Demonstration of Santoku: Optimizing Machine Learning over Normalized Data

Arun Kumar Mona Jalal Boqun Yan Jeffrey Naughton Jignesh M. Patel
University of Wisconsin-Madison
{arun, jalal, byan23, naughton, jignesh}@cs.wisc.edu

ABSTRACT

Advanced analytics is a booming area in the data management industry and a hot research topic. Almost all toolkits that implement machine learning (ML) algorithms assume that the input is a single table, but most relational datasets are not stored as single tables due to normalization. Thus, analysts often join tables to obtain a denormalized table. Also, analysts typically ignore any functional dependencies among features because ML toolkits do not support them. In both cases, time is wasted in learning over data with redundancy. We demonstrate *Santoku*, a toolkit to help analysts improve the performance of ML over normalized data. Santoku applies the idea of *factorized learning* and automatically decides whether to denormalize or push ML computations through joins. Santoku also exploits database dependencies to provide automatic insights that could help analysts with exploratory feature selection. It is usable as a library in R, which is a popular environment for advanced analytics. We demonstrate the benefits of Santoku in improving ML performance and helping analysts with feature selection.

1. INTRODUCTION

Many projects in both industry and academia aim to integrate ML algorithms as well as statistical computing languages such as R with scalable data processing in RDBMSs, Hadoop, and other systems. Almost all implementations require that the input to an ML algorithm be a flat single table. However, most relational datasets are not stored as single tables due to normalization. Thus, analysts often perform joins, especially key-foreign key joins, to bring in more features and *materialize* a single denormalized table.

Proceedings of the VLDB Endowment, Vol. 8, No. 12

Example: Customer Churn (based on [3]). Consider an insurance company analyst modeling *customer churn*, i.e., predicting how likely a customer is to leave the company. She builds an ML classifier using data about customers, including past customers that have churned, which are stored in the `Customers` table with the schema `Customers` (`CID`, `Churn`, `Sex`, `EID`, ...). `EID` is a foreign key that refers to the `Employers` table, which contains information about the customers' employers such as corporations and non-profits. It has the schema `Employers` (`EID`, `State`, `Size`, ...), wherein `Size` indicates how big the employer is in terms of its revenue. Intuitively, the features of the customers' employers could help a classifier predict if the customer will churn. For example, customers employed by large corporations from Wisconsin might be unlikely to churn, in which case including the `State` and `Size` features could improve accuracy. She performs a key-foreign key join to create a denormalized table: `Temp` $\leftarrow \pi($`Customers` $\bowtie$ `Employers`), as shown in Figure 1. She feeds `Temp` to a toolkit that implements a classifier, e.g., Naive Bayes.

The above is a case of "learning after joins" – the analyst is forced to materialize `Temp` because existing ML toolkits cannot process normalized data directly. As we explain in [3], this could introduce redundancy that is avoided by normalization and hurt performance. Moreover, it causes data management burdens for analysts, which could hinder exploratory analyses [5]. Our recently introduced paradigm of "learning over joins", specifically *factorized learning* (FL), helps mitigate these issues by "pushing the ML computations down through joins" to the base tables [3]. In our example, if the analyst provides the base tables (`Customers` and `Employers`) and the join column (`EID`), one can apply FL to Naive Bayes to avoid redundant computations on `Temp`, which could improve performance (see Figure 1; details in Section 2). *Santoku is the first toolkit to offer FL for a set of popular ML models such as Naive Bayes.* While FL avoids redundant computations, it performs extra work for "book-keeping". Thus, FL could be slower than operating over denormalized data on some inputs depending on various parameters of the data, system, and ML model [3]. It will be helpful for analysts if a system could *automatically* decide which tables to join and which to

