Building Blocks for Exploratory Data Analysis Tools

Sara Alspaugh
UC Berkeley
alspaugh@eecs.berkeley.edu

Archana Ganapathi
Splunk, Inc.
aganapathi@splunk.com

Marti A. Hearst
UC Berkeley
hearst@berkeley.edu

Randy Katz
UC Berkeley
randy@eecs.berkeley.edu

Abstract

Data exploration is largely manual and labor intensive. Although there are various tools and statistical techniques that can be applied to data sets, there is little help to identify what questions to ask of a data set, let alone what domain knowledge is useful in answering the questions. In this paper, we study user queries against production data sets in Splunk. Specifically, we characterize the interplay between data sets and the operations used to analyze them using latent semantic analysis, and discuss how this characterization serves as a building block for a data analysis recommendation system. This is a work-in-progress paper.

1. INTRODUCTION

Visual data exploration is a key part of data analysis, but it remains ad hoc, requiring intensive manual intervention to identify useful information. Although many tools exist for cleaning and visualizing data, the intermediate step of determining which questions to ask and how to visualize their results requires a combination of deep domain knowledge and a willingness to try many approaches. For users without domain knowledge who are tasked with gleaning insights from a mass of data, the first step to understanding which aspects of the data are important, interesting, or unusual often requires involves iteratively applying standard techniques and interpreting results. Users with domain knowledge or very specific questions in mind already know which techniques yield important insights and based on the semantics of their problem and their data. As a result, domain knowledge lets users consider fewer possibilities and quickly progress to deeper questions.

Leveraging the facts that:

(1) data exploration often involves applying common techniques, therefore different data exploration scenarios typically have overlapping analysis steps, and

(2) domain experts are likely to have a better idea a priori what is important to investigate, limiting the amount of work they have to do to reach later stages of analysis,

our goal is to build a tool that could make intelligent recommendations to users as to which data exploration actions to take. The anticipated result is that users with time constraints or without domain knowledge can make quick and efficient work of data exploration and be well on their way to formulating hypotheses, assessing assumptions, and planning their modeling approach.

This tool would make data exploration more efficient by drawing user attention to more fruitful paths of inquiry and providing users with intuitive control over exploration operations. Such a tool should draw on a combination of classic domain-agnostic exploratory data analysis techniques and more domain-specific techniques learned by observing what other users with similar data sets found useful. It should use these along with modern methods for determining when the information resulting from a technique was interesting or unexpected [16]. It should also provide intuitive mechanisms for interacting with the output of such techniques for hands-on exploration. Making high-quality suggestions regarding which actions to take and reacting intuitively to user input requires a thorough qualitative and quantitative understanding of the data analysis process. Such suggestions are prerequesites to applying recommendation, searching, and ranking techniques to the problem of efficient exploratory data analysis.

As a first step toward creating such a tool, we study records of data analysis activity from Splunk users. (Splunk is a commercial data analysis product.) The logs contain queries from thousands of users, written in the Splunk query language. These logs provide information only about what a certain set of users, primarily IT professionals, do with certain types of data, primarily IT system data. Moreover, they represent only a portion of the actions the users perform with one particular tool, and as such, only tell us about a particular portion of the data analysis process. Nonetheless, these queries provide a valuable starting point to quantitatively characterize the data exploration process. This characterization is a precursor to acquiring the intuition and data needed to build an intelligent data exploration recommendation tool.

In this paper, we focus on the interplay between certain data set features and the analysis operations applied to those data sets, and discuss how this information could be used by a recommendation tool. In particular, we adopt a technique from natural language processing and use it to measure similarity between data set column names and the ways in which they are used. We then discuss how such a technique could be extended for use by intuitive, intelligent data exploration

IDEA'13, August 11th, 2013, Chicago, IL, USA.
Copyright 2013 ACM 978-1-4503-2329-1 ...$15.00.

tools. We begin, in Section 2, by providing context for the vision we outlined above and relate this idea to existing tools. We describe Splunk and the data set we explore in Section 3. In Section 4, we elaborate on the methodology we use analyzing the ways different types of data is used and present our main result. We discuss future directions in Section 5. Lastly, in Section 6, we outline the goals for building a recommendation system for exploratory data analysis.

2. RELATED WORK

The field of information visualization and data exploration is very large; this section focuses on related work on tools for automated suggestions for portions of the exploratory data analysis process. Many intuitive user interface features that would be ideal to have for an exploratory data analysis tool are available in Tableau [1] which is descended from earlier research in exploratory data analysis such as Polaris [14] and Mackinlay's earlier work [9]. Tableau includes a "Show Me" feature, which is meant to reduce the burden on users by automatically creating a good visualization of a view of their data. The user selects a tuple of columns, and Tableau returns a list of possible visualizations templates that could be used with these columns. It does this by automatically detecting whether columns are numerical or categorical, and suggesting all visualizations which are compatible with the selected column types. However, despite the ease with which users can use Tableau to create attractive visualizations, even with the "Show Me" feature, they are still left with the task of manually indicating which columns of the data to visualize and which of several visualization templates to use to visualize them, and this is done one tuple at a time. For large data sets which have many aspects to explore, this overhead could be significant.

