Estimating the dynamics of SPARQL query results using binary classification

Alberto Moya Loustaunau and Aidan Hogan

IMFD; DCC, University of Chile
{amoya,ahogan}@dcc.uhile.cl

Abstract. We address the problem of estimating when the results of an input SPARQL query over dynamic RDF datasets will change. We evaluate a framework that extracts features from the query and/or from past versions of the target dataset and inputs them into binary classifiers to predict whether or not the results for a query will change at a fixed point in the near future. For this evaluation, we create a gold standard based on 23 versions of Wikidata and a curated collection of 221 SPARQL queries. Our results show that the quality of predictions possible using (only) features based on the query structure and lightweight statistics of the predicate dynamics – though capable of beating a random baseline – are not competitive with results obtained using (more costly to derive) knowledge of the complete historical changes in the query results.

Keywords: SPARQL · Linked Data · Dynamics · Wikidata

1 Introduction

Many applications that consume Linked Data (LD) face challenges related to remote changes in the underlying data. Client-side caches can reduce the network traffic between clients and servers, the load on servers, and the average latency of responses. However, since datasets change over time, for caches to be useful, they should be updated when the underlying data that they reflect change; predicting such remote changes, however, is a challenging problem, particularly when data are accessed as the results of queries to a SPARQL endpoint.

Since datasets change over time, long-running applications that cache and repeatedly use query results obtained from an external SPARQL endpoint may resubmit the queries regularly to ensure up-to-dateness. As a result, without further information as to the dynamics of a particular SPARQL query, applications face the choice of either performing frequent query executions that may be redundant and repeatedly return the same results (aiming for stronger consistency at the cost of more frequent querying), or performing infrequent query executions that may lead to stale data being persisted in the application when the underlying sources change (accepting weaker consistency to improve efficiency/scalability). Given the costs for clients and servers of repetitive requests served over the Web and the potential efficiency gains offered by local caches, several weak consistency approaches have been proposed that try to keep the local data of the applications updated at lower cost by predicting changes [15,6,23].

Some works study data dynamics based on the historical evolution of entities [13,7,21], but following such an approach for SPARQL queries is expensive because (1) a SPARQL query may involve potentially many entities; and (2) it is necessary to have the previous complete versions of data to analyse the entities relevant to a query. Many works have explored the dynamics of Linked Data with the intention of finding patterns that allow for characterizing, recognizing, and predicting changes based on analysis of the domains, predicates, and schema [27,13,22,23,11]. Among these works are hybrid approaches developed to return fresher query results with faster response times by decomposing a query into dynamic sub-queries executed remotely, and static sub-queries executed over local caches [29], but such an approach only focuses on estimating the dynamics of individual triple patterns, and does not consider, for example, query operators. More generally, coping with changes in remote data still presents a major challenge for applications leveraging dynamic Linked Data.

Given a SPARQL query and a dynamic RDF graph (consisting of multiple historical versions), in this work, we address the problem of predicting whether or not the query's results will change in the next version of the RDF graph. To the best of our knowledge, this is the first work to address this problem. Along these lines, we evaluate a general architecture based on Machine Learning that extracts static features from a query, as well as features from the query and dynamic dataset. These features are fed into a binary classifier to predict whether or not the query results will change in a fixed point in the future, or, more ambitiously, can be fed into a regression model to predict when the query results are likely to change. With respect to the features used, we show that there is a trade-off between those that are easy to compute but offer less accurate predictions (e.g., static query features) versus those that are more costly to compute but offer more accurate predictions (e.g., historical changes in query results). Per this trade-off, which features to use for predicting the dynamics of query results may then depend on the particular application.

In order to better understand this conceptual trade-off – and more generally, to evaluate the quality of predictions made by our framework – we create a novel gold standard based on 23 weekly versions of the Wikidata knowledge graph [33] and 221 user-generated queries; we show this gold standard to have a variety of desirable features, including (most importantly) a balance of queries whose results never change, always change, as well as non-trivial queries whose results intermittently change. Using this gold standard, our experiments show that although features based on static characteristics of the query and statistics of changes in the data for individual predicates are more efficient to compute and maintain, they do not offer the same prediction quality as features based on knowledge of historical changes of the input query's results.