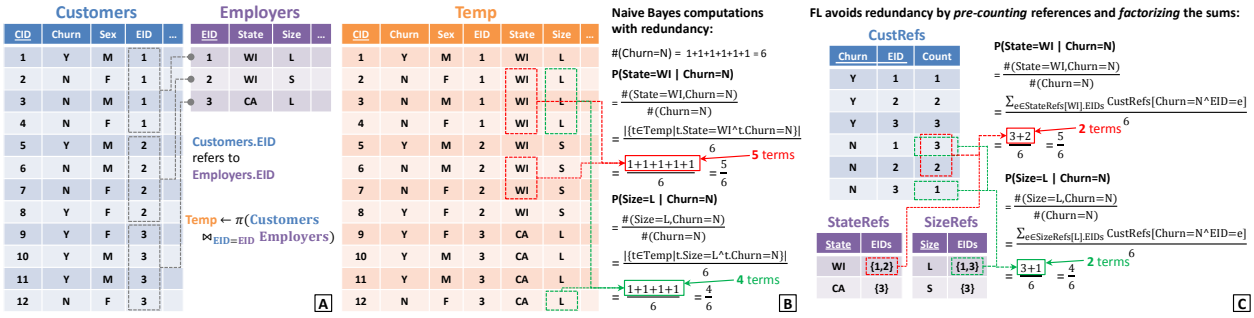

Figure 1: Illustration of Factorized Learning for Naive Bayes. (A) The base tables `Customers` (the "entity table" as defined in [3]) and `Employers` (an "attribute table" as defined in [3]). The target feature is `Churn` in `Customers`. (B) The denormalized table `Temp`. Naive Bayes computations using `Temp` have redundancy, as shown here for the conditional probability calculations for `State` and `Size`. (C) FL avoids computational redundancy by pre-counting references, which are stored in `CustRefs`, and by decomposing ("factorizing") the sums using `StateRefs` and `SizeRefs`.

apply FL on in order to optimize the performance of the ML model over normalized data. *Santoku provides such an optimization capability by using a simple cost model and a cost-based optimizer.*

A closely related scenario is learning over a table with functional dependencies (FDs) between features. For example, we can view `Temp` as having the following FD: `EID` $\rightarrow$ {`State`, `Size`, ...}. This FD is a result of the key-foreign key join.[1] In general, there could be many such FDs in a denormalized table. From speaking to analysts at various companies, we learned that they ignore such FDs altogether because their ML toolkits cannot handle them. While they may not use the database terminology (FD), analysts recognize that such "functional relationships" can exist among features. One can use the FDs to normalize the single table, and then apply FL to different degrees. But once again, these approaches could be slower than using the single table on some inputs. *Santoku enables analysts to integrate such FD-based functional relationships into some popular ML models and automatically optimizes performance.*

Finally, we consider the important related task of *feature selection*. It is often a tedious exploratory process in which analysts evaluate smaller feature vectors for their ML model to help improve accuracy, interpretability, etc. [1, 2]. Ignoring FD-based functional relationships could mean ignoring potentially valuable information about what features are "useful". *Santoku helps analysts exploit FD-based functional relationships for feature selection purposes.* Santoku provides a "feature exploration" option that automatically constructs and evaluates smaller feature vectors by dropping different combinations of sides of FDs. For example, one can drop `EID` (perhaps an *uninterpretable* identifier), or other features from `Employers`, or both. While this may not "solve" feature selection fully, it provides valuable *automatic insights* using FDs that could help analysts with feature selection.

[1]Key-foreign key dependencies are not FDs, but we can view them as such with some obvious assumptions.

Santoku is designed as an open-source library usable in R, which is a powerful and popular environment for statistical computing. R provides easy access to a large repository of ML codes [2]. By open-sourcing our API and code, we hope to encourage contributions from the R community that extend Santoku to more ML models. Many data management companies such as EMC, Oracle, and SAP have also released products that scale R scripts *transparently* to larger-than-memory data. We implement Santoku fully in the R language, which enables us to exploit such R-based analytics systems to provide scalability automatically. For users that do not want to write R scripts, Santoku also provides an easy-to-use GUI, as illustrated in Figure 2.

