DataHub – A Collaborative Dataset Management Platform

The BigData@CSAIL Team

May 13, 2015

1 Introduction

The rise of the Internet, smart phones, and wireless sensors has produced a huge diversity of datasets about all aspects of our lives, from our social interactions to our personal preferences to our vital signs and medical records. Increasingly, researchers and "data science" teams want to collect and collaborate on these datasets. Consider a few examples:

- a geneticist who wants to share and collaborate on genome data with other research groups;
- an athlete who wants to quickly study and visualize his performance data;
- a member of a web advertising team who wants to distill insights from unstructured ad-click data;
- a university scheduling office that wants to share room reservations data with campus;
- a journalist who wants to examine public data related to terrorist strikes in Afghanistan;
- a city that wants to publish transit ridership data;
- a small-shop owner who wants to study their sales data in the context of public economic data

Each of these is an example of a data "enthusiast" or a data sub-community with a need to share, exchange, collaborate on, distill, and/or visualize datasets.

Our thesis is that these (and many more similar) communities are under-served by the current data management and collaboration technologies available. Although there is a wealth of "data science" research addressing stand-alone data analysis issues or building integrated tools for analysis, the dataset management, collaboration, and sharing aspects of these tools are poor, requiring data scientists to use ad-hoc mechanisms to record and reason about datasets. Furthermore, although there is a vast array of "big data" research, the emphasis is on performance, and not on simplifying and automating many of the fundamental bookkeeping and management procedures necessary to facilitate seamless data science.

To this end, we propose **DataHub**, a hosted, collaborative platform for preparing, storing, and analyzing datasets. DataHub*'s goal is to provide a common substrate for members of data science teams to collaborate, analyze, and reason about datasets. Our goal is not, in fact, to replace existing data science tools: rather, we effectively support existing data science tools through much-needed basic infrastructure and management capabilities*.

DataHub stands apart from previous systems in two main ways:

Infrastructure: Hosted *version control* systems like github [9] and bitbucket [10] have demonstrated just how powerful and easy to use version control can be for managing source code and unstructured data. DataHub provides analogous features for large structured datasets. (Thus, in a sense, DataHub is like "github for data".) Like these systems, DataHub provides the ability to create public and restricted-access datasets, search for datasets, create new versions of datasets in a branching tree of version histories, and merge versions. In addition, like github, DataHub is a hosted repository that may contain many datasets created by many different users or collaborating teams. However, DataHub operates differently from github in two key respects. First, datasets are structured, consisting of keyed records with multiple typed, named fields (our specific data model is somewhat less rigid than the relational model, in that all records in a dataset are not required to conform to the same schema.) Providing efficient and scalable versioning mechanisms in such an environment is a significant challenge. In addition, fine-grained differences between versions presents a number of problems related to managing and querying *provenance*. Second, DataHub doesn't require that users "check out" versions of datasets to manipulate them; rather, users can directly query and manipulate versions *in situ* using transactions, a SQL-like language, and a suite of tools we describe below, on the DataHub platform. Providing such concurrent access by many users to a large number of datasets in a hosted environment is also a substantial challenge.

Support for Data Analysis: Building upon the unique capabilities of the DataHub infrastructure, Data-Hub provides a range of functionality aimed at facilitating and simplifying data analysis. The purpose of this functionality is to reduce the burden on data scientists, so that they do not have to deal with low-level data management concerns, freeing their time to focus on data exploration. Key features include:

- *Dataset cleaning and transformation tools*, designed to allow users to progressively refine datasets by adding more and more structure to them, and by removing inconsistencies and outliers. Without automatic data cleaning, users must resort to cumbersome scripting tools to transform the data into a structured form amenable to analysis. Our hybrid data representation allows users to store, represent and reason about datasets whose schemas are evolving.
- *Dataset search and augmentation tools* that enable users to search over a large collection of public and shared hosted datasets; and to integrate and collectively analyze the retrieved datasets and their own datasets. Without this functionality, users must manually search on the web for relevant datasets (without knowing if they will actually be useful), and integrate them by hand with existing datasets.
- *Dataset visualization tools*, for automatically showing users the most interesting aspects of their data, and on publishing visualizations as structured derivatives of datasets. Without in-situ visualization, users will have to rely on exporting data into visual analytics tools just to view simple trends in data.

Since data scientists often use home-grown tools for data analysis, DataHub enables the use of the provided functionality and infrastructure in conjunction with any of these tools or other applications through the versioning API. For instance, data scientists can use the DataHub infrastructure and functionality in conjunction with Matlab, R, or SAS. Further, DataHub enables users to ask "meta-questions" about the analysis workflow itself by automatically tracking the transformation and modification steps (irrespective of whether performed inside or outside DataHub), and capturing detailed provenance information.

