REACT: Context-Sensitive Recommendations for Data Analysis

Tova Milo
Tel Aviv University, Tel Aviv, Israel
milo@post.tau.ac.il

Amit Somech
Tel Aviv University, Tel Aviv, Israel
amitsome@mail.tau.ac.il

ABSTRACT

Data analysis may be a difficult task, especially for non-expert users, as it requires deep understanding of the investigated domain and the particular context. In this demo we present REACT, a system that hooks to the analysis UI and provides the users with personalized recommendations of analysis actions. By matching the current user session to previous sessions of analysts working with the same or other data sets, REACT is able to identify the potentially best next analysis actions in the given user context. Unlike previous work that mainly focused on individual components of the analysis work, REACT provides a holistic approach that captures a wider range of analysis action types by utilizing novel notions of similarity in terms of the individual actions, the analyzed data and the entire analysis workflow.

We demonstrate the functionality of REACT, as well as its effectiveness through a digital forensics scenario where users are challenged to detect cyber attacks in real life data achieved from honeypot servers.

1. INTRODUCTION

Exploratory data analysis has evolved significantly since the emergence of the "Big Data" era. Data analysis software platforms (such as Tableau, Splunk and IBM InfoSphere) are gradually replacing traditional tools, allowing easy-to-use data exploration, visualization and mining, in big data environments, even for users lacking knowledge of SQL and programming languages. Yet, data analysis can be a difficult process, especially for non-expert users, as it requires deep understanding of the investigated domain and the particular context. Users therefore may skip significant analysis actions or overlook important aspects of the data [7].

In this demo we present REACT, a system that assists data analysts by providing personalized recommendations of relevant data analysis actions. REACT exploits previous experience of other analysts working with the same or other data sets that it records. Given the specific context of the user (the analysis actions performed thus far, the results obtained, the properties of the data set at hand, etc.) it uses the recorded information to suggest the next best analysis action to the user in the current context of their session.

Time	Source	Destination	Protocol	Length
0.000000000	172.16.254.128	216.58.208.206	SSL	55
0.000363000	216.58.208.206	172.16.254.128	TCP	60
0.289945000	172.16.254.128	216.58.208.226	TCP	55
0.293835000	216.58.208.226	172.16.254.128	TCP	60
3.464487000	172.16.254.128	8.8.8.8	DNS	73
3.482495000	8.8.8.8	172.16.254.128	DNS	89
3.483014000	172.16.254.128	216.58.208.227	TCP	66
3.483412000	172.16.254.128	216.58.208.227	TCP	66
3.495475000	216.58.208.227	172.16.254.128	TCP	60

Figure 1: Parsed Network Capture File

Generating adequate data analysis recommendations is a theoretical and practical challenge. Ideally, the recommended action should be given with respect to the users' level of expertise, methodology and their thinking process. It should be comprehensible at the current point in the analysis session, and most importantly, should result in a new view of the information, pointing out interesting aspects of the data objects in a way that the users did not see before. Furthermore, it should be presented to the user in reasonable time, i.e., before she decides herself on the next action.

Previous work suggests the use of recommendations in the domain of exploratory data analysis but mostly focuses on one specific aspect of the process. Some works (e.g. [3], [6]) focus on *data retrieval*, providing SQL query recommendations with auto-completion features; other works (e.g.[5]) focus on *data presentation* via OLAP operations, presenting a recommender system for data-cube queries; systems for *data visualization* recommendations were presented e.g. in [10]. While these works all make a notable contribution, user studies show that data analysis sessions interleave actions of all types with dependency of one action on the results of previous actions of the same or different type [1]. Thus, as advocated in the white paper of [7], a more holistic approach is required to fully capture the essence of the analysis work. It should also be noted that most of the previous work assumes an environment with a single data source queried by multiple users. Current big data settings support a large number of data sources and there is an important need to transfer experience gained with one data source to others.

REACT, to our knowledge, is the first one to provide a holistic approach that exploits user experience (in terms of both the individual analysis actions, their workflow, and the analyzed data), on similar-but-not-necessarily-identical data sources, for generating context sensitive recommendations.