In summary, we demonstrate Santoku, the first toolkit to integrate common database dependencies with popular ML models. We explain FL with an example in Section 2 and present Santoku's architecture in Section 3. In Section 4, we discuss how demonstration attendees can interact with Santoku to see its benefits.

2. FACTORIZED LEARNING

Factorized learning (FL) was first proposed for generalized linear models [3]. Santoku extends FL to a few other popular ML models, including Naive Bayes. We explain FL with a detailed example using Naive Bayes.

Example: Naive Bayes assumes that the data examples are samples from a (hidden) joint probability distribution $P(\mathbf{X}, Y)$ over the class label Y, and feature vector $\mathbf{X}$. But it also assumes conditional independence among the features, given Y. Thus, it computes the following: $P(\mathbf{X}, Y) = P(Y)P(\mathbf{X}|Y) \approx P(Y)\Pi_{F \in \mathbf{X}} P(F|Y)$. The probabilities are estimated by counting the frequencies of various combinations of features values and class labels in the training dataset. In SQL terms, this involves a bunch of `COUNT` aggregation queries, which could be batched into a single pass over the data.

[2]`http://cran.r-project.org/`

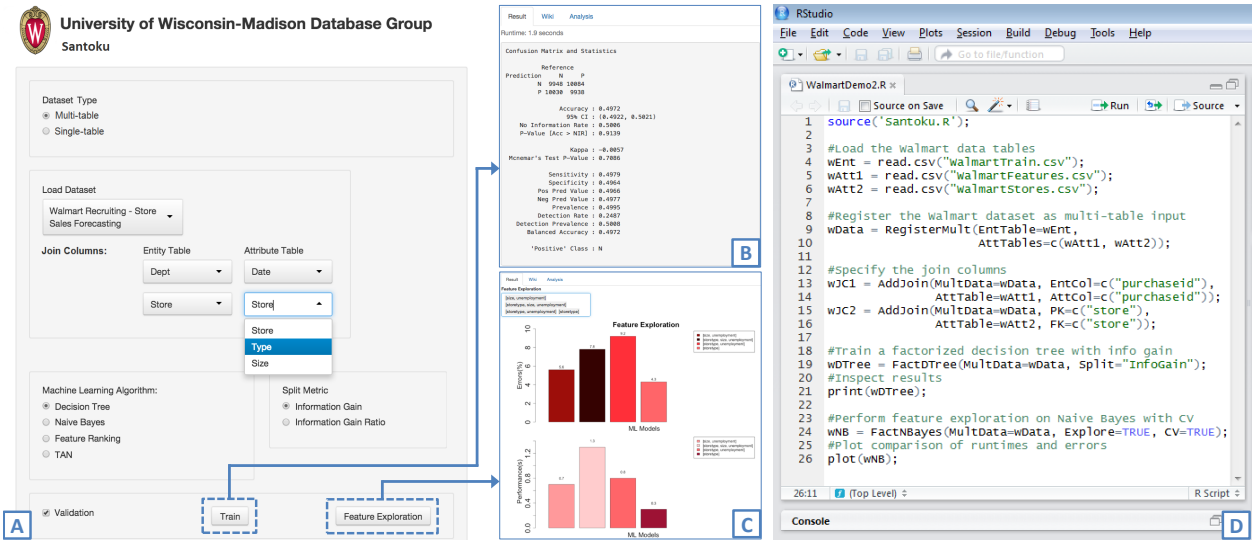

Figure 2: Screenshots of Santoku: (A) The GUI to load the datasets, specify the database dependencies, and train ML models. (B) Results of training a single model. (C) Results of feature exploration comparing multiple feature vectors. (D) An R script that performs these tasks programmatically from an R console using the Santoku API.