Although many organizations have begun efforts to centralize data sets into "data pools" or "data lakes", these efforts typically focus simply on collecting organizational data, rather than trying actually support collaborative access and manipulation of data sets by data scientists. By supporting collaborative features like versioning, annotation, and search, DataHub aims to do much, much more.

2 DataHub **Versioning Features**

The core of DataHub is a hybrid storage system that supports a flexible record-based representation, combined with a versioning system that enables keeping track of versions of a dataset as well as dependencies between datasets and derived products. The other capabilities are built on top of the storage system, and make heavy use of the versioning and provenance features provided by it. DataHub is a hosted platform, designed to be run as a server that many clients use to store their data; there will both be a public DataHub site, as well as potentially many private DataHubs run by organizations.

Storage model. DataHub needs to be able to ingest structured, semi-structured, and unstructured data and allow users to clean it into a structured representation. We adopt a simple model where every record in a table has a *key*, and an arbitrary number of typed, named attributes associated with it. Records are not required to all have the same number/type of attributes.

The second abstraction in DataHub is that of a *dataset*. A dataset consists of a set of tables, along with any correlation/connection structure between them (e.g., foreign key constraints). A directed graph over the datasets captures relationships between versions as well as the *provenance* information that records user-provided and automatically inferred annotations for derivation relationships between data.

Versioning API. The versioning interface in DataHub provides a simple versioning API similar to what version control systems like svn and git provide. These include commands to create, checkout, branch, and merge datasets, and commit updates and rollback changes. Version control systems like git and svn do not operate at the level of structured data, and hence are inefficient; on the other hand, current database systems only support a linear chain of versions, which is inadequate for sharing and collaboration. We describe both these aspects in detail in §**3**.

This basic API can be used by end users and DataHub components such as the data cleaning and integration tools. For example, a user who uploads some genomic data from a spreadsheet and then does cleaning and integration with other datasets would first create the new dataset, propagate and commit raw data to it, and then perform several branching steps as the cleaning and integration are completed.

Query Language. We plan to support SQL as a query and data manipulation language, with a collection of flexible operators for record splitting and string manipulation, including regex functionality, similarity search, and other operations to support the data cleaning engine, as well as arbitrary user-defined functions. The SQL query processor will use the sub-table index to find sub-tables that can *participate* in a query. A sub-table participates in only those queries for which it contains all fields.

Users are not required to compute over data using DataHub: programs are free to copy data out of the system, operate on it however they wish, and then copy it back in (as a new dataset or a new version).

We also plan to provide a graph-like search language over provenance metadata to find datasets of interest, e.g., find all datasets that used a specific input tuple which we found to be erroneous.

Sharing and Exporting. In addition to a basic query/manipulation language, DataHub will include a variety of fine-grained access control features that allow datasets to be declared public/private and readonly/writeable by different users and groups. In addition, we will provide language bindings that make it possible to access DataHub datasets from popular query languages, including Javascript, which will enable sharing of web pages that directly access and manipulate persistent data.

Use Cases. Data versioning is of critical importance in many data science applications. Consider the problem of a team developing a new analytic for detecting features in brain signal data such as EEG. This will involve labeling regions of signals (e.g., regions where the patient is experiencing an event like a seizure, or where the leads are disconnected), and then developing features to predict these labels for unlabeled regions (e.g., based on the energy in particular frequency bands, or the correlation of signal between different leads.) Dataset versioning is valuable here as it lets scientists work in a sort of private sandbox to develop these features and create these labels without interfering with what other scientists are doing. Once these features are developed, they can be merged back into the main dataset for use by other researchers. In addition, sharing features will make it possible to perform this kind of iterative data science in collaboration with external researchers, e.g., people at MIT, or to ingest data from external locations, e.g., hospitals with medical devices.

3 Data Analysis Features

In addition to building the basic DataHub platform with the features outlined above, we plan to focus on several additional problems: automatic visualization and extraction of structured data from unstructured or partially structured text.

3.1 Automatic Visualization.

In traditional data management or visualization systems, given a query, users need to either manually sift through query results, looking for valuable insights, trends, or anomalies; or they need to visualize the query results in many different ways before the "insights" stand out—both options are ad-hoc and time-consuming. One of our goals is to enable users to quickly get visualization suggestions in-situ on the DataHub platform without having to do much upfront work.

