Query Prediction with Context Models for Populating Personal Linked Data Caches

Olaf Hartig
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
hartig@informatik.hu-berlin.de

Tom Heath
Talis Education Ltd.
43 Temple Row, Birmingham, B2 5LS, UK
tom.heath@talis.com

ABSTRACT

The emergence of a Web of Linked Data [2] enables new forms of application that require expressive query access, for which mature, Web-scale information retrieval techniques may not be suited. Rather than attempting to deliver expressive query capabilities at Web-scale, we propose the use of smaller, pre-populated data caches whose contents are personalized to the needs of an individual user. Such caches can act as personal data stores supporting a range of different applications. In this paper we formally introduce a strategy for predicting queries that can then be used to inform an *a priori* population of a personal cache of Linked Data harvested from Web. Based on a comprehensive user evaluation we demonstrate that our approach can accurately predict queries and their execution probability, thereby optimizing the cache population process.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous;
K.8.m [**Personal Computing**]: Miscellaneous

Keywords

Query Prediction, Context, Cache Population, Linked Data

1. INTRODUCTION

In earlier work we introduce an approach to *pro-actively* populate a personalized Linked Data cache for a particular, forthcoming usage context [1]. Such caches present an alternative to maintaining a centralized infrastructure that aims to serve the needs of all users in all situations. By accessing a personalized cache in the context for which it was populated, software applications may efficiently support a particular user in satisfying context-specific information needs. An important step of our cache population process is the prediction of SPARQL queries [4] that applications may execute over the cache in a given, future context. The predicted queries may then be used as a basis for the actual cache population strategy [1].

Each of the predicted queries is associated with an *execution probability* which represents the likelihood that the query is actually executed in the given context. Formally, we define predicted queries as follows:

Definition 1. A **predicted query** is a pair (q, ep) where: q is a SPARQL query; $ep \in [0, 1]$ is an estimated execution probability.

Note that the *actual* execution probability of predicted queries is unknown during the process of pre-populating a cache. Accordingly, the value ep associated with a predicted query (q, ep) represents an assumed (i.e. predicted) probability, computed as part of the query prediction itself. Since execution probabilities indicate the relevancy of queries for our cache population approach we require a prediction strategy that forecasts the execution probability for predicted queries as accurately as possible.

The main contributions of this paper are the precise definition of our query prediction strategy and a user based evaluation thereof.

2. PRELIMINARIES

The basis of our approach is a generic context model which represents a context by a set C of *context attribute*s (CA). Each CA is a tuple (t, v, r) where t denotes the type of the CA, v is an RDF term [3] which denotes the value of the CA, and $r \in [0, 1]$ is a relevancy score which denotes the degree to which the CA is relevant in the corresponding context. We deliberately refrain from defining a particular set of CA types for our model; a multitude of works exist that aim to capture the notion of context by defining relevant concepts (e.g. location, user interest) and their relationships. Depending on the application domain, any collection of such concepts may be suitable (and can be selected) as possible types for CAs.

In addition to a (future) context, our query prediction strategy depends on a specification of user tasks: People aim to meet their information needs by performing certain tasks. Software applications may support the performance of some of these tasks, in which case we call such a task a *supported task*. For example, an application may enable a user to perform the task of finding transport options to nearby places of interest.

The actual performance of supported tasks is context-specific. To capture this dependency formally we model a supported task as a pair (P, cat) where P is a set of symbols that denote context dependent properties (CDP) of the task and cat is a total mapping from P to the set of all CA types in the application domain. In any possible, context-specific performance of such a task, each CDP of the task will always be instantiated by CAs of a certain type. Mapping cat specifies this type. Formally, we define the performance of a supported task in a particular context as follows: Let $st = (P, cat)$ be a supported task and let C be a set of CAs. A *C-specific performance of st* is a total, injective mapping $perf : P \rightarrow C$ such that $\forall\, cdp \in P : \big(perf(cdp) = (t, v, r) \Rightarrow cat(cdp) = t \big)$.