The vision for VizDeck is very similar to that which we have described here in that it aims to automatically suggests visualizations for exploratory data analysis [8]. VizDeck supports seven visualization types, such as histograms and scatter plots, and a few transformations, such as filter. It generates all type-compatible visualizations for a given data set, then ranks them. VizDeck ranks visualizations by recording a few statistical measures for each visualization when a user indicates that a certain visualization is interesting, for example, the entropy of the columns involved. It then attempts to predict unseen scores from these statistical measures based on their similarity to other highly ranked visualizations. This differs from our strategy, which, as described in Section 5, is to predict what transformations or visualizations are desirable for a data set based on its similarity to other data sets for which transformations and visualizations have been recorded. Moreover, while generating all type-compatible visualizations works for small data sets, this approach may not scale to larger data sets and larger sets of visualizations. (See Section 6 for a discussion of this.) However, it is valuable to compare to and build from the techniques used in this and the systems described above.

SAGE is a collection of tools for constructing graphical designs by selecting graphical elements, mapping them to data, combining them, and for finding and customizing prior designs. [11]. Published work on SAGE touches upon a number of ideas related to those we present here. For example, users of SAGE can partially sketch out designs in a number of ways and SAGE attempts to intelligently infer remaining properties, like the mapping of data to visual elements. In addition, SAGE provides users the ability to browse and search related designs based on criteria such as the graphical elements used and the data types of the data mapped to these graphical elements.

DataWrangler is a tool for facilitating data cleaning, in particular, data reformatting [6]. Although this use case is slightly different and possibly subsumed by exploratory data analysis, DataWrangler is relevant in that it uses statistics to disambiguate user actions and rank the most likely desired operation as suggestions to the user. It defines a set of data cleaning transformations and upon user input, provides some guesses as to which of these transformations the user might want to take to re-format the data.

In a similar vein, Profiler is a tool for data quality assessment, for instance, finding missing values, outliers, and misspellings [7]. Profiler is related because data quality assessment is again a subset of exploratory data analysis, and because it provides automatic visualization suggestions for anomaly assessment.

Earlier work took more of an artificial intelligence approach, rather than a data-centric approach. Schiff developed a method to automatically create cognitively motivated visualizations from first principles using very small data sets [12]. St. Amant and Cohen developed a knowledge-based planning system called AIDE to help users strike a balance between conventional statistical packages and automated systems which do not ask the user for any kind of input [13]. Casner automatically designed graphic presentations based on an analysis of a logical description of a task the user was attempting to undertake [4]. More recently, other authors have used an ontology to enumerate valid data mining processes [2].

A common feature of these systems is that they attempt to provide intelligent suggestions or recommendations as part of a broader purpose. As the literature on techniques recommendation systems is very large, the techniques used to date by exploratory data analysis systems represent only a small fraction of proposed approaches [5]. However, they do represent some of the first few attempts to apply recommendation in the domain of visualization.

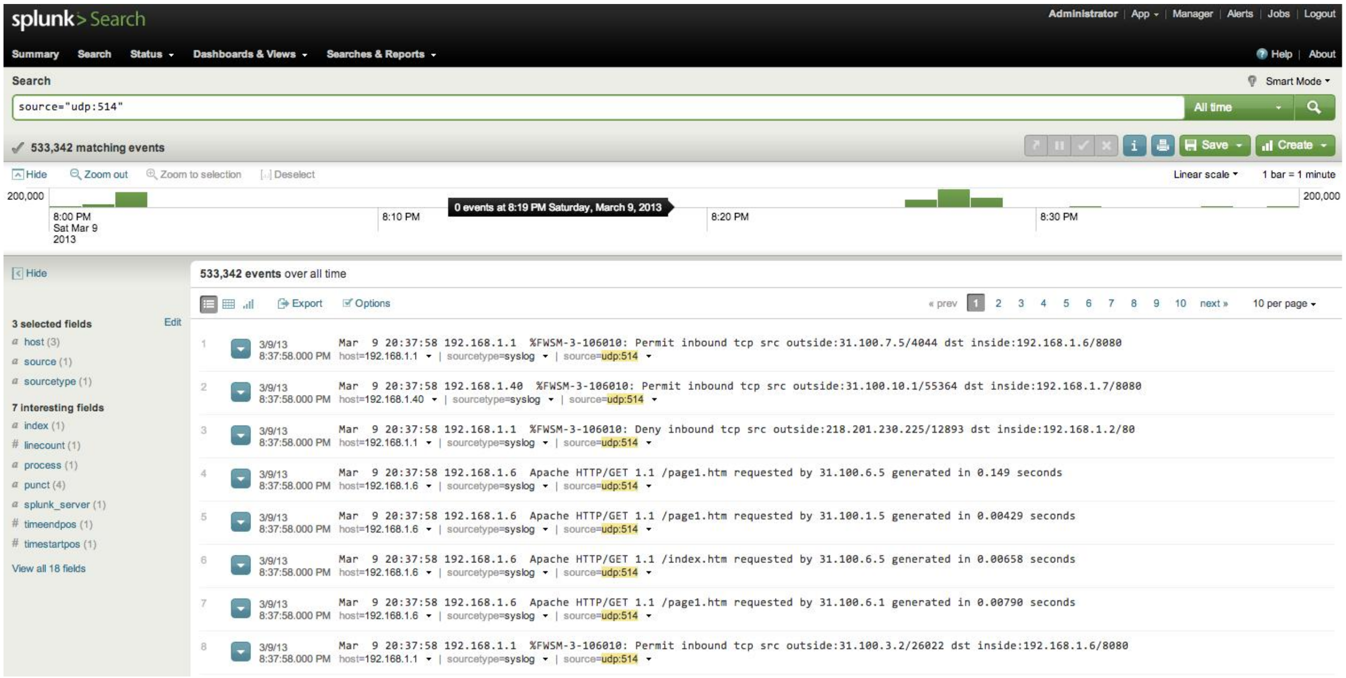
3. CASE STUDY: SPLUNK QUERIES

To begin to quantitatively describe our point in the space of data analysis pipelines, we collected queries from users of Splunk[1]. Splunk is a platform for indexing and analyzing large quantities of data from heterogeneous data sources, especially machine-generated logs. Customers use Splunk for a variety of data analysis needs, including root cause failure detection, web analytics, A/B testing and product usage statistics. Consequently, the types of data sets indexed in Splunk also span a wide range, such as system event logs, web access logs, customer records, call detail records, and product usage logs.