To illustrate, consider the following workflow, which we believe is often used in practice. *Step (1):* The user poses a relational query to select the subset of data they are interested in. In a medical recordsscenario, the user may select all records associated with the keyword "diabetes". *Step (2):* Then, the user considers several candidate views over this subset of data, formed by running aggregation and grouping operations; these views must be studied one by one. For example, one view may be total patients records mentioning "diabetes" by patient age, another view may be the total diagnoses by month. Since these views have two-attributes each, they can be viewed as 2-dimensional graphs. For example, Figure 1(a) may be the total mentions (y-axis) by month (x-axis), while Figure 1(c) may be the mentions (y-axis) by age (x-axis); the axes are not important and are omitted. *Step (3):* Next, the user steps through each view, to decide which views are "interesting." This is the time-consuming step. What makes a view like Figure 1(a) interesting? It depends on the application semantics and what the user is comparing against. For example, Figure 1(a) shows decreasing treatments over time. If treatments for all diseases are down then this view is not very interesting. However, say Figure 1(b) shows the aggregate (all) product treatments over the same time period. Then the diabetes treatments view goes against the general trend. In this case, the view is *potentially* "interesting" because it depicts a trend in the subset of data that the user is interested in that *deviates* from the overall one. Figures 1(c) and 1(d) illustrate a different type of deviation. The first shows the distribution of expenditure by treatment type for diabetics, while the second shows the overall expenditure across all diseases. Again, the "diabetes" trend does not follow the general trend: the types with the most treatments overall are not the types that occur frequently with diabetes mentions.

Instead, we propose to automate *some especially cumbersome aspects* of this search for interesting insights. To eliminate the laborious process of stepping through all possible views that the user has to currently manually perform, we can highlight "potentially" interesting views (or visualizations) of the query results.

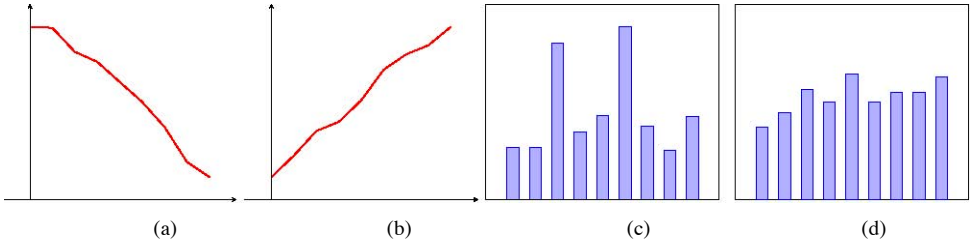

Figure 1: Views (a), (b): Treatments over Time. (c), (d): Treatments by Type.

using automated mechanisms based on deviation. These visualizations will help the user quickly interpret and understand specific "interesting/useful" aspects of the query result.

Doing this will require us to tackle a number of challenges, including:

- Techniques to efficiently explore the space of possible visualizations. This will require looking at how every set of attributes varies with every other subset – a hugely exponential search space – so some kind of pruning / search heuristics will clearly be needed.
- Techniques to incorporate *user feedback* in the exploration process – e.g., to learn what is interesting and what isn't and suppress uninteresting views.
- Metrics to rank the "interestingness" of visualizations, e.g., based on how much variability there is in different attributes, and how they vary over time or with respect to other attributes.
- Techniques to generate approximate visualizations, e.g., that closely conform to the shape of a visualization that would be produced if all data in a dataset was analyzed, but without actually looking at every record.

3.2 Extracting Structure from Text.

While DataHub will natively support unstructured or semi-structured data, there are significant advantages to transforming the data into a structured form: specifically, visualizing and manipulating data becomes much more straightforward if the data is structured. The process of converting data into a structured form amenable to analysis, called *data wrangling*, is acknowledged by data scientists to be tedious and time-consuming, delaying analysis [4]. Thus, in DataHub, we provide in-built data cleaning tools to allow data scientists to get from raw semi-structured data to structured data which can then be analyzed.

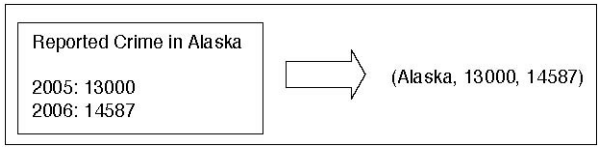

Figure 2: Input, Output Example.

Current Work. We begun to develop an initial cleaning tool called Distill that targets the case when there is only ordered records of one type in the dataset. It starts by asking users to provide a few examples (i.e., input records and their mappings to output records). As an example, consider the following input record, selected from [5], shown in Figure 2. For the input record consisting of three lines on the left, the user has chosen to provide the output record: (Alaska, 13000, 14587). Distill uses the input records to learn a set of *all most specific transformations* (i.e., a transformation that is as specific and detailed as possible) that extract all of these examples correctly: each transformation consists of a sequence of patterns that alternately extract and discard portions of the record. For instance, in the example above, we infer a discard pattern for 'Reported Crime in', an extract pattern for 'Alaska', a discard pattern for '\n 2005:', and so on. Once Distill identifies these most specific transformations, it applies each transformation on the entire dataset, and then picks the transformation(s) that correctly extracts from all records. For datasets with simple structure like the one above (we discuss next structures that we cannot currently handle), Distill can correctly extract the desired fields.

