Towards a Data Analysis Recommendation System

Sara Alspaugh, Archana Ganapathi
alspaugh@cs.berkeley.edu; aganapathi@splunk.com
University of California, Berkeley; Splunk, Inc.

Abstract

System data is abundant, yet data-driven decision making is currently more of an art than a science. Many organizations rely on data analysis for problem detection and diagnosis, but the process continues to be custom and ad hoc. In this paper, we examine the analytics process undertaken by users to mine large data sets, and try to characterize these searches by the operations performed. Furthermore, we take a first stab at a methodical process to automatically suggest operations based on statistical analysis of previous searches performed.

1 Introduction

With the data deluge generated by modern systems, several organizations mine their data for insights to enable data-driven decision making. However, data mining is still fairly ad hoc and the lack of automated analysis and visualization tools appears to be hindering data science, requiring each organization to develop custom tools and techniques.

Data analysis typically starts with preprocessing, normalizing, and filtering the raw data, then transforming the data to a format that can be input to the algorithms being used or calculations being performed, and lastly computing the metrics to facilitate the decision making. Although the algorithms and calculations are fairly standard, the preprocessing step must be customized for each dataset, causing a bottleneck in the rest of the analysis process.

Extracting useful analysis from data requires deep domain knowledge about the data and the system generating the data. One must have a specific set of things to look for in the data. Relying on generic techniques for analysis can result in insights being left undiscovered. Specifically, correlations within a data set or across two data sets can provide more useful insights about the state of a system rather than treating each individual data set as an isolated piece of information.

A good goal is to automatically perform analysis operations on the data without semantic knowledge about the data or the system. This would be equivalent to mimicking the actions of a data scientist without his/her data-specific knowledge. Achieving this goal requires searching through the space of possible operations that can be performed on the data, and finding the subset of interesting or "insightful" operations. As a result, we require (1) a policy for exploring the space of possible operations, (2) a metric for deciding which operations to present to the user and (3) a policy for incorporating user feedback to iterate on the analysis.

In this paper, we present a first cut at exploring the space of analysis operations for data sets. We examine user search logs from Splunk, a platform for indexing and searching data. We profile these searches and identify the characteristics of searches people use to extract insights from their data. Since a majority of these searches are targeted at monitoring production systems and used for problem detection and diagnosis, this is an important step towards understanding the data-driven decision making behind system monitoring.

2 Overview

Splunk is a platform for indexing and searching large quantities of data from heterogeneous data sources, especially machine-generated logs. It has a MapReduce-like architecture, details of which can be found in [3]. It includes a search language for querying and manipulating data and a GUI with visualization tools. Our first major goal is to build a semi-automated data exploration tool on top of this platform. We first give a brief overview of Splunk, and explain why it is a good choice for our initial efforts. We subsequently elaborate on the structure of the search language. Lastly, we give a high-level sketch of our vision for what a semi-automated data exploration tool will look like in this context and suggest some possible approaches.

There are several key characteristics of Splunk that make it especially useful for monitoring and troubleshooting:

1. Events are organized temporally—as most problems have symptoms visible in a particular time window, troubleshooting is often driven by time and a temporal data lay out facilitates such analysis.
2. No schema is required at index time—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

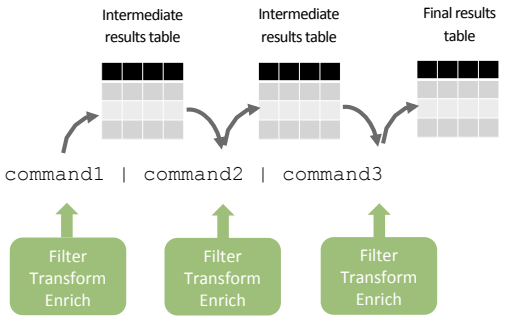

Figure 1: A search consists of a pipeline of operators which operate in sequential stages. Each stage of a search filters, transforms or enriches data it receives from the previous stage, and pipes it to the subsequent search stage.

priori, as the structure of the data may evolve over time. Splunk only requires the presence of a time stamp and an event delimiter. All other structure is imposed at search time based on what is being extracted from the data.