3.1 Definitions and Overview

Table 1 lists the terminology and definitions introduced in this section. To use Splunk, the user indicates where the data they want Splunk to collect and index is located. For example, data might be in a log directory on a file system or collected from a remote server via a certain port. Upon collection, Splunk organizes this data temporally into

[1] `www.splunk.com`


splunk> Search
Administrator | App | Manager | Alerts | Jobs | Logout
Summary Search Status Dashboards & Views Searches & Reports
Help | About
Search
Smart Mode
All time
Save
Create
Hide Zoom out Zoom to selection Deselect
Linear scale
1 bar = 1 minute
200,000
0 events at 8:19 PM Saturday, March 9, 2013
8:00 PM Sat Mar 9 2013
8:10 PM
8:20 PM
8:30 PM
Hide
Export
Options
« prev 1 2 3 4 5 6 7 8 9 10 next »
10 per page
Edit
View all 18 fields

```
source="udp:514"
```

533,342 matching events

533,342 events over all time

3 selected fields
- host (3)
- source (1)
- sourcetype (1)

7 interesting fields
- index (1)
- linecount (1)
- process (1)
- punct (4)
- splunk_server (1)
- timeendpos (1)
- timestartpos (1)

```
1  3/9/13              Mar  9 20:37:58 192.168.1.1  %FWSM-3-106010: Permit inbound tcp src outside:31.100.7.5/4044 dst inside:192.168.1.6/8080
   8:37:58.000 PM      host=192.168.1.1 | sourcetype=syslog | source=udp:514
2  3/9/13              Mar  9 20:37:58 192.168.1.40  %FWSM-3-106010: Permit inbound tcp src outside:31.100.10.1/55364 dst inside:192.168.1.7/8080
   8:37:58.000 PM      host=192.168.1.40 | sourcetype=syslog | source=udp:514
3  3/9/13              Mar  9 20:37:58 192.168.1.1  %FWSM-3-106010: Deny inbound tcp src outside:218.201.230.225/12893 dst inside:192.168.1.2/80
   8:37:58.000 PM      host=192.168.1.1 | sourcetype=syslog | source=udp:514
4  3/9/13              Mar  9 20:37:58 192.168.1.6  Apache HTTP/GET 1.1 /page1.htm requested by 31.100.6.5 generated in 0.149 seconds
   8:37:58.000 PM      host=192.168.1.6 | sourcetype=syslog | source=udp:514
5  3/9/13              Mar  9 20:37:58 192.168.1.6  Apache HTTP/GET 1.1 /page1.htm requested by 31.100.1.5 generated in 0.00429 seconds
   8:37:58.000 PM      host=192.168.1.6 | sourcetype=syslog | source=udp:514
6  3/9/13              Mar  9 20:37:58 192.168.1.6  Apache HTTP/GET 1.1 /index.htm requested by 31.100.6.5 generated in 0.00658 seconds
   8:37:58.000 PM      host=192.168.1.6 | sourcetype=syslog | source=udp:514
7  3/9/13              Mar  9 20:37:58 192.168.1.6  Apache HTTP/GET 1.1 /page1.htm requested by 31.100.6.1 generated in 0.00790 seconds
   8:37:58.000 PM      host=192.168.1.6 | sourcetype=syslog | source=udp:514
8  3/9/13              Mar  9 20:37:58 192.168.1.1  %FWSM-3-106010: Permit inbound tcp src outside:31.100.3.2/26022 dst inside:192.168.1.6/8080
   8:37:58.000 PM      host=192.168.1.1 | sourcetype=syslog | source=udp:514
```

Figure 1: The default GUI view displays the first several events indexed, with extracted fields highlighted on the side, and a histogram of the number of events over time displayed along the top. The user types their query into the search bar at the top of this view.

events by delineating events based on their timestamp, and processes these events using a MapReduce-like architecture, details of which can be found in [3]. Splunk does not require that the user specify a schema for this data, as much log data is semi-structured or unstructured, and there is often no notion of a schema that can be imposed on the data a priori. Rather, *fields* and *values* are extracted from events at run time based on the *source type*. Specifically, when a user defines a new source type, Splunk guides the user in constructing regular expressions to extract fields and values from each incoming raw event. Splunk includes a query language for searching and manipulating data and a graphical user interface (GUI) with tools for visualizing query results. The queries we collected were written in this language.