Contributions: (1) We evaluate a general architecture for predicting when/if the results of a SPARQL query will change in a future version of a dynamic RDF dataset. (2) We evaluate a number of features to instantiate this architecture based on analysis of the query, analysis of the dynamics of predicates in the data, and analysis of historical changes in the query results. (3) We create a gold standard for these tasks based on 23 consecutive versions of Wikidata and a set of 221 real-world SPARQL queries. (4) We use this gold standard to compare the predictions obtained using different types of features and classifiers.

2 Related Work

A variety of works have addressed the issue of modeling and consuming dynamic Linked Data from a broad range of perspectives [24,26,27,30,13,9,12,8,14,22,11]. One of the major challenges considered is that of keeping cached copies of remote dynamic data – cached for reasons of efficiency and scalability – up-to-date on the consumer side, which we refer to as the *synchronization problem*.

Some works have addressed the synchronization problem on the publisher side, proposing notification mechanisms that keep registered consumers informed about relevant changes to the data [18,24,26,16,12,10]; although such approaches may facilitate strong consistency – meaning that consumers are kept up-to-date with the remote data on the publisher side – they centralize the burden of synchronization on the publisher, potentially leading to scalability issues.

Conversely, a variety of works have looked at building models of remote data that can help to predict which data are most dynamic, and which are most static, indicating which subsets of the data may need be refreshed from the remote source more often [24,26,27,30,13,9,22,11]; such works consider changes in RDF datasets at differing levels of granularity, including documents [13], domains [13,22,23], predicates [13,23], characteristic sets[1] [22,11], etc. Features at different levels of granularity can be fed into different predictive models based on Poisson Processes [27], Markov Chains [32], Empirical Distributions [19], Machine Learning classification and regression [23,11], as well as a variety of other heuristics [2,32,15] and metrics [6,15,1]. Such approaches obviate the need for a subscription/notification mechanism. However, to ensure strong consistency in the presence of highly dynamic data, consumers may need to conservatively send a great many refresh requests to the server, which may be even more costly than a subscription/notification mechanism; hence such approaches are better suited for scenarios where weak consistency is more acceptable.

Specifically regarding the dynamics of SPARQL query results, Passant and Mendes [24] proposed SPARQLPuSH as a notification framework based on PubSubHubBub (recently standardized as WebSub [10]) aiming for strong consistency. Rather aiming for weak consistency, Umbrich et al. [29,31,28] proposed various methods to obtain knowledge about dynamics for different query patterns, mainly based on predicates. Dehghanzadeh et al. [5] later proposed a method to estimate the freshness using cardinality estimation techniques based on predicates and characteristic sets. Combining the notion of subscription-based notifications and predicting dynamics, Knuth et al. [15] propose a middleware to

[1] A characteristic set is the set of predicate terms used to describe a given subject [20].

which consumers may subscribe that periodically ranks and schedules refreshes for queries according to policies that take into account how likely the results are to be stale, how long ago the query was last refreshed, how many results previously changed, how long the query takes to run, etc.

Novelty: Given a query and a dynamic RDF dataset, we aim to predict whether or not the query's results will have changed in a fixed point in the near future. Our work thus complements existing works aiming for weak consistency, but (i) generalizes the problem, evaluating a framework that can incorporate statistics on predicate dynamics [29,31,28,5] and historical changes in query results [15] (as proposed in prior works for addressing related but yet distinct problems relating to dynamic data), as well as novel types of features, (ii) introduces new features based on query operators and statistics; (iii) creates a novel gold standard based on Wikidata and presents comparative results that indicate the relative predictive power inherent in different types of features.

3 Preliminaries: RDF and SPARQL

RDF is a conceptual data model based on directed graphs that can be used to describe resources on the Web. *RDF terms* are elements of the set $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$ composed of IRIs $\mathbf{I}$, literals $\mathbf{L}$, and blank nodes $\mathbf{B}$. A tuple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times (\mathbf{I}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is called an *RDF triple*, where s is called *subject*, p is called *predicate*, and o is called *object*. An *RDF graph* is a set of RDF triples.