3. Heterogeneous data sources can be cross-correlated—most often, diagnosing the root cause of a problem requires coalescing data from multiple log files that capture information from various system components. Splunk inherently coalesces various data sources by time, and allows people to interleave data using specific fields as "primary keys".

4. Data is manipulated via a simple, yet expressive search language—Splunk provides a user interface to explore the data, as well as a simple search language that allows users a range of options from free text searches to heavy duty statistical and analytical searches.

These features make Splunk an ideal candidate for a first approach at semi-automatic data exploration, because it handles efficiently indexing data, extracting fields, performing computation, and visualizing results, allowing us to focus on the problem of how to automatically suggest interesting analyses. In addition, computation in the Splunk search language is structured as a pipeline, which facilitates the iterative construction of analyses.

2.1 The Search Language

The search language is modeled after the Unix `grep` command and pipe operator. Figure 1 shows a schematic of the various elements of a search string. Data, in the form of events with fields, is passed through each command in the search string. [1] Each command filters, transforms or enriches data it receives from the previous stage, and pipes it to the subsequent search stage. Within a search stage, i.e., between pipes, one can extract information from the data, augment the data, or summarize the data using a variety of search commands. At the simplest end of the spectrum, one can perform plain text searches for specific strings or match field-value pairs. At the complicated end of the spectrum, one can perform statistical and analytics operations such as clustering the data using k-means.

Below is an example of a Splunk search to provide a count of errors by detailed status code:

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

The first segment searches all events for the keyword "error", and passes these events to the second stage. The stats command takes these events and computes a count per status code. In the final stage of the search, we append the status description for each status code based on an external lookup into a separate table.

2.2 Automating Data Analysis

In addition to the platform, search language, and visualization tools, there are also pre-built packages called apps, each tailored for certain types of data sets, which include searches that generate reports and dashboards. For example, using a web intelligence app, users can view graphs depicting real-time visitor trends, platform usage, traffic status, and more, and create three dozen types of business, marketing, and IT reports with minimal effort. Other examples include apps for monitoring Windows and Unix logs, PCI compliance, deployment, and enterprise security. Although these apps are useful, their applicability is limited because they are tailored for certain common data sets by domain experts for particular purposes. This makes them unhelpful for exploring new data sets or uncovering interesting visualizations beyond those thought up by an app's author.

Thus, in the context of the Splunk platform, our proposed analysis recommendation tool would enable users to quickly and iteratively create reports and visualizations like those found in apps, but for any type of data set, without needing a semantic understanding of the data. It will do this by performing a bounded, user-guided search through the space of possible data transformations to produce the same types of analyses desired by app users. For example, our tool might recommend a set of top four plots depicting time series or correlations in the data, with options for changing plot type. The user can either click on a one of these plots to see more like it, or can delete one of these plots they deem to be irrelevant, to have it be replaced by a different plot. A similar interface could be provided for other operations, such as joining or grouping. When an operation produces a new data, it can be reincorporated into for iterative analysis.

There has been much prior work on building recommendation systems. Both Google search [1] and Netflix [2] are classic examples of content recommendation based on statistical analysis. There has also been much

[1] You can think of events as being like rows, and fields as being like columns, though this is not strictly accurate. For example, all events need not have the same fields.

Data set	Searches
Ad hoc customer searches	10958
Scheduled customer searches	97727
All customer searches	108685
Firewall app	8
Deployment monitoring app	44
Unix app	86
Web intelligence app	124
Windows app	23
All apps	285

Table 1: Sources and number of search strings were collected. We selected a random subset of customer searches which were issued over the span of a little less than one year. The apps chosen are five of the most popular apps, but we made no particular effort to choose representative ones. Given app's diversity in purpose and complexity, due to targeted raw data set, as well as author, we focus here primarily on customer searches.

research on improving such content recommendation systems [5, 4]. However, what we are proposing here is a recommendation for analysis operations on the data rather than the content of the data.