Users almost always compose such queries in the GUI. The default GUI view displays the first several events indexed, with extracted fields highlighted on the left hand side, and a histogram of the number of events over time displayed along the top. A screen shot of this default view is shown in Figure 1. The user types their query into the search bar at the top of this view. The query consists of a set of *stages* separated by the pipe character, and each stage in turn consists of a *command* and *arguments*. We refer to a query with arguments removed as a query *template*. Splunk passes events through each stage of a query. Each stage filters, transforms or enriches data it receives from the previous stage, and pipes it to the subsequent stage in the query, updating the results that are displayed to the user as they are processed. A simple example of a query is a plain text search for specific strings or matching field-value pairs. A more complex example can perform more advanced operations, such as clustering the data using k-means.

We manually grouped all of the operations users perform in Splunk into a set of *use cases*. By use case, we mean an abstract way in which an argument can be used. For instance, an argument can be used as a filter, aggregated, sorted on, or grouped by, and each of these are use cases. Table 2 lists all use cases we encountered, along with examples and the shorthand labels we use to refer to each. Note that these use cases do not represent all possible use cases in Splunk, but rather are just those we encountered in our examination of a subset of arguments used in the queries.

When the user enters a query that performs a filter, the GUI updates to display events which pass through the filter. When the user uses a query to add or transform a field, the GUI displays events in updated form. Most queries result in visualizations such as tables, time series, and histograms, that appear in the GUI when the query is executed. Users can specify these visualizations by typing queries in the default view, as described above, and can also create "apps," which are custom views that display the results of pre-specified queries, possibly in real time, which is useful for things like monitoring and reporting. Although the visualizations users can create in Splunk do not represent the full breadth of all possible visualizations, they still capture a useful set of standard and commonly used ones.

3.2 Splunk Query Details

The Splunk query language is modeled after the Unix `grep` command and pipe operator. Below is an example query that provides a count of errors by detailed status code:

`search` error `| stats count by` status `| lookup` statuscodes status `OUTPUT` statusdesc

In this example, there are three stages; `search`, `stats`, and

Term	*Definition*
event	a raw, timestamped item of data indexed by Splunk, similar to a tuple or row in databases
field	a key corresponding to a value in an event, similar to the concept of a column name
value	part of an event corresponding to a certain field, similar to a particular column entry in a particular row
query	a small program written in the Splunk query language, consisting of pipelined stages
stage	a portion of a query syntactically between pipes and conceptually a single action or operation
command	the part of a stage that indicates what action or operation to perform on the data
operation	an abstract category made of similar actions or commands, for example, filter or aggregate are operations
argument	the parts of a stage that indicate what fields, values, or option values to use with a command
use case	an extension of the concept operation to account for how an argument is used in an operation
template	a query string with arguments removed, like a skeleton or scaffolding

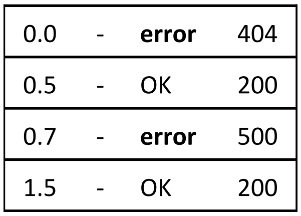
Table 1: Terminology describing Splunk data.

Use case	*Example*	*Label*
filter events containing *value*	`search` *value*	FILTER VALUE
filter events on values of *field*	`search` *field* `=200`	FILTER ON FIELD
queries sorted by *field*	`sort` *field*	SORT BY
field projected	`table` *field*	PROJECT
field renamed as *field*	`rename foo as` *field*	RENAME
field passed to `top`	`top` *field*	TOP
field aggregated	`stats count` *field*	AGGREGATE
grouped by *field*	`stats count foo by` *field*	GROUP BY
field used as function domain	`eval` *field*`=(foo+bar)/1024`	DOMAIN
arg used in arithmetic transformation	`eval foo=`*arg*`*100`	ARITHMETIC
arg used in conditional	`eval foo=case(bar>bat, bar,`*arg*`)`	CONDITIONAL
field used in other transformation	`eval foo=tonumber(`*field*`)`	FIELD IN TRANSFORMATION
value used in other transformation	`eval foo=replace(bar,`*value*`)`	VALUE IN TRANSFORMATION
value passed to option	`head limit=`*value*	OPTION

Table 2: Use cases, examples, and the labels for each in Figure 4. For each argument, the frequency with which it appeared in each use case was tallied up and applied in the LSA calculation described.

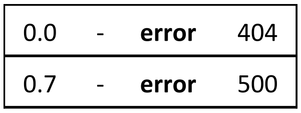
`lookup` are the commands in each stage, `count` `by` and `OUTPUT` are functions and option flags passed to these commands, and “error”, “status”, “statuscodes”, and “statusdesc” are arguments. In particular, “status” and “statusdesc” are fields.

To see how this query operates, consider the following toy data set:



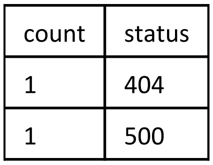
0.0	-	**error**	404
0.5	-	OK	200
0.7	-	**error**	500
1.5	-	OK	200

The first stage filters out all events not containing the word “error”. After this stage, the data looks like:



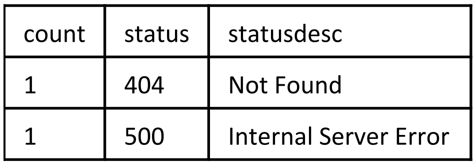
0.0	-	**error**	404
0.7	-	**error**	500

The second stage aggregates events by applying the `count` function over events grouped according to the “status” field, to produce the number of events in each “status” group.



count	status
1	404
1	500

The final stage performs a join on the “status” field between the data and an outside table that contains descriptions of each of the codes in the “status” field, and puts the corresponding descriptions into the “statusdesc” field.



count	status	statusdesc
1	404	Not Found
1	500	Internal Server Error

We collected 50,000 such queries from users of a cloud installation of Splunk. The data set consists of a list of query strings along with an anonymized unique identifier for the issuing user and the time of issuance. Table 3 summarizes some basic information about this query set. To

Total users	602
Total queries	50,000
Unique queries	7,012
Parseable unique queries	5,813

Table 3: Characteristics of the set of queries analyzed. These metrics give some measures of the size of the data set. Note that the approximately 17% of the data set that is not parseable are queries that contain infrequently used commands or queries that are malformed. We plan to extend the parser to support these infrequently used commands in the near future.

parse these queries, we custom-wrote an open source parser for this query language so that it could be separately available from the closed-source Splunk system. This parser is capable of parsing 63 most common commands of 128 seen in our dataset, thus parsing a large proportion of gathered queries.