SPARQL is the recommended query language to retrieve and manipulate data stored in the RDF format. In this work, we focus on SPARQL `SELECT` queries, where we will first define a SPARQL 1.0 query. Let $\mathbf{V}$ be a set of variables disjoint from the set of RDF terms. A *SPARQL expression* is built recursively as follows. (1) A *triple pattern* $t \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{B} \cup \mathbf{V})$ is an *expression.* (2) If Q_1 and Q_2 are expressions and R is a filter condition, then Q_1 `FILTER` R, Q_1 `UNION` Q_2, Q_1 `OPTIONAL` Q_2, Q_1 `AND` Q_2 are *expressions*. Finally, if Q is an expression, V a list of variables and Δ a boolean value, $\texttt{SELECT}^{\Delta}_{V} Q$ is a SPARQL `SELECT` query, where V denotes the projected variables, and Δ the `DISTINCT` option that when true, removes duplicate results. The semantics of a SPARQL `SELECT` query Q is defined in terms of its *evaluation* over an RDF graph G, denoted $Q(G)$, giving a set of partial mappings from projected variables to the set $\mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$; we refer to Pérez et al. [25] for definitions. (A SPARQL query is in fact evaluated over a set of *named graphs*; though we consider RDF graphs here, our methods also generalize to the named graphs setting.)

Our method supports SPARQL 1.1 `SELECT` queries, which allow a variety of additional features. One key feature in this extension is that of *property paths*, which allows for matching arbitrary length paths in an RDF graph, potentially returning or matching the endpoints of the path. An IRI p is a *path expression*; if e, e_1 and e_2 are path expressions, then `^`e (inverse of e), e_1`/`e_2 (e_1 followed by e_2), e_1`|`e_2 (e_1 or e_2), e`*` (zero or more e), e`+` (one or more e), e`?` (zero or one e), and `(e)` (parentheses used for precedence) are also *path expressions*; finally, if p, p_1,

$\dots, p_n$ are IRIs, then $\texttt{!}p$, $\texttt{!(}p_1\texttt{|}\dots\texttt{|}p_n\texttt{)}$ and $\texttt{!(}p_1\texttt{|}\dots\texttt{|}p_k\texttt{|\^{}}p_{k+1}\texttt{|}\dots\texttt{|\^{}}p_n\texttt{)}$ for $k+1 \leq n$ (negated property sets) are path expressions. Thereafter, a *path pattern* (s, e, o) where e is a path expression is an expression. Other features supported in SPARQL 1.1 include sub-queries, negation, aggregation, value binding, and so forth; for brevity, we do not introduce definitions for all such features.

One topic we do wish to highlight, however – as it relates to the behavior of a query over a dynamic RDF graph – is that of the *monotonicity* of SPARQL queries [3,4]. We say that a `SELECT` query Q is monotone if and only if $G_1 \subseteq G_2$ implies that $Q(G_1) \subseteq Q(G_2)$ for RDF graphs G_1 and G_2; intuitively, as data are extended, the results of a monotone query can only be extended. Monotonic SPARQL features include, for example, joins, unions, paths and filters; on the other hand, non-monotonic SPARQL features include negation and optional [3].

4 Predicting the Dynamics of Query Results

We consider a *dynamic RDF graph* to be a sequence of n discrete versions of an RDF graph denoted $\mathcal{G} := (G_1, \dots, G_n)$; in practice, we assume these versions to have regular intervals (e.g., hourly, daily, weekly, etc.). Further given a SPARQL (1.1) `SELECT` query Q, we would like to have knowledge of the dynamics of its results. The specific problem we consider in this work – which we call One Shot Change (OSC) – accepts $\mathcal{G}$, Q and a positive integer k as input and outputs a boolean value predicting whether or not the results will change from G_n to some future version G_{n+k} (i.e., $Q(G_n) \neq Q(G_{n+k})$?). In the case of bag semantics, we define "$\neq$" in terms of bag inequality, meaning that two bags of query results are different if the multiplicity of any result differs. We thus currently focus on predicting if the results of a query will change, rather than predicting when, or to what extent, they will change; these latter problems are left for future work.

