Machine Learning for Data Unification
Practical Applications in Tamr's Software Platform

by Michael Stonebraker, Chief Technology Officer, Tamr Inc.

I Introduction

Machine learning (ML) is currently a much-discussed topic, and in many instances the enthusiasm for the subject exceeds its practical implementations. Since Tamr began as a research project at MIT CSAIL (the Computer Science & Artificial Intelligence Laboratory) and other institutions in 2012, our goal has been to apply ML to the challenge of unifying data at scale to address what we believe is a pervasive problem for large enterprises that has not been effectively addressed by mainstream data integration tools.

We begin in Section 2 with a brief introduction to ML to get everybody on the same page. We call it "ML for dummies". Then, in Section 3 we describe why ML is of interest in data analysis in general. Section 4 turns to the places in the Tamr product where ML is used. Finally, in Section 5 we discuss issues of Tamr scalability in an ML context.

II ML for Dummies

Suppose an analyst in the human resources (HR) department wants to discover the relationship between salary and age in his enterprise. Although it is plausible to get a complete data set on all employees, this may take a lot of time, and the employee wishes to use a sample which she has in hand. Figure 1 shows such a data sample.

Age	Salary
25	41,000
47	65,000
55	85,000
22	30,000
39	84,000
52	55,000

Figure 1: A Sample of Employee Salaries

There are many ways the employee could proceed, but a simple one is to assume a linear model for this data. In other words, the model is:

Salary = constant-1 + constant-2 * age

Then **linear regression**, an algorithm explained in most linear algebra textbooks and available in most statistical packages, could be used to "fit" the data to this model by finding values for

constant-1 and constant-2.

In this case, the final model is:

Salary = 14,132 + 1147 * age

Any statistics package will also show one how good the model is. For regression, it will show how much of the variance among salaries is explained by the model above. If the predictive value of the model is sufficiently strong, then the employee will conclude that salary is linearly correlated with age. It would also allow her to make **predictions** about salaries. For example she would expect a 49 year-old employee to earn $70,335.

Of course, there are many other factors that might influence salaries, for example gender (hopefully not), job classification, location (one would expect salaries in Manhattan to be higher than salaries in rural Ohio), etc. As a result, the employee could move on to a more sophisticated model based on additional data fields. The employee could also be concerned that she did not have a random sample of employees, which could skew the results. Hence, she might decide to collect more data. She might also try to find industry-wide data from the public web to compare her results against.

The scenario describes a very simple analysis scenario whose goal is insight and prediction. We now relate this example to machine learning.

- In an ML context, the available data is called **training data**.
- The fields of interest are often called **features**.
- The training data is used to fit a **model**, which is chosen by a human.
- The model can then be used for prediction or insight.

If the model is a poor fit to the training data, a different model can be tried or more training data collected.

Often a human will be asked to review the model predictions, perhaps on a sample of data. If the human corrects the model, then this information can be used as additional training data, and the model can be rerun. This is called **active learning**, because the model is improved if additional knowledge is collected. It pays to be smart about which predictions to ask a human to correct. It is clear that one should select the prediction that the model is most uncertain about, since that will produce the maximum corrective effect.

Another concern is **overfitting**. If the number of features is comparable in size to the training data, then there is not enough data available to correctly fit a model. In this case of overfitting, spurious results should be expected, and one solution is to get more training data. Another option is to limit the number of features considered to avoid this issue.

The training data is assumed to be **ground truth**, i.e. its validity is not questioned. Training data is not assumed to be a random sample, although this would be very helpful. Instead active learning is assumed to correct for skew over time.

In summary, ML is about fitting a model to training data. The choice of model is up to a human. For example, we could have used a non-linear function for our analysis, for example:

Salary = constant-1 + constant-2 (age - 20) **2

There are many, many possible models. Recently deep neural networks have received a lot of press and appear to work well in image recongition. Similarly, logistic regression is useful in solving categorization problems, i.e. where an object should be classified into one of a finite set of possibilities. Lastly, Naïve Bayes is a widely used model which produces robust results on a broad class of ML problems. As one can expect, experience is very helpful in deciding which model to use on a particular problem

It is certainly plausible to implement a collection of ML models and have a computer try them out on a training data set and give advice on which one to use, based on quality of fit. However, many models have parameters which require specification by a skilled person, so this exercise still requires a sophisticated analyst at the helm. We will see an example of this issue in Section 4.

The training data is whatever can be collected or is lying around. The features are often a trial and error process to discover a good fit. It is also common to use **enrichment** (joining your data set to another one) to generate additional fields that can be used as features. These topics are further explored in Section 4.

III Why ML Is Important In Data Analysis

Although the previous example was very basic and only required "learning" two constants, things are rarely this simple in the real world. Consider for example, a collection of "spend" transactions, which are records of purchase orders in an enterprise. Figure 2 shows seven example transactions for a hypothetical enterprise.

