Faster Feature Engineering by Approximate Evaluation

Michael R. Anderson
University of Michigan
mrander@umich.edu

Michael Cafarella
University of Michigan
michjc@umich.edu

ABSTRACT

The application of machine learning to large datasets has become a vital component of many important and sophisticated software systems built today. Such *trained systems* are often based on supervised learning tasks that require *features*, or extracted signals that distill complicated raw data objects into a small number of salient values. A trained system's success depends on the quality of its features.

Unfortunately, *feature engineering*—writing code that turns raw data objects into feature vectors suitable for a machine learning algorithm—is tedious and time-consuming. Because "big data" inputs are so diverse, feature engineering is a trial-and-error process with many small, iterative code changes. Because the inputs are so large, *each code change* can involve time-consuming data processing (over each page in a Web crawl, for example). We introduce Zombie, a data-centric system that accelerates feature engineering through intelligent *input selection*, optimizing the "inner loop" of the feature engineering process. Our system yields feature evaluation speedups of **up to 8x** in some cases and reduces engineer wait times **from 8 to 5 hours** in others.

1. INTRODUCTION

The ultimate success of a trained system depends on the ability of features to accurately represent the task-specific variations within the raw data. Commodity features, such as token counts, are often the first attempt at developing a feature set. However, these features are by necessity general: they cannot take advantage of nuances specific to the data or task. Using commodity features as a starting point, a feature engineer can develop additional features tailored to the particular dataset by applying domain expertise [2, 6].[1]

Feature engineering can be a highly iterative, trial-and-error process, because, unfortunately, good features are hard to devise. First, with a large, diverse set of inputs (e.g., a Web crawl), the engineer never quite knows the "input specification" and must repeatedly fix bugs as unexpected raw inputs are discovered. Second, predicting the usefulness of a proposed feature is difficult; a programmer may implement a feature only to throw it away after finding it ineffective. Feature engineers, then, often must make and evaluate many small iterative changes to their code. But evaluating each feature code change can take a great deal of time, since it entails executing the feature code over a very large set of raw input data and retraining the machine learning model. With the research presented here, we address the sheer amount of time needed to execute the *feature code evaluation loop*.

[1]Deep learning methods have great promise for producing high-quality models without traditional explicit feature engineering [7]. However, we believe there will always be a strong role for human-provided domain knowledge.

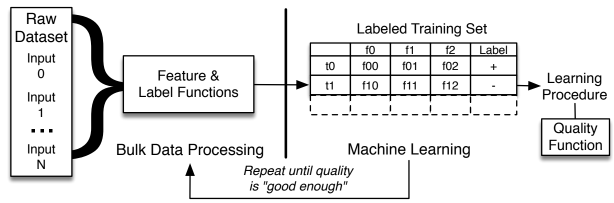

Figure 1: In the feature evaluation loop, bulk data processing and machine learning to date have been commonly implemented as separate systems.

We model the current feature engineering process typically used in practice (illustrated in Figure 1) as a task parameterized by 6-tuple of inputs $(R, \mathcal{F}, L, T, Q, G)$ as follows:

1. Starting with a **raw dataset** R, the feature engineer writes or modifies a set of **feature functions** $\mathcal{F}$. Each function $f \in \mathcal{F}$ takes as input a raw input $r \in R$ and emits a numeric or boolean feature value. The values are concatenated to form an unlabeled feature vector.
2. $\mathcal{F}$ and a **label function** L are applied to R using a bulk data processing system, such as MapReduce, producing a training set of feature vectors, $(\mathcal{F}(R), L(R))$.
3. The training set is provided to the **training procedure** T of a machine learning system. $T(\mathcal{F}(R), L(R))$ produces a trained model.
4. The engineer evaluates the quality of the model using a **quality function** Q (e.g., the model's accuracy over a holdout set). If $Q(T(\mathcal{F}(R), L(R)))$, does not meet the engineer's **quality goal** G, she begins again at Step 1.

In this development model, the engineer can face significant downtime in Step 2, waiting for the feature code to be applied to a large dataset. Even with a distributed processor like MapReduce or Spark, executing complex feature code over millions of data items takes considerable time. Our research focuses on the efficient development and execution of feature functions over very large raw datasets (Steps 1 and 2).

We have implemented Zombie, a system that saves considerable time in the above development cycle by reducing the time cost of generating the features for a suitable training set (Step 2) [3, 4]. Like giving a developer a faster compiler, these savings allow the feature engineer to iterate on feature code quickly; because of this improved engineer productivity, tasks can be completed faster or with a higher quality outcome than under standard development processes.