The main challenges here are exploring the space of possible operations, deciding which operations to present to the user, and incorporating user feedback. One initial idea is to construct the search pipeline one command at a time, using a Markov model learned from past searches. Note that every Splunk search must start with `search`. To add commands, at each step, generate a list of possible next commands. These next commands can be ranked based on a number of things, for example, the probability (based on frequencies observed in real searches) the command will be the ith command given the current string of commands that appear up to place i, or the probability that the command will be used when the data has this or that statistical property. Each time a command is appended to the analysis pipeline, generate a new list with updated rankings. This approach is very loosely inspired by automatic spell checking tools and statistical machine translation. In the remainder of this paper, we examine real searches from customers and apps, and use this study to drive a discussion about this iterative, statistical machine translation-inspired approach.

3 A Study of Searches

To better understand how to develop a semi-automated analysis recommendation tool for data exploration on top of Splunk, we have begun a study of both customer searches as well as app searches. We have collected approximately 100,000 search strings from 81 customers, in addition to 285 search strings from five of the most popular apps. A list of these sources is given in 1. [2]

[2]Note that because customers might have apps installed, some of the logged customer searches will also be app searches.

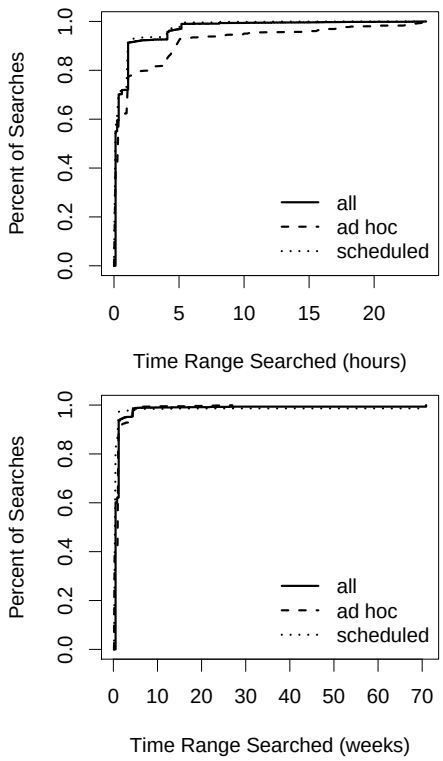

Figure 2: Cumulative frequency distribution of time ranges that were searched over in customer searches, split into two plots for readability. The top graph gives all time ranges under a day, while the bottom graph shows time ranges from one day to a little over a year. Only about 69% of searches specified a time range, but of those that did, 92% were over ranges of less than a day, and many were over ranges on the order of seconds, a fact which we found surprising.

We first present some high-level descriptive statistics about these search strings, then the results of an n-gram analysis. Our motivation for computing these search n-grams was to gain some intuition for how to semi-automatically and iteratively build analysis search strings. This has resulted in a set of ideas for next steps toward our proposed analysis recommendation tool, which we present in Section 4.

We classified customer searches into two types: ad hoc, which are generally one-off searches run by an analyst via the GUI or the CLI, and scheduled, which are run regularly like a `cron` job, usually for the purpose of creating an alert, dashboard, or report. Of the over 100,000 searches we analyzed, 10% were ad hoc.