Proposed Work. Our focus will be on incorporating the following functionality to enable Distill to wrangle, transform, and clean a much larger space of datasets, all in a limited number of passes over the data (to promote efficiency).

Generalization on Dataset: At all times, Distill maintains a set of all most specific transformations that correctly extract all the examples provided by the user. Then, when parsing the actual dataset, we may find that none of the most specific transformations are able to parse the new records correctly. For instance, in our example above, we may find that the attribute value corresponding to 2005 is missing in some records. Here, we must infer that transformations that extract from the second line for the second attribute of the output record, are indeed incorrect. Further, we must generalize our set of most specific transformations to be more tolerant to records for which we may not be able to extract the value corresponding to 2005.

Noise Detection: When there is more than one record type in the dataset, or if there is noise or exceptions, simply applying generalizations is not sufficient in some cases and may in fact lead to errors — often the record is so different from the ones encountered previously that we cannot directly use the transformations identified so far. Thus, a key issue is to determine if we need to generalize our transformations or identify a new type of record or noise. In such a case, we can ask the user to label "confusing" records. Cleverly maintaining a sample of records such that we do not store too many records that look similar, yet at the same time ensuring that we have adequate coverage to learn a transformation for each record type (without knowing the number of types in advance), will be a challenge.

Active Structure Determination: Human input is certainly needed for records of a completely new type, but even for records of the same type, we may benefit from having additional human input to correctly select between equivalent transformations. We will leverage crowdsourcing principles [3, 6, 7, 8] to identify the records with the highest "information gain" to label that will allow us to discard as many of our set of most specific transformations as possible, or will lead us to generalize as many as possible.

Further, we can allow the user to flag errors for incorrectly extracted records in the extracted output. We can then leverage that input to further improve our extraction. However, an error flag gives us significantly less information than a labeled example: in such a scenario, we can rule out all most specific transformations that extract the same way, but it is harder to pick the "next best" transformation.

Assisted Data Cleanse: Once the transformation is complete, there are many straightforward "cleaning" procedures that we can employ to clean the data even further. For instance, we can automatically detect (soft) functional dependencies and missing or incorrect values. However, in DataHub, we do not clean data unless supervised by the user. Here, as well, there are interesting challenges in allowing users to interactively clean the data with the least effort. Prior work on automated data repairs, e.g., [1, 2], has investigated the case when there is no user input; in such cases, the system may end up making unintended errors. In typical cases, there are a few categories that these repairs fall under; having users verify the most critical or impactful repairs would significantly reduce the number of unintended errors.

Leveraging Semantic Information: Once structure is identified, users can be asked to provide the target schema for the final structured data, and given the target schema, DataHub's cleaning tool will automatically identify a complete sequence of steps to transform semi-structured data to the final schema. However, this step can be significantly challenging if the user provides an attribute in the target schema that is not explicitly present or mentioned in the semi-structured data.

To mitigate this hurdle, we propose to leverage a *knowledge graph*, i.e., a *knowledge base* organized as a graph, to interpret user schemas. Open-source knowledge bases such as YAGO [12] or Freebase [11] could be easily leveraged for this purpose. Our knowledge graph can be viewed as comprising of *entities* and their properties (e.g., the entity "Canada", with its population and GDP), and *concepts* that describe information about types of entities (e.g., the concept "Country").

References

[1] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, pages 268–279, 2011.

[2] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[3] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD Conference*, pages 61–72, 2011.

[4] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

[5] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[6] A. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. CrowdScreen: algorithms for filtering data with humans. In *SIGMOD Conference*, pages 361–372, 2012. Online: http://doi.acm.org/10.1145/2213836.2213878.

[7] A. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, pages 1203–1212, 2012. Online: http://dl.acm.org/citation.cfm?id=2396761.2398421.

[8] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5):267–278, 2011. Online: http://dl.acm.org/citation.cfm?id=1952376.1952377.

[9] http://github.org. GitHub (retrieved Oct. 14, 2013).

[10] http://www.bitbucket.org. Bitbucket (retrieved Oct. 14, 2013).

[11] http://www.freebase.com. Freebase (retrieved Oct. 14, 2013).

[12] http://www.mpi-inf.mpg.de/yago-naga/yago/. YAGO (retrieved Oct. 14, 2013).