Architecture: In Figure 1, we provide an overview of a general architecture for making predictions with respect to the OSC prediction task. A SPARQL query Q and a dynamic RDF graph $\mathcal{G}$ are given as input. The Query Feature Extractor extracts static features from the query, which include statistics about the number of elements in the query (e.g., number of triple patterns, variables, etc.) as well as the presence of particular query operators (e.g., `UNION`, `FILTER`, recursive path expressions, etc.). The Predicate Feature Extractor takes as input the predicates from the query and statistics about those predicates from the dynamic graph to produce a set of aggregated numerical features about the dynamicity of predicates used in the query. The Result Feature Extractor takes as input the results of the query over past versions of the dynamic graph from which it produces further features. All such features are passed to a (pre-trained) binary classifier to generate the final OSC prediction. We will now describe in more detail the query, data and result features considered in this work.

Query Features: Our initial set of features is based on static analysis of the input SPARQL query. Figure 2 shows only a sample of evaluated features, as well as – for the purposes of illustration – their values for an example query.

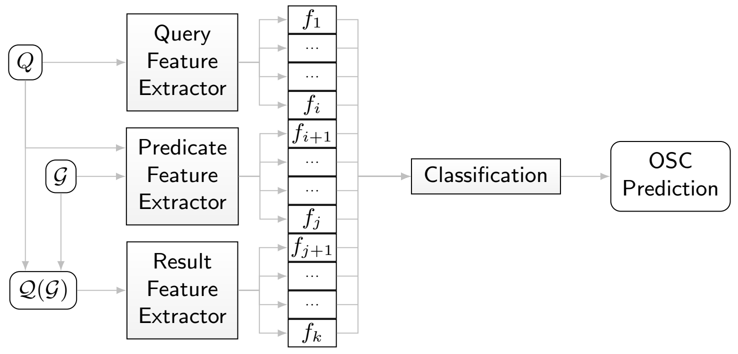

Fig. 1. Proposed architecture for predicting change in query results

The first group of features captures statistics about the query, indicating loosely its size and complexity. One might consider that the higher these values are, in general, the more dynamic we can expect the query to be since there are more "opportunities" for the query results to be affected by change. In some cases, however, the hypothesized correlation is not direct since, for example, adding more triple patterns may serve to narrow the query down and focus it on a static part of the graph (e.g., when looking for the movies of directors, adding a triple pattern to restrict the results to directors who have died may reduce the likelihood of changes in the results in a future version). Hence it will be of interest to see, experimentally, how these features affect the predictions.

The second group indicates the query operators used; we capture the presence or absence of query operators and solution modifiers from SPARQL 1.1.

In the third group gathers related features into one dimension: in the case of *Recursive path*, we group queries with path expressions of the form e`*` or e`+`. On the other hand, in *Negation*, we group non-monotonic features that allow for modeling difference (`MINUS`, `NOT EXISTS`, `!BOUND`[2]. These features – though course-grained – are straightforward to extract from a query, and offer valuable insights into how the query may behave in a dynamic query; for example, the *Negation* feature captures information about the (non-)monotonicity of the query, while we suppose that *Recursive paths*, which may traverse an arbitrary number of triples in the graph, might be more sensitive to change. Again, such correlations are not without exception and will require empirical study.

Predicate Features: The next component extracts features that capture information about the dynamics of predicates used in the query (without *evaluating* the query on the dynamic graph). This component captures how often and how many triples change for a predicate in a time interval, with the idea that – as in previous works [29,31,28,5] – predicates capture rich information about the dynamics of a dataset, where the results of queries with dynamic predicates will be more sensitive to change. Assuming that the number of predicates is *relatively*

[2] We recall that `!BOUND` can be combined with `OPTIONAL` to express negation.

```
SELECT ?item
WHERE {
  ?item :instance_of :human .
  ?item :gender :female .
    { ?item :place_of_birth :Wales }
  UNION
    { ?item :place_of_birth ?pob .
      ?pob :located_in* :Wales }
  OPTIONAL
    { ?sitelink schema:about ?item .
      ?sitelink schema:inLanguage "cy" }
  FILTER (!BOUND(?sitelink))
}
LIMIT 100
```