The majority of customer search strings are short; Figure 3 shows that 80% are length three or less, and 99% less than length ten. Since most searches are scheduled searches, this means that such search strings tend to be very short, usually consisting of a simple `search` for a key term, or a `search` then `stats`. Conversely, only 46% of ad hoc search strings are length three or less. Figure 3(b) is a CDF of app search string length, and Fig-

n	Unique	n-gram	Percent	Command types
1	435	`^search`	68.514	filter
		`^search$`	31.460	filter
		`table$`	13.192	format
		`stats.count$`	11.332	count
		`rex`	10.944	extract
2	1242	`^search stats.count$`	10.593	count
		`^search timechart.count$`	7.690	count, visualize
		`^search top$`	6.938	filter
		`^search rename`	5.746	format
		`^search rex`	4.803	filter, extract
3	1531	`^search rename rex`	3.040	filter, extract, format
		`rename rex table$`	3.039	format, extract
		`^search rename table$`	1.972	filter, format
		`^search rex rename`	1.949	filter, extract, format
		`rex rename table$`	1.949	filter, format
4	1299	`^search rename rex table$`	3.039	filter, format, extract
		`^search rex rename table$`	1.949	filter, format, extract
		`stats.max eval.round collect head`	1.490	max, round, store, filter
		`eval.round collect head export$`	1.490	round, store, filter
		`fields eval./ stats.max eval.round`	1.490	filter, divide, max, roumd
5	1066	`fields eval./ stats.max eval.round collect`	1.490	filter, divide, max, round, store
		`^search eval fields eval./ stats.max`	1.490	filter, divide, max
		`eval fields eval./ stats.max eval.round`	1.490	filter, divide, max, round
		`stats.max eval.round collect head export$`	1.490	max, round, store, filter
		`eval./ stats.max eval.round collect head`	1.490	divide, max, round, store, filter

Table 2: Summary of top search string n-grams for all subset of support case searches. The number of total unique n-grams can decrease because searches that are less than length n are not included for n-gram analysis. The majority of customer search strings are short and simply filter and format log events, occasionally computing some simple statistics.

ure 3(c) compares customer searches with app searches. In apps, search strings tend to be longer; with 48% of length three or less and 91% less than length ten. This varies greatly by app, however, with certain apps, such as the deployment monitor, skewing toward very long searches, and other apps, like the Unix app, skewing short.

Information about the time ranges that customer searches were run over are presented in 2. There are a few caveats to note. The first is that not all searches specify a time range—some are just specified to run over all events in an index. Thus, only searches that specify a time range, approximately 69% of customer searches, are included. The second is that, for readability, searches were split into ranges of less than a day, and ranges over a day (the longest range is a little over a year). Searches in the first category make up about 92% of searches. Many searches span very small time ranges; 450 searches are over a range of a minute or less, and 31 searches were over a range of one second. Many of these searches turn out to be the same search string repeated many times. Many searches examine less than two hours of data, and searches looking a more than a day are much less common.

Table 2 depicts some results of computing all n-grams for each logged customer search string. To compute these, we reduce each search string to a list of commands and their function arguments (e.g., `stats` is a command and `count` is a function argument), preserving the order that they appear in the original search string. We then compute n-grams from this list, where an n-gram is any sublist of elements of size n. We show only the top five most frequently occurring n-grams for $n \in \{1, \ldots, 5\}$, in the interests of space, and because most search strings are short. A quick note on notation:

- `^cmd` means *cmd* is the first command in the search.
- *cmd*`$` means *cmd* is the last command in the search.
- *cmd*`.`*fn_1*`.`*fn_2* means *cmd* is called with *fn_1* and *fn_2* as arguments.
- *cmd*`.`*fn_1*`(`*fn_2*`)` means *cmd* is called with *fn_1* as an argument, which has *fn_2* as an argument.
- `[`*cmd*`]` means that *cmd* is contained in a subsearch.

Along with each n-gram, we include a short description what type of commands are included, e.g., the `search` command is a filters events, while the `fields` command filters fields, but both are filters.