When a person interacts with an application in order to perform a supported task, some of the user actions will cause the application to issue queries that will be evaluated over the user's personal cache. We assume that applications generate such queries by instantiating prepared templates for queries. To denote the set of *query template*s an application may instantiate when used to perform a supported task $st = (P, cat)$ we write $QT(st)$. Furthermore, each such query template $qt \in QT(st)$ is modeled as a tuple

HT'12, June 25–28, 2012, Milwaukee, Wisconsin, USA.
ACM 978-1-4503-1335-3/12/06.

$qt = (q, ip, sub)$ consisting of a SPARQL query q, an instantiation probability $ip \in [0,1]$, and a partial, injective mapping $sub : \mathrm{vars}_{\mathsf{where}}(q) \rightarrow P$ where $\mathrm{vars}_{\mathsf{where}}(q)$ denotes the set of query variables that occur only in the WHERE clause of q. The variables in the domain of sub act as placeholders. If an application instantiates a query template during a context-specific performance of a supported task, then it replaces each placeholder of the template by the value of the CA that instantiates the corresponding CDP. Formally, we define such an instantiation as follows: Let $st = (P, cat)$ be a supported task; let $qt = (q, ip, sub) \in QT(st)$ be a query template of st; and let $\mathrm{dom}(sub)$ denote the domain of mapping sub. Furthermore, let $perf$ be a C-specific performance of st. The *$perf$-based instantiation of qt*, denoted by $\mathrm{inst}(qt, perf)$, is a SPARQL query that we obtain from q when for each query variable $?v \in \mathrm{dom}(sub)$ we replace all occurrences of $?v$ in q by v where v is the value in CA $perf\big(sub(?v)\big) = (t, v, r)$.

3. QUERY PREDICTION

The general idea of our query prediction strategy is to generate all possible instantiations of those query templates that applications may instantiate in a future context given by a set C_{fut} of CAs[1]. To allow for an automated implementation of this idea, we assume a complete specification of all tasks supported by all applications that may access the personal cache; in particular, the specified list of query templates for these supported tasks is assumed to be complete. Hence, an additional input to our approach (in addition to C_{fut}) is a (full) set of supported tasks ST_{input} which is given as part of a usage scenario specification [1].

To predict any query that is possible w.r.t. C_{fut} and ST_{input}, we consider all query templates of all supported tasks in ST_{input}. For each of these templates we generate all instantiations that are possible in the context represented by C_{fut}. Formally, the complete set of queries that we may generate is given as follows:

$$\big\{\mathrm{inst}(qt, perf) \;\big|\; st \in ST_{\mathsf{input}} \text{ and } qt \in QT(st) \text{ and } perf \text{ is a } C_{\mathsf{fut}}\text{-specific performance of } st\big\}$$

For all queries that we generate we have to estimate an execution probability. Our estimation is based on the following assumption: The likelihood that a query $q = \mathrm{inst}(qt, perf)$ generated for a task $st \in ST_{\mathsf{input}}$ is executed in the context represented by C_{fut} depends on i) the general instantiation probability ip of $qt \in QT(st)$ and ii) the likelihood of $perf$. The first of these parameters is given as part of ST_{input}. For computing the second parameter we introduce the notion of a *performance estimation method*, that is, a function that, for any finite set C of CAs and any supported task st, maps any possible C-specific performance $perf$ of st to a value $\xi(perf) \in [0,1]$ such that $\xi(perf)$ is an estimate for the likelihood that st, in the context represented by C, is actually performed as specified by $perf$. Examples for such functions are ξ_{min} and ξ_{avg}:

$$\xi_{\mathsf{min}}(perf) = \min\big(\Phi(perf)\big) \;;\; \xi_{\mathsf{avg}}(perf) = \frac{1}{|\Phi(perf)|} \sum_{r \in \Phi(perf)} r$$

where

$$\Phi(perf) = \{r \mid cdp \in \mathrm{dom}(perf) \text{ and } perf(cdp) = (t, v, r)\}$$