Amount	Merchant	Location
$20,000	Apple, Inc.	Cupertino, Ca
$3000	Apple	California
$100	Apple Store	Bolyston St. Boston, MA
$500	Best Buy	Manchester, NH
$700	AAPL	San Francisco, CA
$500	Apple Co.	New York
$3000	BEST BUY	Cincinnati

Figure 2: A Collection of "Spend" Transactions

The goal is to map each of these transactions into a classification hierarchy. The root node could be "parts", and underneath this node are various categories of parts, like fasteners, computers, sheet metal, etc. A human can surmise that Figure 2 contains only computer purchases. However, to perform this task automatically, one must realize:

- Anything with Apple in the name regardless of location is a computer purchase, unless it is for $100 or less;
- Capitals and lowercase letters are interchangeable; and,
- Abbreviations should be expanded to their full length.

A machine learning system can learn complex rules such as these, based on example training data. In effect, ML can **extend** examples into a full solution, especially in cases where it is difficult to write down exactly what rules are to be utilized. Hence, when the relationship is "fuzzy", an ML system is very good at teasing out the details. This is often the case with character string data.

A second reason to use ML is the issue of scale. Suppose one wants to classify 20M transactions – something that we see frequently with our customers like GE. One can write 3 rules to classify the 7 transactions above. However, there is no hope of writing enough rules to deal with the number of transactions in the GE use case. Hence, ML can provide a scalable solution, when human intervention fails.

In the next section we turn to Tamr's use of machine learning in its data unification platform.

IV Tamr and ML

There are two main places where Tamr uses machine learning: entity consolidation and entity classification. We discuss each in turn.

4.1 Entity Consolidation (deduplication)

One of the tasks for which GE is using Tamr entails "supplier mastering". Specifically, GE has dozens of procurement systems. The CFO has figured out that GE can save about $1B per year if they can provide the following information to each procurement officer. When a given contract comes up for renewal, locate all the other contracts with the same supplier, and then demand "most favored nation" status. In other words, find the division with the best terms and then demand the same treatment. However, this requires entity consolidation across dozens of supplier databases, each containing several thousand suppliers. In other words, GE needs to deduplicate about 1 million supplier records.

According to GE, there is enormous value to be extracted by optimizing terms with "the long tail" of their supplier base. Procurement officers are very good at negotiating terms for the largest suppliers common across divisions as this is a task that can be accomplished manually by picking up the phone. But this is a 'high friction' process, and they are typically lack the time and resources for such extensive manual collaboration with the long tail of suppliers with less spend. In aggregate, optimizing the long tail is very valuable; in effect it requires "finding a lot of needles in a very large haystack".

Here is the Tamr approach to deduplication. The first step is to look for mechanisms to enrich the data. In this way, extra fields can be added to each record which can assist in the deduplication process. The Dow Jones identifier for each supplier is a popular added field along with a variety of GE-internal data elements.

As a result of this step, each supplier has K attributes of information. The next step depends on the availability of training data. This consists of a collection of pairs of records which a human specifies as matches (i.e. duplicates) and a collection of pairs of records that are non-matches.

To this data Tamr fits a decision tree model in the following manner. For each attribute Tamr requires a **distance function**, D (a1, a2), which specifies how far apart are any two values a1 and a2. In general, a distance function can be user-specified. However, for each character string attributes, Jacard and cosine similarity distance are popular metrics, and Tamr asks a human to choose between these two. For numeric data, Tamr uses arithmetic distance. For each attribute, Tamr then chooses a collection of **split points** based on dividing the training data into L equal sized buckets. Then, for each attribute it tries these L "split points", and greedily chooses the attribute and the split point that most accurately classifies the training data. In effect each of the L* K cases is a predicate of the form:

Attribute-I < split point => non-match
Attribute-I >= split point => match

and

Attribute-I >= split point => non-match
Attribute-I < split point => match

Tamr chooses the predicate that best fits the data at hand. With this "root node" chosen, Tamr continues to greedily fit the two second level nodes. It continues in this fashion until the benefit of additional levels is marginal or until a user-defined maximum depth, Max, is reached. In effect Tamr is fitting a decision tree model to the training data, with parameters D, L and Max. As noted in the introduction, it is important to choose these parameters wisely to get good results.

If there is not enough training data, Tamr uses active learning to get more. Tamr has an optional "cluster review" process. This step allows a human to review suggested matches and to correct ones that are in error. Hence, cluster review produces additional training data to refine the model used, and can be thought of as an active learning scheme.

So far, Tamr has identified collections of records that it thinks represent the same entity, i.e. are duplicates. Consider one particular collection that represents an employee, Michael Stonebraker, as shown in Figure 3.

Michael Stonebraker	73	Boston, Ma
Mike Stonebraker	72	Cambridge, Ma
M. R. Stonebraker	73	Moultonborough, N.H

Figure 3: The "cluster" for Michael Stonebraker.

Often an enterprise wants to find a "golden record" for each cluster. In this case, Tamr needs to process each cluster to a minimum form. For Michael Stonebraker, this would be:

Michael Stonebraker	73	{Boston, Ma. Cambridge, Ma Moultonborough, N.H}

Obviously, we need a canonical form for my name, a resolution for the two conflicting values for my age, and the recognition that I have three different mailing addresses. This is the job of the Tamr golden record system.