We see that the majority of customer searches, as previously observed, are simply filtering and formatting log events, and occasionally computing some simple statistics. This suggests that building analysis search strings based solely on n-gram frequencies computed from customer search logs, as proposed in Section 2.2, will yield relatively simple searches; the next section show some examples of automatically generated searches, along with additional heuristics used to guide the exploration towards more "interesting" queries. An example of search strings that might be suggested by using approximately this algorithm is given in Section 4, along with a discussion of

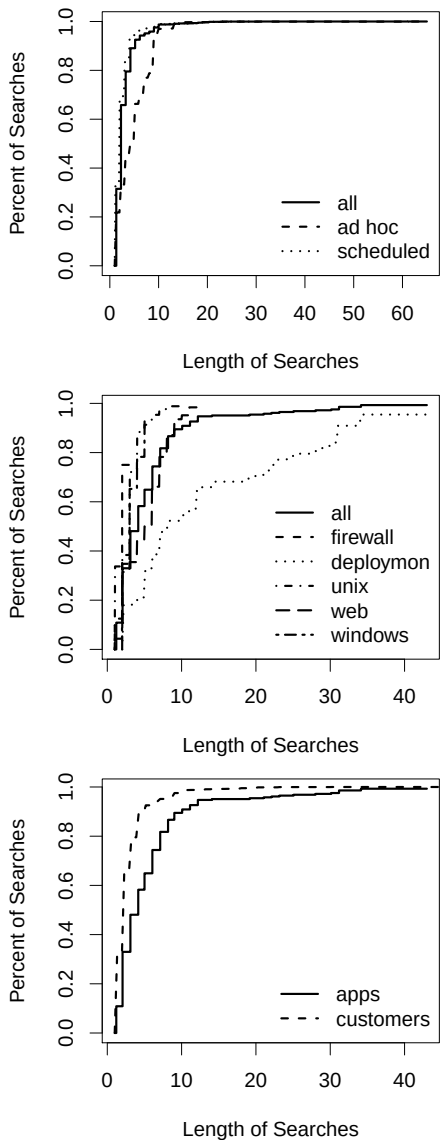

Figure 3: Distributions of search string lengths. The top graph shows the search search string lengths for customer searches, the middle for apps, and the bottom graph compares customer searches and apps. The majority of search strings are short, though app search strings tend to be longer.

possible improvements.

Another observation is that some common command strings are simply permutations of one another, e.g., `rename rex` and `rex rename`. This suggests that for our analysis recommendation tool, it might be useful to define a canonical search language, one of the purposed of which would be to represent permutations of commands that do the same thing in a single way. This would also help eliminate other idiosyncrasies such as the fact that `head export` is appended to commands that are run from the command line interface.

4 Toward Analysis Recommendation

Table 3 illustrates the rough idea for automatically generating the recommended searches as mentioned in Section 2.2. In this algorithm, the construction of search strings is like a Markov chain. This example uses the customer search string data provided in the previous section for the probabilities that weight the choice of the next command in the string. In this example, we start with the most frequently occurring starting command, which is always `search`. Although not shown here, we also need a way to pick the arguments to `search`. One possibility is to search for all values of categorical fields, or for commonly occurring search strings. We do not want to search for very rare search strings ($n = 1$) since that represents a different use case; we can assume that users will not be interested in analysis of a single data point. Next, we look at the list of commands that most frequently occur after `search`, and so on. When we reach the end of a search string, we output that as a suggested search.

Note that in the example we do not strictly explore the path dictated by ranked frequency (iteration 3 should actually be `^search rename rex`). This is because that path terminates relatively quickly, and we wanted to give an example of a longer search. This demonstrates one of the weaknesses of this particular next-command ranking function. In addition, this ranking function does not account for equivalent permutations of commands. In the remainder of this section, we discuss several challenges we must solve, relating to these issues.

Choosing time ranges and arguments to commands: In order to protect customer privacy, we did not look at the fields or values in the customer search strings, since that is sensitive information. However, when suggesting an analysis, we must also decide what fields or values to include as arguments to each command. In addition, we must choose a time range of data to analyze. Potentially, similar approaches for ranking which commands to include next in an analysis pipeline could apply for ranking which fields or values to include as arguments to each command. Or, especially in the case of time ranges, it might be reasonable to ask the user to specify.