In a nutshell, given a repository of recorded analysis sessions and the state of a user within its current session, REACT

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2882903.2899392

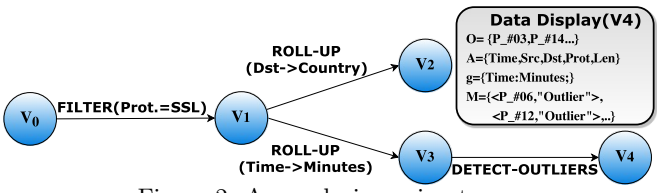

Figure 2: An analysis session tree.

operates as follows: First, it identifies previously performed similar sub-sessions, based on notions of *data-similarity* and *analysis actions-similarity* that we define. From these we retrieve a set of candidate next actions. Since the actions may be diverse and operate on distinct data sets, rather than looking at the individual frequent actions, our next step is to identify common generalized *action templates*. Finally, by considering the properties of the current data set, we instantiate the templates with actual parameters to obtain *concrete action recommendations* that have the potential to highlight new and interesting point-of-views of the given data source.

Our solution makes three primary contributions: (i) A generic data model that captures the nature of exploratory data analysis, including: data retrieval, cube operations and data mining, and can be expanded for visualization as well. Therefore `REACT` is easily adjustable to state-of-the-art data analysis platforms. (ii) Our system engine can identify similar analysis session patterns, made over different data sources. This novel approach significantly increases the pool of relevant actions for the recommendation process. (iii) Our system groups and then generalizes individual analysis actions performed by users, in order to find better recommendation candidates. This makes `REACT` a good fit for practical scenarios where there are many data sources and the chances for multiple users to perform the exact same actions are slim.

To demonstrate the functionality of `REACT`, our scenario of choice is digital forensics in the context of cyber security threat detection. Participants will be invited to take part in numerous forensics challenges from the HoneyNet project[1], for identifying malware communication, APT (advanced persistent threats), or exploits in the provided data. The participants will be asked to solve analysis challenges of increasing level of difficulty and we will compare the time it takes to complete them with and without the assistance of `REACT`. We will also provide a "behind the scenes" view of the system, demonstrating the operation of the underlying algorithms and their key components.

The remainder of this paper is structured as follows: Section 2 provides an overview of the data model and some theoretical background. A description of the system architecture and flow is presented in Section 3. Section 4 describes the demonstration scenario in more detail.

2. TECHNICAL BACKGROUND

We briefly review the technical background underlying `REACT`. We start with the data model, then overview the main algorithms for computing analysis recommendations.

Data source. A typical data analysis session starts when a data analyst inquires into a matter (e.g. *Detect a backdoor communication session within a network capture file*, as illustrated in Figure 1). Initially, she parses the traffic capture to a tabular format and loads the data to an analysis UI. Then, she executes a series of analysis tasks, to fulfill her assignment.

We abstractly model a data source by a triplet $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \mathcal{H})$, where $\mathcal{O}$ is a set of data objects, $\mathcal{A}$ is a domain of attribute names and $\mathcal{H}$ is a set of semantic hierarchies, one per attribute name, that defines an abstraction refinement order on the attribute values. For example, semantic hierarchies that can be employed for the *time* and *IP* attributes of the data source in Figure 1 are: $hours \leq minutes \leq seconds$ and $country \leq city \leq IP$.

Data Display, Analysis actions, Analysis tree. Inspired by [1], we assume that an analyst performs actions of three main categories: ***data retrieval*** actions, performed to select and filter the relevant data objects for the current assignment (e.g. `FILTER` BY "protocol=*HTTP*"); ***data representation*** operations are performed to alter the point-of-view of the data objects and include OLAP cube exploration, sort, highlight, and alike (e.g., `ROLL-UP` 'session time' FROM *'seconds'* TO *'days'*) and ***data mining*** tasks such as clustering and outlier detection (e.g., *`CLUSTER` BY 'source IP address'*).

At each point of the analysis we are given some *Data Display* $D = (O, A, g, M)$, where $O \subseteq \mathcal{O}$ is the current subset of data objects and $A \subseteq \mathcal{A}$ are the attributes being considered. g denotes the current group-by preferences for the objects, identifying one abstraction level per attribute (and possibly a corresponding aggregate function) and represents the cube point-of-view of the current display (e.g. in the current example g can be {$\langle$*Time*:Minuetes;IP:country $\rangle$, 'avg_length':`AVG`(*length*)}). Finally, M associates a (possibly empty) set of labels with each object, originating from data mining operations (Refer to Figure 2 for an example).