Given the concept of performance estimation methods we now define the construction of a predicted query:

Definition 2. Let C be a set of CAs; let $st = (P, cat)$ be a supported task; let $qt = (q, ip, sub) \in QT(st)$ be a query template; let $perf$ be a C-specific performance of st; and let ξ be a performance estimation method. The **ξ-based query prediction result for qt and $perf$**, denoted by $\mathrm{qp}^{\xi}(qt, perf)$, is a predicted query:

$$\mathrm{qp}^{\xi}(qt, perf) = \Big(\mathrm{inst}(qt, perf)\,, \mathrm{p}^{w_{\mathsf{ip}}, w_{\mathsf{perf}}}\big(ip, \xi(perf)\big)\Big)$$

where $w_{\mathsf{ip}}, w_{\mathsf{perf}} \in \mathbb{N}^{+}$ are weights and

$$\mathrm{p}^{w_{\mathsf{ip}}, w_{\mathsf{perf}}}\big(ip, \xi(perf)\big) = \big(w_{\mathsf{ip}} \cdot ip\big) \cdot \big(w_{\mathsf{perf}} \cdot \xi(perf)\big)$$

Given Definition 2, the complete set of predicted queries that can be computed with our query prediction strategy (for C_{fut}, ST_{input}, and a performance estimation method ξ) is the following:

$$\big\{qp^{\xi}(qt, perf) \;\big|\; st \in ST_{\mathsf{input}} \text{ and } qt \in QT(st) \text{ and } perf \text{ is a } C_{\mathsf{fut}}\text{-specific performance of } st\big\}$$

4. EVALUATION

We evaluated the accuracy of our prediction of execution probabilities by assessing their degree of correlation with participants' ratings of the likelihood of executing each query; we refer to the *per query* aggregate of these ratings as *actual execution probabilities*. This was achieved by presenting participants with a concrete description of a *stranded traveller* scenario and asked to rate (on a scale of 0-10) the likelihood that, if stranded at a specific airport with access to our hypothetical application, they would ask various questions involving certain locations near the airport. The degree of correlation between the predicted and actual execution probabilities was calculated on a per-airport basis (and for each permutation of weights $w_{\mathsf{ip}}, w_{\mathsf{perf}} \in \{1, 2, 10\}$ and aggregation functions ξ_{min} and ξ_{avg} used in computing these probabilities) using *Spearman's rank correlation coefficient* ρ. Table 1 shows the highest and lowest values of ρ for all queries across each airport. Comparison of each ρ to the critical values (taken from [5]) in the final column shows that all permutations of weights and aggregation functions produced correlations that are statistically significant at the 5% *alpha level* (α=0.05), for all airports. Therefore, we conclude that our approach is able to predict, with a high degree of accuracy, the actual execution probability of queries instantiated with a wide range of values in the stranded traveller scenario.

Airport	N	Lowest ρ	Highest ρ	Crit. Val. at α=0.05
Coleman	151	0.216	0.328	0.165
Edmonton	122	0.289	0.566	0.165
Halifax	86	0.321	0.660	0.179

Table 1: Highest and lowest values of Spearman's rank correlation coefficient ρ for all queries across each group.

5. REFERENCES

[1] O. Hartig and T. Heath. Populating Personal Linked Data Caches using Context Models. In *Proceedings of WWW*, 2012.

[2] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011.

[3] G. Klyne and J. J. Carroll (eds.). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, Feb. 2004.

[4] E. Prud'hommeaux and A. Seaborne (eds.). SPARQL Query Language for RDF. W3C Recommendation, Jan. 2008.

[5] P. H. Ramsey. Critical Values for Spearman's Rank Order Correlation. *Journal of Educational Statistics*, 14(3), 1989.

[1] In our cache population approach [1], C_{fut} results from an enrichment process applied to a given set of forecasted CAs. Due to space constraints we omit the definition of that process in this paper.