№ of triple patterns	7
№ of variables	3
№ of projected variables	1
№ of predicates	6
`FILTER`	✓
`LIMIT`	✓
`UNION`	✓
`GROUPBY`	✗
Sub-query	✗
Recursive path	✓
Negation	✓
Predicates	{ `:instance_of`, ... }

Fig. 2. Example query and a sample of the features extracted from it

```
:LMessi :name "L. Messi" ;
 :instance_of :Human ;
 :children 1 ;
 :played :2014WC .
```

```
:LMessi :name "L. Messi" ;
 :children 2 ;
 :played :2014WC, :2018WC .
```

```
:LMessi :name "L. Messi" ;
 :instance_of :Human ;
 :children 3 ;
 :played :2014WC, :2018WC .
```

Fig. 3. Example dynamic RDF graph $\mathcal{G} = (G_1, G_2, G_3)$ (from left to right, resp.)

```
:LMessi :instance_of :Human ;
 :children 1 , 2 ;
 :played :2018WC .
```

```
:LMessi :instance_of :Human ;
 :children 2 , 3 .
```

Fig. 4. Changes for $\mathcal{G}$ in Figure 4, showing $G_1 \oplus G_2$ (left) and $G_2 \oplus G_3$ (right)

low, such statistics can be easier to maintain. Formally, given two RDF graphs G_i and G_j, we denote by $G_i \oplus G_j$ the set of triples $(G_i \cup G_j) - (G_i \cap G_j)$ where "$-$" denotes set difference; in other words, $G_i \oplus G_j$ denotes the triples in one graph or the other but not both (noting that $G_i \oplus G_j = G_j \oplus G_i$). Next, given an RDF graph G and an IRI p, let $\#(G, p) := |\{(x, y, z) \in G : y = p\}|$ denote the number of triples in G with predicate p. Finally, given a dynamic RDF graph $\mathcal{G} := (G_1, \ldots, G_n)$ and a predicate p, we denote by $\Delta(\mathcal{G}, p) := \Sigma_{i=1}^{n-1} \frac{\#(G_i \oplus G_{i+1}, p)}{\#(G_i \cup G_{i+1}, p)}$ the normalized sum of the number of triples with the predicate p that changed between pairs of consecutive versions.

Example 1. Consider the example dynamic RDF graph $\mathcal{G}$ in Figure 3 (based on real data from Wikidata, with IRIs modified for the purposes of readability). In Figure 4 we show the pairwise changes between each version: $G_1 \oplus G_2$ and $G_2 \oplus G_3$. For a predicate p, the value $\Delta(\mathcal{G}, p)$ is then the sum of triples with predicate p in the graphs $G_1 \oplus G_2$ and $G_2 \oplus G_3$ divided by the number of triples with predicate p in the graphs $G_1 \cup G_2$ and $G_2 \cup G_3$. Looking at Figure 4, for example, `:name` does not appear ($\Delta(\mathcal{G}, \texttt{:name}) = 0$), `:children` appears twice in $G_1 \oplus G_2$ and $G_1 \cup G_2$, as well as, twice in $G_2 \oplus G_3$ and $G_2 \cup G_3$ ($\Delta(\mathcal{G}, \texttt{:children}) = 4/4 = 1$), and so forth.

Given a query Q with a set of predicates $\{p_i, ..., p_n\}$, we may then consider a variety of aggregate functions over $\Delta(\mathcal{G}, p_1), ..., \Delta(\mathcal{G}, p_n)$, such as max, mean, etc., to compute a final numeric feature for the query, representing a summary of the level of dynamicity of the predicates it contains.

Result features: The third type of feature we capture indicates how many times the query results Q have changed over the past versions of the dynamic graph. While this feature offers rich information for prediction, given a query that has not previously been seen, it is costly to compute, since it requires the evaluation of the query over each past version within an interval, which in turn requires maintaining indexes over the full data for a variety of past versions.