We assume a domain $\mathcal{T}$ of analysis actions that an analyst can perform on the data display through a given UI. Examples are `FILTER BY`, `ROLL-UP`, `CLUSTER BY`, etc. Each action $t \in \mathcal{T}$, takes some parameters (e.g. the filtering selection criteria, roll up attributes, clustering function, etc.) and, given as input some data display, generates a corresponding new display. We use $t(p)$ to denote the action $t \in \mathcal{T}$ with its concrete parameters p. We may *generalize* a concrete action $t(p)$ into an *action template* $t'(p')$ by replacing the action name or some of the parameter tokens in p by new variable names. For example, the two concrete filtering actions `FILTER`($length \geq 48$) and `FILTER`($length = 5$) may both be generalized to the action template `FILTER`($length$ $\mathtt{x}_1$ $\mathtt{x}_2$) where $\mathtt{x}_1$ generalizes '='; and $\geq$ and $\mathtt{x}_2$ generalizes '48' and '5'.

Generalization defines a partial order among templates. We write $t'(p') \leq t(p)$ if $t'(p')$ equals $t(p)$ or generalizes it. Given two actions $t_1(p_1)$, $t_2(p_2)$, we say that a template $t'(p')$ is their most specific generalization if it generalizes the two and there is no other template $t''(p'')$ s.t. $t'(p') \leq t''(p'')$. W.l.o.g. we will assume below that all actions have the same number of parameters/tokens. All results generalize to the general case via padding.

In our analysis we will be interested in action templates that occur frequently. Given a set T of concrete actions and some threshold Θ, we say that an action template $t'(p')$ is *frequent* in T if $\frac{|\{t(p)|t(p)\in T, t'(p')\leq t(p)\}|}{|T|} > \Theta$. In particular, we will be interested in the most specific such templates. We say that $t'(p')$ is *maximal-frequent* if it is frequent and there is no other frequent template $t''(p'')$ s.t. $t'(p') \leq t''(p'')$.

Analysis sessions work intuitively like website navigation sessions - at each point of the session one may backtrack

[1]http://honeynet.org

to a previous data display and take an alternative navigation path. We thus model an analysis session as an ordered labeled tree whose nodes are the data displays. The edges outgoing each node are labeled by the performed action and lead to the resulting data display node. The order captures the execution time-line. See Figure 2 for an illustration of an analysis session tree.

2.1 Generating recommendations

Given a repository of previously performed sessions (modeled as trees, as described above), and the current user session (modeled as well as a tree) we first search the repository to identify the top-k similar subtrees. Then we use them to generate next-action recommendations.

Identifying similar subtrees. Our notion of subtree similarity is based on *edit distance*: An *edit script* is constructed from edit operations that can be applied on nodes and on edges: *delete/add* a node or an edge, and *alter* the label of a node or an edge. Each operation has a cost, and the edit distance is the cumulative cost of edits required to transform one tree into another. Add/delete operations have a unit cost, whereas the cost of an alter operation for node/edge labels reflect the similarity between the data displays and analysis actions that they represent, respectively. Next, We intuitively define these similarity notions.

Data display similarity. Recall that different analysts may operate on distinct data sources (as well as on distinct sets of objects in the same source), yet we would like to benefit from the users' experience across data sources and sets. Given two data displays, we first employ schema matching [8] to match (when possible) between the attribute of the two displays. We then compare the content of the matched attributes. Since the objects being examined may be very different, and thus also the attribute domains, rather than comparing the actual value distributions we characterize their structure via the standard notion of entropy and compare the resulting entropy values. Finally, given two data displays, their overall similarity score (whose exact formula is omitted for space constraints) incorporates (1) the schema similarity matching score, (2) the attributes entropy similarity, (3) the similarity in the group-by preferences (defined by their distance in the abstraction hierarchy) and (4) the similarity in mining labels (defined via Dice coefficient, which is a standard similarity measure for multisets).

Actions similarity. We measure the similarity between two concrete analysis actions $t_1(p_1)$ and $t_2(p_2)$ by the sum of their distances from their common most specific generalization[2]. For instance, in our example above, the distance between `FILTER`($length \geq 48$) and `FILTER`($length = 5$) is 4, given that `FILTER`($length$ $\mathtt{x}_1$ $\mathtt{x}_2$) is their most specific common generalization.