Tamr starts with a collection of user-specified column rules which define how to aggregate the column values in a cluster into a "golden value". Tamr supports "choose the most frequent value", "majority consensus", "keep all values" and "choose average value". Based on applying these rules, Tamr reduces each cluster of data to a simpler one with less multi-valued attributes.

Then, Tamr examines each column, looking for for patterns of values. For example, in the Stonebraker cluster, it removes the duplicate value "Stonebraker" and is left with:

M.R.
Mike
Michael

Then, it assumes that longer strings are better than shorter ones, and forms candidate substitution rules, as follows:

M.R. ⟶ Michael
Mike ⟶ Michael

Tamr performs this analysis for each multi-valued field in any column that does not have the "keep all" designator. The net result is a collection of possible rules and a count of the number of times each occurs.

Tamr then sorts the rules into frequency order and presents the first one to a human along with a sample of the clusters to which it applies. The human is asked to respond "yes", "no" or "maybe". Tamr automatically applies or discards the rule in the first two cases. In the third case, it asks a human to start tagging values as "correct" or "not correct". Based on this training data, Tamr then forms a decision tree model for the collection of clusters. This process of examining the most frequent possible rules continues until a human decides that the point of diminishing returns has occurred.

4.2 Entity Classification

One of the common things Tamr is asked to do is record classification. We already mentioned this use case above. For example, GE wishes to classify "spend" transactions into its pre-existing classification hierarchy. Here is how Tamr accomplishes this task.

Tamr requires training data, which can be available as a result of previous work, which can come from active learning, or which can come from user-written rules, which we explained above. In the GE case, they wrote 500 rules (about the maximum number a human can comprehend). These rules classified about 10% of the transactions (about 2M out of 20M). Tamr then used machine learning to fit a logistic regression model, which works well when the task is to put records into buckets. Of course, user review provided feedback for an active learning system. GE uses review for high value transactions.

V Scalability

Consider entity consolidation again. Notice that the algorithm in the previous section is order (N **2) where N is the number of records. In the case of GE, we have 1M records, so the computation is order (10 **12). If we can compare two records and compute a distance in 10 microseconds, then this computation will take 10 ** 7 seconds -- about 100 days. Clearly, we need to do better. Two approaches come to mind, and Tamr uses both of them. First, we can use parallelism to cut down on the latency. If we do 100 way parallelism, then this computation will take 1 day instead of 100 days. This is a start but we need to do better.

A second approach is to divide the data into B buckets, such that the records in each bucket are unlikely to have a small distance to records in a different bucket. Hence, we only need to compare records within each bucket and not across buckets. This reduces the complexity of the calculation from order (N **2) to order (N **2 / B). If B = 1000, then our computation is reduced by another factor of 1000, and the GE example can be performed in 1.5 minutes.

So the problem is: how to define the buckets? Again, Tamr turns to machine learning, and we use a training set of pairs of records that match. Our objective is to find a collection of (possibly overlapping) predicates that can be used to assign records to buckets, each predicate defining a single bucket. Remember from the previous section that each attribute has a distance function D that specifies how far apart two values are. Also, in each distance space, we can define a collection of L separator values that are used in the predicates below.

We examine predicates of the form:

Distance-att-i (value-1, value-2) < separator-1

and

Distance-att-j (value-3, value-4) < separator-2

We try all pairs of the K attributes and all pairs of the L separators, giving us a total of K * (K -1) * L * L predicates of the above form. We greedily choose the predicate that maximizes a cost function, C, of the form:

C = constant-1 * number-of-matching-pairs-of-records-that-satisfy-predicate

Minus

constant-2 * number-of-non-matching-pairs-of-records-that–satisfy-predicate

This cost function tries to put a lot of matching records in the same bucket and to minimize the number of non-matching records that are in the bucket. We have found experimentally that this cost function works well, but we are always exploring possible improvements.

We then remove the records that satisfy the predicate from the training set to get a revised training set. We then search again for the best predicate of the above form. We continue to pick predicates in this fashion until one of two conditions is reached:

We exhaust the training set

or

We reach a predefined stopping point of M predicates.

We then allocate each predicate to a separate bucket, and the bucketing is complete. Notice that the collection of chosen predicates may overlap. In this case a record may be allocated to more than one bucket. On average with real-world problems that Tamr has tackled, each record is allocated to 2-3 buckets, and we typically use a few thousand buckets. At the expense of some duplication (a factor of 2-3) we achieve a speed up of around 1000x, which was used in the example computation earlier.

VI Summary

This short white paper has sketched the current uses of ML in Tamr. As noted, we use logistic regression, decision trees as well as ad-hoc models. All use training data to fit a model, and most allow for human feedback and active learning. Over time, our models will get better as we see more use cases and additional techniques may be employed. In addition, as our product adds additional capabilities (more supported transformations, data cleaning, etc.), other uses of ML will likely emerge. Our quest is always to support scalable solutions which minimize human involvement and provide maximum return on investment for our customers.