Choosing an appropriate ranking functions: The ranking scheme we proposed was purely based on customer search string n-gram frequencies. A statistical approach is reasonable; however, in addition to n-gram frequencies, we may want to incorporate other information, including, but not limited to:

1. The type of data source: e.g., Splunk searches often specify the index or source type being searched over (as index and source type are fields in each data set), thus, when computing next-command rankings, we can consider only search strings that search over the same values for those fields.

Iteration	Suggestion to Output	Current Command	Next Command
1	^search$	^search	^search stats.count$ ^search timechart.count$ ^search top$ ^search rename
2	^search stats.count$ ^search timechart.count$ ^search top$	^search rename	^search rename rex ^search rename table$ ^search rename join
3	^search rename table$	^search rename join	^search rename join table$ ^search rename join [search...
...	...	...	...
i	^search rename join [search rename] rename convert table$		

Table 3: An example algorithm for automatically generating recommended searches based on customer search n-gram frequencies. We start with the most frequently occurring starting command, which is always `search`. Although not shown here, we also need a way to pick the arguments to `search`. When we reach the end of a search string, we output that as a suggested search.

2. Search strings from apps: similar to the previous case, we can leverage, where available, the domain-knowledge present in already-created apps when the data being searched is of the same type the app is intended to be applied to, by giving higher weight to n-grams computed from those apps.
3. Statistics related to the data itself: Certain commands are more interesting or more often applied when the data set in question has certain simple statistical properties. A simple example of this is that when a field always has the same value, a time series plot of that field is unlikely to be interesting.

Exactly how to include this information is an open question. Moreover, we may want to use a hybrid rule-based and statistical approach in order to generate more sophisticated analysis. For instance, commands such as `anomalies` and `kmeans` are infrequently used, and not always applicable, but have the potential to yield interesting insights. Thus, we could have a rule that looks for cases when these commands make sense to apply (e.g., must have appropriate data type and entropy) and include those as suggestions.

Incorporating user feedback: When a user rejects a suggestion or requests similar suggestions, this should affect which suggestions are provided subsequently, but it exactly how this should be done is a matter of future research.

Continuous, online training: We saw that the customer search strings with which we might initially generate our rankings tend to be quite simple. A simple `search` is a good start but by itself does not make for very sophisticated analysis. In addition, such search strings likely arise from a different use case than our tool is targeted for, namely, trouble-shooting, rather than offline, open-ended data exploration. Since the performance of statistical approaches depend heavily on how good the input data is, this is undesirable. The more a semi-automated data exploration tool is used, the more examples of interesting search strings we will have. Our tool must have some way of incorporating this information.

Creating a canonical analysis language: We have already seen simple examples where some sequence of operations in the Splunk search language are equivalent to others, or where the search language has certain quirks not particularly relevant to the actual analysis being done. The performance of our tool might be improved if it only needs to consider search strings which have been canonicalized into a form of the analysis language which transforms equivalent commands into a single form.

Applicability to other platforms: The solutions to the above challenges will ideally be based on principles general enough to apply to many other data analysis platforms. It should be conceptually straightforward to apply our solution to other systems which also use pipelined languages. In other cases, such as with SQL-like systems, it might be simplest to include a translation step which translates between SQL and our canonical analysis language.

References

[1] Google Social Search, A Recommendation Engine. http://googlesystem.blogspot.com/2011/02/google-social-search-recommendation.html.

[2] Netflix. http://www.netflix.com.

[3] L. Bitincka, A. Ganapathi, S. Sorkin, and S. Zhang. Optimizing data analysis with a semi-structured time series database. In *SLAML*, 2010.

[4] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Recommendation systems: A probabilistic analysis. *Journal of Computer Systems Science*, 63(1), 2001.

[5] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The Adaptive Web*, pages 325–341, 2007.