It is important to note that we do not have access to any information about the data sets over which the queries were issued because these data sets are proprietary and thus unavailable. Having access only to query logs is a common occurrence for data analysis, and algorithms that can work under these circumstances are therefore important to develop. Further, by manually inspecting the queries and using them to partially reconstruct some data sets using the fields and values mentioned in the queries, we are fairly certain that these queries were issued over multiple different (albeit similar) sources of data, thus suggesting the results presented here will generalize across different datasets.

4. CHARACTERIZING DATA ANALYSES

At a high level, the recommendation problem is: given a data set to explore, what visualizations and other such information should the tool suggest to the user? This problem can be viewed abstractly as one of finding the right mapping from points in the space of all possible data sets to points in the space of all possible exploratory data analysis visualizations; however, this formulation is a difficult multi-class multi-label classification problem. Moreover, we currently lack the data to solve this problem as stated above since we don't have access to full user data sets, nor the "correct" labels, which are in this case, complete and user-approved exploratory data analysis visualizations. However, the user query data from Splunk still allows us to explore one key idea regarding the solution of this problem, which is that users are likely to explore and analyze semantically similar data sets in similar ways. This suggests that a reasonable strategy for making exploratory data analysis visualization recommendations for a given data set is to recommend the same things that users found useful for other, semantically similar data sets. We discuss one way of measuring semantic similarity between data sets and the operations applied to them.

4.1 Latent Semantic Analysis

We borrow a technique from natural language processing called latent semantic analysis (LSA) [10]. To use LSA, we start with a matrix of frequencies called the term-document matrix M. In this matrix, M_{ij} is the number of times term i appeared in document j. It is often useful to re-weight the terms of this matrix using factors like tf-idf [10]. We then find low-rank approximation to this matrix using singular value decomposition (SVD). This yields three matrices $U\Sigma V^T$, which can be used to compute points corresponding to the terms and documents projected onto a lower-dimensional space. A schematic of this process is shown in Figure 2. The reason this is useful is that whereas the original matrix lists only the number of times each term actually appeared in each document, LSA provides access to all terms that are relevant to each document by capturing how similar terms and documents are to one another.

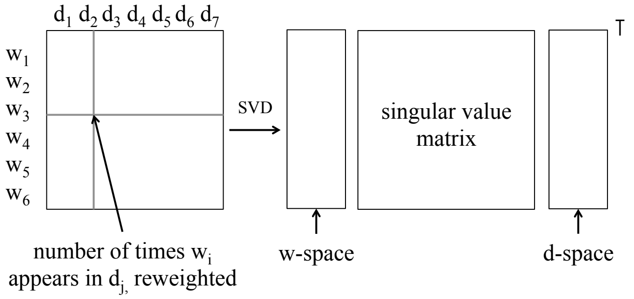


Figure 2: LSA starts with a frequency matrix for which each entry M_{ij} represents a count of the number of times word i appears in document j. After some optional re-weighting, SVD is applied to this matrix to obtain an approximation of the original frequency matrix that expresses relationships of similarity between the words and documents. This similarity between elements is measured by the distance between their corresponding points in the lower-dimensional space.

We apply LSA by replacing the term-document matrix with an argument-use case matrix, so that M_{ij} now contains the number of times argument i was used in use case j, where argument and use case are used in the sense defined in Section 3. We consider only the arguments that are used by more than three users and in more than three different query templates. We primarily consider only arguments that are fields (i.e., that represent column names), for reasons that will be clear later. To generate the frequency of arguments in use cases, we first parse each query and extract out all arguments. Then, guided by a classification of Splunk commands into more generic operation categories (e.g., the `search`, `regex`, and `where` commands are classified as filters) we manually inspect each query and develop a set of rules for deciding which use case a given argument occurrence falls into. We then encoded these rules to process the queries and count the the frequency of arguments in use cases. These steps, summarized in Figure 3, yield the desired frequency matrix. The idea is that applying LSA to these frequencies will capture how similar arguments and use cases are to one another.