Figure 1(A) shows a simple instance of our insurance customer churn example. The output of the join, `Temp`, has redundancy in the features from `Employers`, e.g., values of `State` and `Size` get repeated more often. This results in redundancy in the computations for Naive Bayes when it counts occurrences to estimate the conditional probabilities for those features. Figure 1(B) illustrates the additions needed for both `State=WI` and `Size=L` when operating over `Temp`. In contrast, FL avoids redundant computations by pre-computing the number of foreign key references, and by factoring them into the counting. Figure 1(C) illustrates the reference counts that are temporarily stored in `CustRefs`, which is obtained, in SQL terms, using a `GROUP BY` on `Churn` and `EID` along with a `COUNT`. The list of `EID` values for `State=WI` (and `Size=L`) are also obtained. Thus, we can reduce the sums for those features into smaller sums, viz., 2 terms instead of 5 for `State=WI`, and 2 instead of 4 for `Size=L`. While training computations are reduced, extra work is needed to obtain the pre-aggregated references. Thus, whether or not this approach is faster depends on the data properties, especially the dimensions of the input tables. Santoku uses a simple cost model to pick the faster approach on a given input.

Usually, the learned ML models are also "scored", i.e., their prediction accuracy is validated with a set of test examples. For Naive Bayes, this requires us to compute the *maximum a posteriori* (MAP) estimate on a given test feature vector $\mathbf{x}$ as follows: $argmax_{y \in \mathcal{D}_Y} P(Y = y) \Pi_{F \in \mathbf{X}} P(F = \mathbf{x}_{(F)} | Y = y)$ [4]. Essentially, scoring involves a multiplication of conditional probabilities. Thus, we can exploit the redundancy in scoring as well, not just learning, by factorizing the computations on a test set and pushing them through the joins. We call this technique *factorized scoring*, and we use it in Santoku when the models need to be validated.

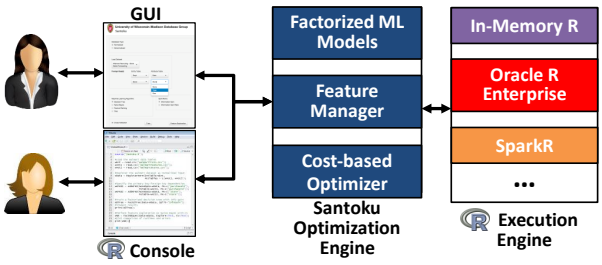

Figure 3: High-level architecture. Users interact with Santoku either using the GUI or R scripts. Santoku optimizes the computations using factorized learning, and invokes an underlying R execution engine.

3. SYSTEM OVERVIEW

We discuss Santoku's architecture (see Figure 3), and explain how it fits into a standard analytics ecosystem.

Front-end: Santoku provides custom front-ends for two kinds of analysts – those who prefer a graphical user interface (GUI), and those who prefer to write R scripts. The GUI is intuitive and has three major portions (Figure 2(A)). The first portion deals with the data – analysts can specify either a multi-table (normalized) input or a single-table (denormalized) input. For normalized inputs, the analyst specifies the base tables and the "join columns", i.e., the features on which the tables are joined (the foreign keys and primary keys in database parlance) using menus. For denormalized inputs, the analyst specifies the single table and any "functional

relationships" among the features (the left and right sides of FDs in database parlance) using menus. The second portion deals with the ML model – they choose a model and its parameters, and can either train a single model or perform feature exploration, possibly with validation. The third portion displays the results – a summary of the execution for training (Figure 2(B)), and plots comparing several feature vectors for feature exploration (Figure 2(C)). Interestingly, we were able to implement Santoku's GUI in R itself using its graphics and visualization libraries. The GUI is rendered in a browser, which makes it portable. Santoku also provides an intuitive API that can be used in R scripts (Figure 2(D)). This enables analysts to exploit Santoku's factorized ML models programmatically. The operations on Santoku's GUI also invoke this API internally.