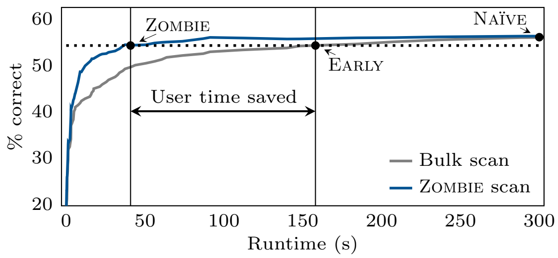

Figure 2: Feature evaluation learning curves. As runtime increases, more raw data are processed for the training set, improving the model's quality. Stopping a Bulk scan early gives a good estimate of model quality. ZOMBIE's scan can stop much sooner.

2. FEATURE EVALUATION AS A QUERY

We view feature function evaluation fundamentally as a data management problem, so we model Steps 1–4 above as a database query, which is executed with each new iteration of the engineer's feature code. The user's feature function $\mathcal{F}$ (from Step 1 above) can be considered a user defined function executed over a large raw data set R and joined with a set of labels L to produce a table of training data (Step 2). The data in this table is then aggregated by training the learning system T and evaluating its accuracy with an evaluation function Q to produce a single quality value (Steps 3 and 4). We might write this as the following SQL:

SELECT $Q(T(\mathcal{F}(R.\text{data}), L.label)$ AS *quality*
FROM rawDataSet R, labels L
WHERE $R.\text{id} = L.\text{id}$

To speed up feature evaluation, then, we need to reduce the query's runtime, much of which is the cost of applying $\mathcal{F}$ to R. A straightforward way to do this is to approximate the query answer by only generating features for a subset of the raw data and training the learning system on this reduced training set. This subsetting routine in itself is not interesting; using just a subset of a large training corpus is a common feature engineering practice already in use. Unfortunately, the ideal size and composition of a data subset is difficult to determine: the amount of data needed to train a model greatly depends on the features representing that data, which continually change as the engineer refines the feature set.

A slightly more sophisticated method of subset construction is to continuously re-train and evaluate the learning system as the training set is populated with newly processed features, and then stop processing the raw data early once some stopping criterion is met (our system stops when the learning curve starts to plateau). In Figure 2, the EARLY data point on the Bulk scan learning curve shows that a great deal of runtime can be saved while still producing an accurate evaluation metric.

3. OUR APPROACH

With ZOMBIE, however, we further reduce runtime by leveraging the fact that in a large dataset, like a Web crawl, only a small portion of the data may be relevant to the learning task and to the engineer's feature functions. If we can limit the processing of the data to these items, a much smaller training set can be constructed, similar to active learning [8]. Unfortunately, standard active learning methods examine the features of potential items, entailing the feature generation runtime that we are trying to reduce.

To avoid this limitation, our system pre-processes the raw data, creating a large number of *index groups*, each containing raw data items similar to one another. Akin to the offline sample creation of approximate query processing systems like BlinkDB [1], ZOMBIE uses raw data items in the highest utility index groups to generate features to answer the evaluation query. Unlike approximate query processors, we cannot know which index groups will be most useful to a particular feature function's evaluation query. A raw data item may produce useful features under some functions, but be irrelevant under others. Thus, we must learn the utility of the index groups online for each evaluation query.

Online learning has a tradeoff between exploration and exploitation: we want to select items from the best input groups, but we must also process items from unknown—and potentially suboptimal—index groups to learn which are best. ZOMBIE uses a multi-armed bandit algorithm to manage this tradeoff: a raw data item is randomly drawn from an index group (selected by the UCB bandit algorithm [5]), processed to generate features for the training set, and used to re-train the learning system. The index group's utility is updated based on the incremental change in the model's quality.

This method of raw input selection is shown by the ZOMBIE scan learning curve in Figure 2. The point labeled ZOMBIE shows how effective this method is at reducing the feature generation runtime over a Bulk scan method, even with early stopping. Our experiments (described in our full paper [3]) have shown up to an 8X speedup over the Bulk scan with early stopping in some cases, as well as reducing engineer wait time from 8 to 5 hours in others, on a series of text classification tasks. ZOMBIE's operation is orthogonal to statistical improvements in learning algorithms, and so could be easily combined with advances in that area.

4. CONCLUSION

The features used for a machine learning task are essential for a successful trained system, and engineering great features is a difficult task. Our research aims to remove some of the tedium and unproductive thumb twiddling inherent in the current practice of feature engineering by providing tools to find the right features for the right data. By intelligently selecting raw data to process for feature generation, ZOMBIE allows the feature engineer to quickly evaluate feature effectiveness for a given learning task, improving productivity and the potential for building a high-quality trained system.

REFERENCES

[1] 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*, 2013.

[2] M. Anderson, D. Antenucci, V. Bittorf, et al. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[3] M. R. Anderson and M. Cafarella. Input selection for fast feature engineering. In *ICDE*, 2016.

[4] M. R. Anderson, M. Cafarella, Y. Jiang, G. Wang, and B. Zhang. An integrated development environment for faster feature engineering. In *VLDB*, 2014.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[6] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78, 2012.

[7] Q. V. Le, M. Ranzato, R. Monga, et al. Building high-level features using large scale unsupervised learning. In *ICML*, 2012.

[8] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.