4.2 Results

The results of applying LSA to our problem after re-expression, as described above, are depicted visually in Figure 4. There are a number of interesting observations. Some

```
1. Parse queries.
   search source=eqs7day-M1.csv
     | eval description=
       case(depth<=70, "Shallow",
              depth>70 AND depth<=300, "Mid",
              depth>300, "Deep”)

2. Extract arguments meeting criteria.
   return [source, description, depth]

3. Tally usage frequencies.
   if filtered_as_field(“source”):
      usages[“source”].filtered_as_field += 1
```

Figure 3: Analyzing queries is challenging because they are essentially small programs. This figure shows the steps we used to compute frequencies from the query data for use with LSA. We first parsed the query to extract arguments. In the figure, arguments are italicized. Those that are fields are also underlined. The rest of the query that is part of the command is in bold. We select arguments that are used by at least three users and in at least three query templates. We then tally the number of times each argument appears in each of the use cases listed in Table 2.

apparently semantically similar arguments are placed close to one another; for example, `uid`, `user_id`, and `user` get placed close together. Some arguments that are used in similar ways are placed close to one another; for example, `60` and `1024`, which are both used to convert values, are close together. Use cases that get used with similar arguments are placed close to one another; for example, TOP and AGGREGATE are placed close together (note that `top` is basically a sorted aggregation), and PROJECT and FILTER ON FIELD are placed closed together. Some arguments are somewhat ambiguous; for example, the types of `login` and `level` are hard to guess, and also, many of these arguments could mean different things in different data sets. Many arguments for which a connection is not immediately apparent get placed close together. This could be because they really are all used similarly, or because we have chosen to project onto a two dimensional space for the purpose of visualization, where a higher dimension would have made this distinction. Lastly, some use cases are placed farther away from the main cluster of points. This is generally because such use cases occur more frequently than the others, and also intuitively means that such use cases are distinguished because they represent an important dimension. Often with LSA the two most frequent and distinct use cases will be place on opposite axes from one another.

While some of these observations could be made from examining the raw frequencies, or from reasoning about the nature of the operations performed, the value of LSA is in uncovering similarity that is not explicitly captured in the raw frequencies. The hypothesis put forth in this section is that the intuition underlying the application of LSA to terms and documents carries over to this domain, and the initial results presented here provide evidence that qualitatively supports this. Thus, the intuition here is that the raw frequencies describing which contexts an argument appears in represents a set of constraints that reflect the similarity of meaning among arguments, and the underlying structure of meaning is revealed by LSA. This suggests that we can learn more from LSA than we could from simply examining raw frequencies and operation types.

To better illustrate and provide context to these results, we highlight a subset of arguments for which we can partially reconstruct plausible example data sets. Two such example data sets, which together include 15 of the arguments that we analyzed, are shown in Table 4. These two data sets are similar in that they both fall under the domain of IT data sets. However, they are still distinct data sets from distinct sources. The idea is that we can leverage the similarities between these data sets to apply operations that make sense even if the data is from a new source we that haven't seen yet.

4.3 Discussion

In order for LSA to work, we needed to make several adjustments to the raw frequencies we computed. In particular, because skew in frequencies or skew in document length (in our case, frequency of use case occurrence) causes undesirable behavior, we had to correct this skew. The undesirable behavior is that most points are "crushed down to zero," that is, almost all of the points in the lower-dimensional space get crushed down to a small space around the origin, with a few points placed far apart down opposite axes. To avoid this, we even out the distribution of frequencies by applying common techniques, including removing arguments that occur too little or too often, re-expressing frequencies using logarithms, and re-weighting frequencies using tf-idf. Such re-expression to find a scaling of the data that produces an even distribution – with points level and symmetrical – is a reoccurring point of emphasis in Tukey's original exposition on exploratory data analysis [15]. Tukey emphasizes the need for re-expression to make it easier to comprehend, especially through visualization:

> "A basic problem about any body of data is to make it more easily handleable by minds. The grasping is with the eye and the better job is through a more symmetric appearance."

This is consistent with our experience, in that without re-expressing the frequencies, LSA yielded a difficult-to-interpret lower-dimensional projection, as described above.

5. NEXT STEPS

We can extend LSA further, adapting an extension called latent semantic indexing (LSI) [10]. Using LSI in the standard way, the lower-dimensional space can be queried using a set of terms to find documents most relevant to them. Doing this requires computing the centroid of the terms in the queries, and returning those documents which are closest to that centroid. We can extend this idea for our problem domain in an analogous way. Instead of using the frequency of arguments in certain use cases as done in the analysis described above, we can use the frequency of arguments in specific functions rather than operations. The functions would be more specific than operations and also callable, so, for example, in place of using AGGREGATE like we do here, we would specify specific aggregating functions like `count` or