Santoku Optimization Engine: The core part of Santoku is its optimization engine, which has three components. The first component is a library of R codes that implement factorized learning and scoring for a set of popular ML techniques – decision trees, feature ranking, Naive Bayes, and Tree-Augmented Naive Bayes (TAN) [4] as well as linear and logistic regression using gradient methods [3]. We adapted the implementations of these models from standard R packages on CRAN. We expect to add more as our system matures. The second component is the *feature manager*. It manipulates the feature vectors of the datasets. It handles three major tasks: normalization of single tables using FDs, denormalization by joining multiple tables, and constructing the alternative feature vectors for feature exploration. The third component is a cost-based optimizer that uses a cost model to determine whether or not to use factorized learning and scoring on a given input (given by the analyst, or constructed internally as part of feature exploration). The cost model is calibrated based on the R execution engine.

Back-end: Since Santoku is implemented in R, it simply "piggybacks" on existing R execution engines, with the standard in-memory R perhaps being the most popular. Several commercial and open-source systems scale R to different data platforms, e.g., Oracle R Enterprise operates over an RDBMS (and Hive), while SparkR operates over the Spark distributed engine. Such systems enable Santoku to automatically scale to large datasets.

4. DEMONSTRATION DETAILS

We divide the demonstration into three phases: (1) A brief introduction to factorized learning with an example, and an overview of Santoku. (2) A "hands-on" phase in which we demonstrate Santoku's GUI and API in an R console to show its benefits on multiple datasets. (3) A performance comparison phase in which we present Santoku's performance against naive approaches on both real and synthetic datasets to give a better picture of Santoku's optimizations. We will use animated slides during the demonstration to explain the example. Next, we describe the hands-on phase (the GUI and the R console) as well as the performance comparison phase in more detail.

End-to-end Execution with GUI: We demonstrate both multi-table (normalized) and single-table (denormalized) inputs using real datasets from different application domains – retail, hospitality, transportation, and recommendation systems. A Santoku user can specify the kind of input as well as the join columns (for normalized input) or the functional relationships between features (for denormalized input) on the GUI using dropdown menus that list the features. We will provide the schemas of our datasets to help users choose features. The user can then select an ML model they want and specify its parameters (or use default parameters). Finally, they can run the training. They also have an option of validating the model to check its accuracy on a test set. This step is illustrated in Figure 2(A). The quality and performance results of the execution will be displayed on the screen, as illustrated in Figure 2(B). They can then try feature exploration, possibly including validation. Santoku will automatically create multiple feature vectors and plot their quality and performance results, as illustrated in Figure 2(C). This will demonstrate how Santoku can be helpful in selecting a more accurate subset of features. The user can also change the ML model to see new results displayed, or perform a similar exploration with another dataset.

R Script with Console: We demonstrate how Santoku's API can be used in R scripts. We will show the scripts that are automatically generated by the GUI-based input specifications from the previous phase. The user can edit the scripts in an R console and modify the specifications. We will explain the usage of the functions in our API. This will show how more advanced analysts might interact with Santoku. This step is illustrated with a screenshot in Figure 2(D).

Performance Comparison: We demonstrate the executions of the R scripts on the real datasets with Santoku's optimizer disabled. This will provide a better picture of the performance optimizations performed by Santoku. We present a set of "offline" results based on synthetic datasets that drill into Santoku's optimizer by varying the number of tables joined, the number of tuples, the number of features, different ML models, etc. This will provide users with a better picture of the performance tradeoff space for Santoku's optimizer when operating over normalized data without having to wait for these executions to finish.

5. REFERENCES

[1] M. Anderson et al. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.

[2] P. Konda et al. Feature Selection in Enterprise Analytics: A Demonstration using an R-based Data Analytics System. In *VLDB*, 2013.

[3] A. Kumar et al. Learning Generalized Linear Models Over Normalized Data. In *SIGMOD*, 2015.

[4] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[5] C. Zhang et al. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*, 2014.