Optimized search algorithm. To find the top-k similar trees we employ a refinement of the subtree similarity search algorithm in [2] with the edit distance computation method suggested in [11]. First, as [11] supports edit operations on nodes only, we replace each edge in our trees by an edge-node-edge triplet, assigning the edge label to the corresponding node. Second, since the algorithm in [2] is linear in the data size, and the number of recorded analysis sessions may be large, to allow for instant response time we preprocess the data, clustering similar analysis trees together. We compare the user tree only to the center of each cluster, and further explore cluster members (within a bounded radius from the center) only for the most similar ones.

[2]normalized by the maximal distance between a concrete action and a template where all tokens are variable

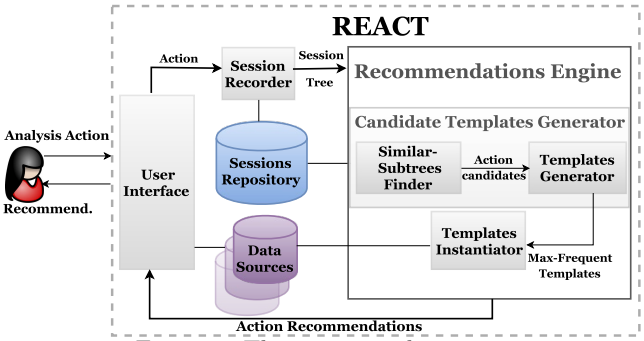

Figure 3: The system architecture.

From a storage (and computation) perspective it is important to note that the data displays need not be fully recorded to enable this computation. Indeed, for each node we only record the set of attribute names along with the entropy value for each attribute, and the attribute names (resp. mining labels) in g and M. Schema matching may also be precomputed for all data sources.

Generating recommendations. Once the set of similar trees is identified, we use them to decide on the best next action. This is done in three steps.

First we consider the nodes (in the retrieved set of similar subtrees) that correspond to the current user node. We take the edges (actions) outgoing these nodes and form a multiset consisting of candidate actions. Since the actions may be diverse and operate on distinct data sets, our next step is to identify common generalized action templates. Formally, the *maximal-frequent* action templates as defined above.

Finally, by considering the properties of the current data set, we instantiate the templates with actual parameters to obtain *concrete action recommendations* that are relevant to the given user and have potential to highlight new and interesting point-of-views of the data source. For that we employ notions presented in [4] to formulate an interestingness measure. Given an action template, the domain of instantiations that we consider for its variables, consist of the common instantiations in the actions that it generalizes (mapped to the current schema). For each such full instantiation, we calculate its interestingness by executing the action (using the measure mentioned above), and select the top-k most interesting such instantiations.

3. SYSTEM OVERVIEW

We implemented `REACT` in Python 2.7, as an intermediate plug-in between an analysis UI (Bootstrap 3) and the data source layer. The main system components and workflow are illustrated in Figure 3). A user analyzes data via a UI (see Figure 4) to which `REACT` is plugged in. Her analysis actions are passed from the UI to the *Session Recorder* which incrementally constructs the current session's tree representation and updates the *Sessions Repository* correspondingly. It then passes the session tree as input to the *Recommendations Engine* which works as follows. Given the current session tree, the *Candidate Templates Generator* calls the *Similar-Subtrees Finder* that searches the *Sessions Repository* for similar sub-sessions. It extracts from them a list of *Action Candidates* and passes them to the *Templates Gen-*

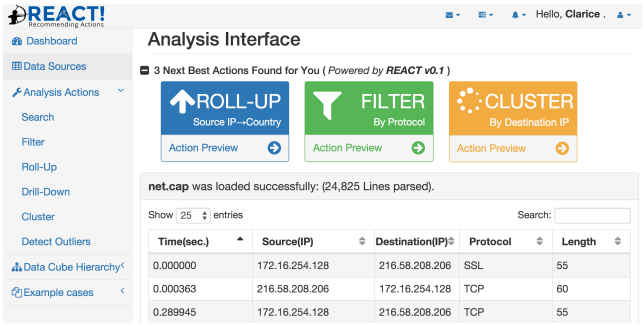

Figure 4: Analysis UI powered by REACT recommendations

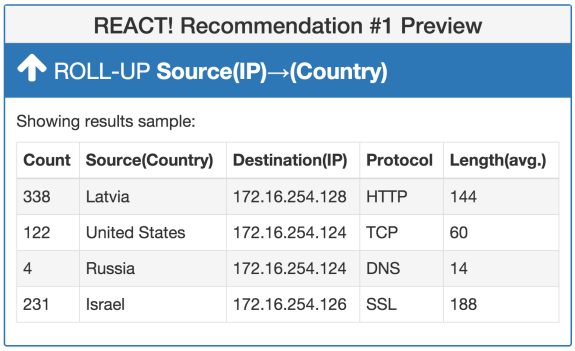

Figure 5: Action preview (Corresponding to Figure 4)