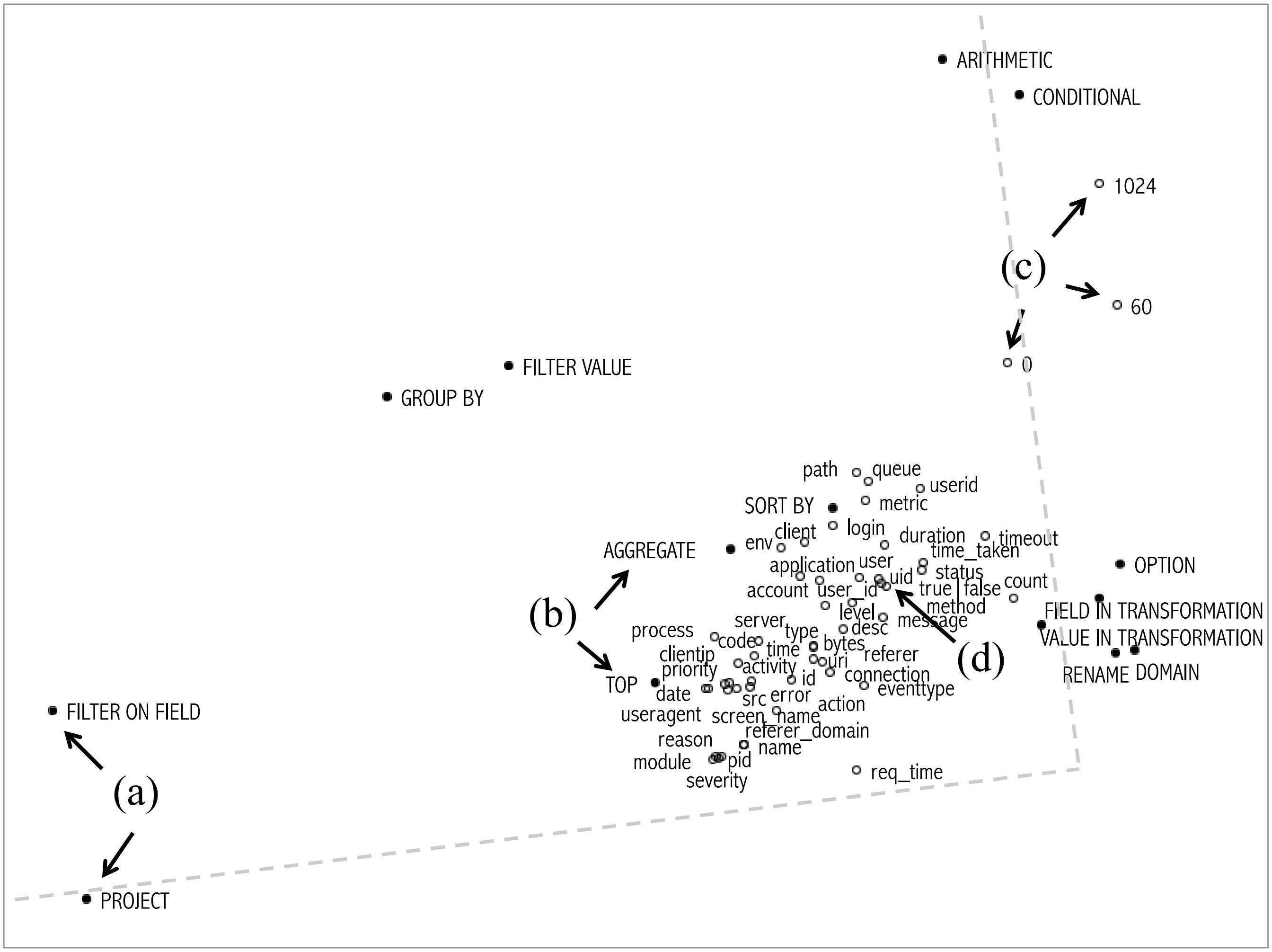


Figure 4: This figure depicts the frequency of arguments (white circles) used in different use cases (black dots) projected onto a two dimensional subspace using SVD. The nearer two points, the more similar they are. We can use this fact to read off some interesting observations about the data from the figure. For example, this figure shows that based on how the arguments are used, the use cases PROJECT and FILTER ON FIELD are similar (a). Likewise, AGGREGATE and TOP are similar, which makes sense as top is a sorted aggregation (b). The numerical arguments `1024` and `0` and `60` are similar, and also are more commonly used as an argument to ARITHMETIC than GROUPED BY (c). Also, the probably-similar arguments `uid`, `user_id`, and `user` get placed close together (d). Lastly, points which get placed further out on an axes, sketched in the figure as dotted lines, are interpreted as being an important dimension, and use cases that are on opposite axes get used with "opposite" arguments, in a way. Thus we see that FILTER ON FIELD is an important use case, and that it tends to get used with different arguments than ARITHMETIC does.

`average`. When a user uploads a data set, we will process that dataset with LSA as well, and then determine which functions to apply to that data set by looking up the column names of that data set using LSI and determining the functions which are closest to them.

This approach suffers from the problem that if a column name does not appear in the term set, it can't be looked up. To get around this, we can extend column names using more context in the form of *fingerprints*. These are extended descriptions of columns that would include, in addition to the the column name, other attributes such as column type, and indicator variables indicating which other columns are present. The fingerprint could also include pipeline context such as previous stages in the query. Then, we will define a new distance function between fingerprints, and use this to lookup the fingerprint or the centroid of fingerprints which are closest to that with the missing column. For example, we would look for matching column types if the queried fingerprint contains a column name that has not been seen before, or look for fingerprints with column names that are near-matches to the column name in the queried fingerprint. A special approach would have to be adopted for operations that take additional arguments, such as filters. One possibility is using partially bound operations for these cases.

Hypothesized data set: Web access logs

sourcetype	_tm	method	stat	useragent	uri
access_*	0	GET	200	Pingdom	/login
access_*	1	GET	404	NewRelic	/apple-touch-
access_*	2	GET	200	Pingdom	/favicon.ico

Hypothesized data set: Mobile metrics

module	metric	device	user
http	memory	ios	foo
mysql	cpu	ios	bar
docs_api	memory	android	bat

Table 4: Two hypothesized data sets partially reconstructed using queries. These are meant as an illustration of how the arguments analyzed relate to the underlying data set.

In addition to extending LSA, there are additional building blocks that we plan as future work to facilitate building such a recommendation system:

- a characterization of the types of operations users perform on system data, to prioritize actions to support for a given domain,
- an identification of common analysis patterns, which suggest what to prioritize so that the common cases for system data analysis can be made fast and intuitive,
- a set of methodologies for analyzing the data analysis process,
- studies examining the behavior of data analysts in various domains, using a variety of tools.