5 Gold Standard Dataset

In order to evaluate the effectiveness for predicting changes in SPARQL query results, we require a dynamic RDF graph, with access to various historical versions; preferably this graph contains real-world, large-scale, diverse RDF data, and with sufficient changes between versions to provide both positive and negative examples of queries whose results change. Furthermore, we require a set of SPARQL queries that can be answered against this dynamic graph; preferably these queries again should be diverse, representative of real-world user queries, of a variety of shapes and sizes, using a variety of query operators, and with a mix of both dynamic and static results over the dynamic graph. In particular, we choose Wikidata [33] for our experiments, which, we shall argue, meets the aforementioned requirements. We first give details of the dynamic data we collect from Wikidata; thereafter, we discuss the queries we use in our experiments.

RDF Data: We use 23 Wikidata snapshots from 18/04/2017 to 27/09/2017, which are captured (roughly) weekly in the *truthy version* that contains triples (without qualifiers) that have the best non-deprecated rank for a given property; this data corpus was previously collected and used by Gonzalez and Hogan [11]. The first version has 1,102,242,331 triples and 3,276 unique predicates, while the final version has 1,924,967,162 triples (+74%) and 3,660 unique predicates (+11%). Figure 5, on the left, shows the growth in triples as the time progress, while on the right, shows the numbers of triples added and removed version-to-version. We see that although many more triples are added than removed, there are some triples removed each version. We further note some peaks in triples added in some versions, which may be due to bulk imports of data. Between version 11 and 12 we have an almost 2 week gap because we were not able to obtain the data for that week, but this causes only the third highest peak. The dynamic graph considers a total of 32.3 billion triples across 23 versions.

Table 1 shows the ten most dynamic predicates according to the number of added and deleted statements involving that predicate (Total) and the ratio of added and deleted statements divided by the total number of statements for that predicate across all snapshots (Dyn); for the latter, we only include predicates

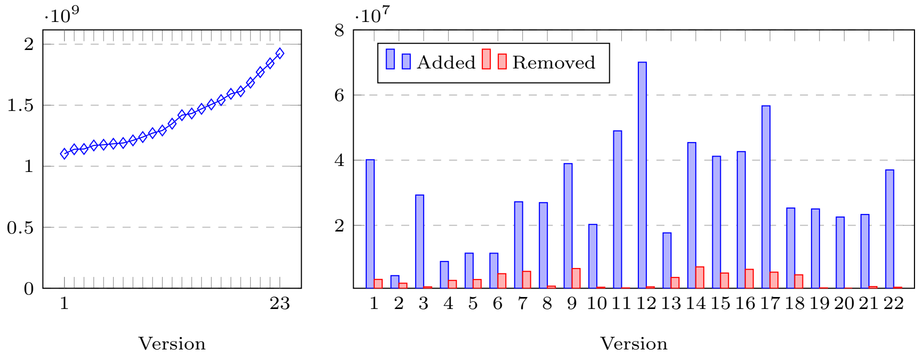

Fig. 5. Evolution of total triples in Wikidata (left) and number of triples added to and removed from the n^{th} version for the $n+1^{\text{th}}$ version (right)

Table 1. Top-10 dynamic predicates per total (left) and proportional (right) changes

№	Predicate	Total	Dyn
1	schema:description	464567354	0.04
2	schema:dateModified	99330382	0.14
3	schema:version	99330164	0.14
4	schema:name	44487864	0.01
5	rdfs:label	44487864	0.01
6	skos:prefLabel	44487864	0.01
7	wdt:P2093 (author name)	30671685	0.09
8	rdf:type	21808957	0.02
9	wdt:P31 (instance of)	12483471	0.02
10	schema:about	10773295	0.02

№	Predicate	Total	Dyn
1	owl:complementOf	156862	0.99
2	owl:onProperty	156862	0.99
3	owl:someValuesFrom	86674	0.71
4	wdt:P2462 (member of the deme)	3563	0.62
5	wdt:P3383 (film poster)	4388	0.58
6	wdt:P2331 (Cycling Archives ID)	4229	0.33
7	wdt:P1112 (*deleted*)	4366	0.20
8	wdt:P505 (general manager)	4959	0.18
9	schema:dateModified	99330382	0.14
10	schema:version	99330164	0.14

that appear in all snapshots and appear in $\geq$ 1,000 statements overall. Though the largest number of changes involves the predicate `schema:description`, its dynamic value is low due to it being a common predicate. On the other hand, the OWL properties have high dynamicity ratios due to taking blank node values; we currently do not consider isomorphism of graphs due to blank nodes.

