 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 Netorks (Deep Learning), and some types of topic modeling problems such as LDA (for a formal description see [6]). In our demonstration, we will show how ActiveClean facilitates debugging complex ML models to understand the effects of dirty data. The demonstration will consist of a visual interactive interface that will allow the analyst to specify and evaluate a model training pipeline implemented in PySpark [8]. The analyst can use the interface to visualize a sample of potentially dirty records and apply data cleaning scripts to different subsets of data. ActiveClean will correctly incorporate those changes back into the trained model, and allow the analyst to iterate. We will present two experimental scenarios where the models are affected by dirty data:

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 vido. 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 reviews. 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 [6].

2.1 What Is New?

Machine learning, specifically active learning, has been applied in prior work to improve the efficiency of data cleaning [12, 13, 4]. Human input, either for cleaning or validation of automated cleaning, is often expensive and impractical for large datasets. A model can learn rules from a small set of examples cleaned (or validated) by a human, and active learning is a technique to carefully select the set of examples to learn the most accurate model. This 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 applying machine learning to facilitate improved data cleaning, and in contrast, ActiveClean explores the problem of the impact of data cleaning on user-specified models. This new problem setting leads to questions of correctness and "push-down" prioritization, i.e., how information about a user's subsequent data analysis (the model that is trained) can inform which data to clean. Existing approaches are designed for homogeneous data (all dirty or all clean) and cannot be applied here. One of the primary contributions of this work is an incremental model update algorithm with correctness guarantees for mixtures of data. The essence of the approach is to model the entire human's iterative process as a Stochastic Gradient Descent loop, which captures many different model types.

2.2 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.

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

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 $r(\theta)$.

2.3 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.

2.4 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)
   (g) detector = DetectorUpdater(detector,
       cleaned_data)
3. Output: current_model
```

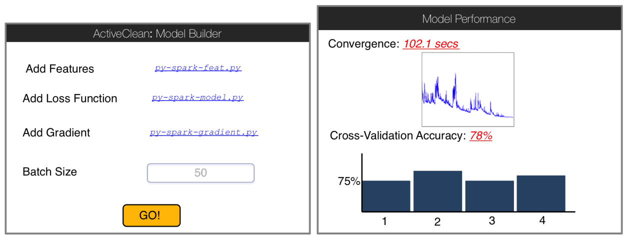
2.5 Technical Details

We overview some of the research contributions and the details can be found in [6].

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.


ActiveClean: Model Builder

Add Features py-spark-feat.py

Add Loss Function py-spark-model.py

Add Gradient py-spark-gradient.py

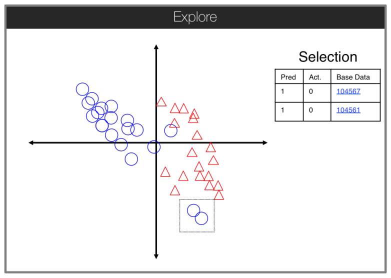
Batch Size 50

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



Pred	Act.	Base Data
1	0	104567
1	0	104561

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

3. THE INTERFACE

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

3.1 Initialization

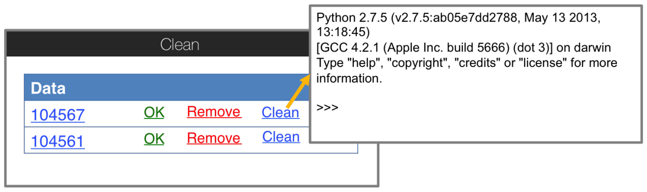
The first part of ActiveClean is called the Model Builder, this is an interface that the analyst uses to specify the problem. She loads three user-defined functions written in PySpark [8] based on the descriptions in the previous section. Optionally, she can change the batch size from our default setting 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

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 [9] to visualize the points in 2D. The batch size setting controls the number of examples plotted on this screen. T-SNE is a non-linear dimensionality reduction technique that is widely used to visualize complex data distributions. In the Diagnose panel, we use color coding to indicate the label in the case of classification. The analyst can select examples from the plot for further inspection.

3.3 Clean Interface

If the analyst decides that the data are actually dirty, then she



```
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: Cleaning Interface. The analyst can choose to remove data, write a custom cleaning operation, or automatically clean the data using an existing cleaning operation.

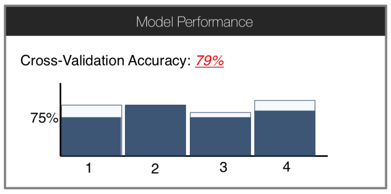


Figure 5: Update Visualization. We visualize the changes in model accuracy after data cleaning.

can open the Clean panel. This panel gives her the option to remove the dirty record or apply a cleaning operation (specified in Python). ActiveClean keeps track of previously written cleaning operations and the analyst need not write a new script each time, and simply assign a previously written transformation. This serves two purposes: (1) it reduces effort for the analyst, and (2) it helps ActiveClean taxonomize the types of errors in the dataset. By understanding which data are similarly corrupted, we can guide future samples to draw similar data in future samples. This may draw data not quite cleaned by the operations described in the script. This optimization is described in detail in our technical report [6]. Essentially, for each distinct script, ActiveClean also trains a classifier to learn the conditions when that operation is applicable.

3.4 Iteration and Updates

Finally, after the batch of data is cleaned, ActiveClean updates the current best model. We re-run the cross-validation and visualize the differences in the model's accuracy (Figure 5). This process iterates until the analyst is satisfied with the accuracy. At that point the system ends, and returns a python pickle of current best model parameters.

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. For the latter, users will clean a dataset they have not encountered, which we hope accentuates the value of an automatic, prioritized sample selection algorithm. Overall, the demonstraint is intended to show how ActiveClean quickly improves the accuracy of the Video Segmentation Model and the Topic Model.

To this end, we will first run the demo in automatic mode. Here, 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, the demo will execute the iterative cleaning loop automatically, and compare both ActiveClean and naive Active Learning. Thus, 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 the resulting models, 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

The growing popularity of predictive ML models in data analytics adds additional challenges in managing dirty data. Progressive data cleaning in this setting is susceptible to errors due to mixing dirty and clean data, sensitivity to sample size, and the sparsity of errors. The underlying statistical problems are subtle and not known to many analysts. The key insight of ActiveClean is that an important class of predictive models, called loss models (e.g., linear regression and SVMs), can be cleaned progressively with guarantees by embedding the process into an incremental optimization loop. 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] 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. Crotty, A. Galakatos, and T. Kraska. Tupleware: Distributed machine learning on small clusters. *IEEE Data Eng. Bull.*, 37(3), 2014.
[4] 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.
[5] G. Inc. Tensorflow. `https://www.tensorflow.org/`.
[6] 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`.
[7] 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.
[8] P. Spark. Pyspark. `https://spark.apache.org/docs/0.9.0/python-programming-guide.html`.
[9] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
[10] 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*, 2014.
[11] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *ICML*, 2015.
[12] 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.
[13] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5), 2011.