In addition, we plan to evaluate specific recommendation approaches, including algorithms to rank various actions and visualizations that effectively encapsulate the transformations being recommended. This effort will require substantial quantitative and qualitative analysis of queries as well as the data sets to which the queries referred. To this end, we plan to characterize a variety of data sets and evaluate the recommendation system prototype's effectiveness in suggesting useful analysis actions for these.

6. RECOMMENDING VISUALIZATIONS

The analysis presented here are important steps toward creating more intelligent data analysis tools. An effective tool to assist in data exploration via intelligent suggestions requires mechanisms for recommending visualizations, in addition to an intuitive interface to allow the user to provide feedback on suggestions, indicate changes to make, explore data in more detail, or take control over the creation of visualizations. This tool should support modern versions of the techniques articulated by Tukey in his original text on the topic [15], such as histograms, scatter plots, and run charts, which have now become standard practice, in addition to newer types of visualizations. As a sketch of how such a tool might work: the user uploads their data, and initially receives a set of visualizations to browse, ranked in some fashion. Ideally, the tool would be able to identify and prioritize visualizations which conveyed some unexpected information, possibly by using recent techniques for anomaly detection with explanations [16]. The user could choose to inspect certain visualizations further, or provide feedback to the tool about what he or she would like to see. For example, the user might indicate a desire to see more visualizations for a certain subset of the data, or certain type of visualizations for different subsets of the data. The tool should also have some means for inferring which visualizations were most valued by the user to use that information when making future suggestions.

Ranking would be preferable over suggesting all type-compatible visualizations. For example, even if we consider only suggesting two-dimensional scatter plots for a simple data set consisting of n numerical columns, there are $n^2 - n$ possible visualizations. With each additional data type, visualization type or data transformation that we consider, of which there are at least dozens, if not hundreds, the space of possible suggestions increases by an additional $O(n^k)$ term, where k is the dimensionality of the possible visualizations. Thus, for realistic data sets, all possible type-compatible visualizations would number in the tens or hundreds of thousands and would overwhelm the user.

Instead, suggestions should be selected to achieve both breadth and relevance. Breadth of suggestions is important for covering as many different types of questions as possible and giving the user wide set of options for moving forward with the exploration, which will depend in part on their goals. For example, the tool should not just suggest scatter plots at first, or only suggest time series charts. One way to achieve breadth using LSI is to recommend operations that provide good coverage of the space; i.e. that are spread far apart.

Relevance is important in two senses, first in that the suggestions must be compatible with the type of the data. For example, categorical values that happen to be integers should not be treated as numerical values and averaged. Second, the suggestions should make semantic sense for the data. For example, with server performance data, while it is type-compatible to visualize 10th percentile response time over time, users with such data are much more likely to derive value from viewing the 95th or 99th percentile response time over time. One way to achieve relevance using LSI is to recommend operations that are near by fingerprints from semantically similar data sets.

7. CONCLUSION

In this paper, we motivated the need for intuitive, intelligent data exploration tools that reduce the overhead of manual visualization decisions, and instead enable the user to spend more time learning from the data. One possible reason that this vision has not been fully realized is that it is hard to extract the right features from exploratory data analysis examples. To progress towards this vision, it is important to understand the process of data analysis on a deep level. We contribute towards this understanding with a study of data analysis queries from Splunk. We were able to make progress because Splunk programs are concise instances of people analyzing data. The results of this analysis provide some indication that our goal is practical. In particular, we demonstrate promising evidence that what a data set is, semantically, influences what analysis operations are performed, in Section 4. These results suggest that we can recommend similar types of visualizations for similar types of data sets. In other words, we can suggest analysis operations based on measuring the semantic similarity between a given data set and data sets for which "good" analyses are known. In Section 6, we outline ideas for what it would mean to suggest "good" visual analyses and what a tool that did this might look like.

8. REFERENCES

[1] Tableau software. www.tableausoftware.com.

[2] Abraham Bernstein, Foster Provost, and Shawndra Hill. Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *Knowledge and Data Engineering, IEEE Transactions on*, 17(4):503–518, 2005.

[3] Ledion Bitincka, Archana Ganapathi, Stephen Sorkin, and Steve Zhang. Optimizing data analysis with a semi-structured time series database. In *SLAML*, 2010.

[4] Stephen M Casner. Task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics (TOG)*, 10(2):111–151, 1991.

[5] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 2011.

[6] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM Human Factors in Computing Systems (CHI)*, 2011.

[7] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Advanced Visual Interfaces (AVI)*, 2012.

[8] Alicia Key, Bill Howe, Daniel Perry, and Cecilia Aragon. Vizdeck: self-organizing dashboards for visual analytics. In *ACM International Conference on Management of Data (SIGMOD)*, 2012.

[9] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)*, 5(2):110–141, 1986.

[10] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[11] Steven F Roth, John Kolojejchick, Joe Mattis, and Jade Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 112–117. ACM, 1994.

[12] Michael Schiff et al. *Designing graphic presentations from first principles*. University of California, Berkeley, 1998.

[13] Robert St. Amant and Paul R Cohen. Intelligent support for exploratory data analysis. *Journal of Computational and Graphical Statistics*, 7(4):545–558, 1998.

[14] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.

[15] John Tukey. *Exploratory data analysis*. Addison-Wesley, 1977.

[16] Kiri Wagstaff, Nina Lanza, David Thompson, Thomas Dietterich, and Martha Gilmore. Guiding scientific discovery with explanations using DEMUD. In *Conference on Artificial Intelligence (AAAI)*, 2013.