erator. The latter generalizes the actions to obtain a set of *maximal-frequent templates*. The templates are then passed to the *Templates Instantiator* that instantiates them into a set of explicit "most relevant" actions. To identify these actions it executes the candidate instantiations over the data source and measures the potential *interestingness* of the results. Finally, the selected *Action Recommendations* are returned to the user via the UI (Figure 4) where she can preview the recommendation results set (Figure 5).

4. DEMONSTRATION

We demonstrate the functionality of `REACT`, as well as its effectiveness, in the context of digital forensics in cyber security. Note that when solving cases of network breeches and malware attacks, forensics researchers often have to dive into a vast amount of machine generated data coming from multiple sources (e.g. network traffic files, system logs, memory dumps, etc.). This is a particularly suitable setting for demonstrating the capabilities of `REACT` as it includes all the challenges that the system addresses: The data sources vary from one investigation to another, and furthermore, even when working on the same data, users have different goals, as well as a distinct expertise and methodology.

We will begin with an overview of the system and its user interface, then invite the participants to take part in solving several forensics challenges from the HoneyNet project, published in the past few years. Each challenge has a distinct goal and data set, e.g., identifying malware communication in network traffic capture files, tracing APT (advanced persistent threat) activity in *syslog* messages and spotting exploits in memory dump files. The participants will be given analysis tasks in an increasing difficulty level and will be asked to perform the analysis actions via a dedicated UI.

To demonstrate the effectiveness of `REACT`, participants will be split into two groups, one working with the full fledged system, and another working with a degenerated version of the system that does not provide recommendations. We will record the time to completion of both user groups and present the average difference, demonstrating the superior success rates and speedup that `REACT` achieves.

Users working with the full system will be guided by `REACT` in performing the analysis actions required to complete the task. Within each analysis session, `REACT` will generate personalized recommendations using a sessions repository consisting of previous sessions recording tasks performed by users over the available data sources. We will complement the demonstration by providing a "behind the scenes" view of the system. Following the analysis work of participants, we will show how the system finds similar sessions, then generalizes the identified candidate actions into action templates and finally how it instantiates the templates to obtain a personalized recommendation to fit the user's context.

Related Work. As mentioned previously, there has been much work on recommender systems [9], and explicitly for data analysis [3] but they focus mostly on one specific aspect of the process. Our holistic approach incorporates these facets into a comprehensive solution. Our architecture is modular and can employ complimentary measures of similarity and interestingness depicted e.g. in [4]. Analysis recommendation systems often assume that users are working on the same data source/set of objects with a sufficient number of sessions that one can learn from [6]. However, this may not be the case in current distributed settings and our work allows to leverage sessions over multiple sources.

Acknowledgments. We thank Daniel Deutch and Amir Gilad for their insightful comments. This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071 and by a grant from the Blavatnik Interdisciplinary Cyber Research Center.

5. REFERENCES

[1] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *INFOVIS*. IEEE, 2005.

[2] N. Augsten, D. Barbosa, M. Böhlen, and T. Palpanas. Tasm: Top-k approximate subtree matching. In *ICDE*, 2010.

[3] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, 2009.

[4] L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *CSUR*, 38(3):9, 2006.

[5] A. Giacometti, P. Marcel, and E. Negre. *Recommending multidimensional queries*. Springer Berlin Heidelberg, 2009.

[6] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for sql. *PVLDB*, 2010.

[7] J. Liu, A. Wilson, and D. Gunning. Workflow-based human-in-the-loop data analytics. In *HCBDR*, 2014.

[8] T. Sagi and A. Gal. Schema matching prediction with applications to data source discovery and dynamic ensembling. *PVLDB*, 2013.

[9] M. Sarwat, J. Avery, and M. F. Mokbel. Recdb in action: recommendation made easy in relational databases. *PVLDB*, 2013.

[10] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 2015.

[11] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM*, 1989.