Queries: The corpus collected by Gonzalez and Hogan [11] does not offer queries. In order to achieve a set of SPARQL queries that are answerable over Wikidata and with which we could run experiments, we took the user-contributed example queries from the Wikidata Query Service[3], consisting of 389 SPARQL 1.1 `SELECT` queries of varying degrees of complexity, touching upon various domains of data, and with a mix of varying query operators.

A total of 96 queries had to be removed from the original set of 389 because they asked for information external to Wikidata (using `SERVICE`) or asked for qualifiers that are not present in the truthy version. We also eliminated 12 queries that returned bindings to blank nodes to facilitate comparison between

[3] `https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples`

Table 2. Distribution of triple patterns (C: Constant, V: Variable, P: Path).

Pattern	**№**	**%**
V C V	435	59.43%
V C C	190	25.96%
V V V	9	1.23%
C C V	6	0.82%
V V C	5	0.68%
C V V	3	0.41%
C V C	1	0.14%
C C C	0	0.00%
V P C	69	9.43%
V P V	13	1.78%
C P V	1	0.14%
C P C	0	0.00%

Table 3. Distribution of paths

Pattern	**№**	**%**
e`*`	62	74.70%
e_1`/`e_2	56	67.47%
e`+`	8	9.64%
e_1`|`e_2	6	7.23%
e`?`	6	7.23%
`!(`e`)`	0	0.00%
e	83	100.00%

the results. Furthermore, some queries featured non-deterministic elements – such as `LIMIT`/`OFFSET` without ordering, `SAMPLE` or temporal functions – such as `TODAY()` – that may lead to changes in results not related to changes in the data; we remove 23 queries with `SAMPLE` and temporal functions, and add an `ORDER BY` clause to all queries to ensure determinism with `LIMIT`/`OFFSET` and more generally to facilitate quick comparison of results. We also eliminated 37 queries with empty results for all snapshots. As a result of this process of filtering, we end up with a total of 221 non-empty deterministic queries.

In Table 2, we provide statistics on the distribution of different types of triple patterns in the 221 queries before the transformation; of key importance is that 86.20% of the triple patterns have a constant in the predicate position, meaning that they are compatible with a model based on the dynamics of predicates. We also see that 83 triple patterns (11.34%) feature a path expression, where Table 3 provides the distribution of these expressions (remarking that multiple patterns can be used in one expression); of note here is that recursion is commonly found (74.70% use e`*` while 9.64% use e`+`), and that we find no negated property paths.

Finally, we look at how the results for these queries change over the 23 versions. Figure 6 shows how many queries had some results change between two consecutive versions, where we see that approximately half of the result sets change in each version. But are these always the same queries that change every time? Figure 6 shows the number of versions in which some result changed for each query; the queries are ordered by the number of changes in their results, where we can see that the results of 44 queries (19.90%) change each time, 17 queries (7.69%) never change, and more generally, we note a quite uniform distribution of queries in between. More generally speaking, we conclude that our query set has a good balance of queries whose results never change, queries whose results always change, and queries whose results sometimes change.

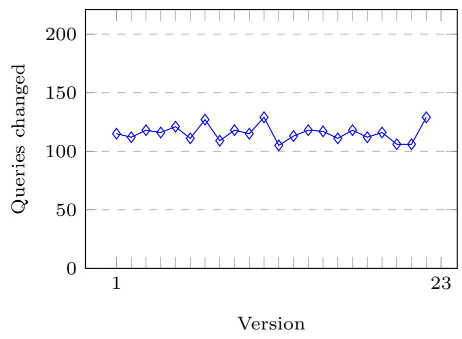

Fig. 6. Number of queries with some results changed for each version

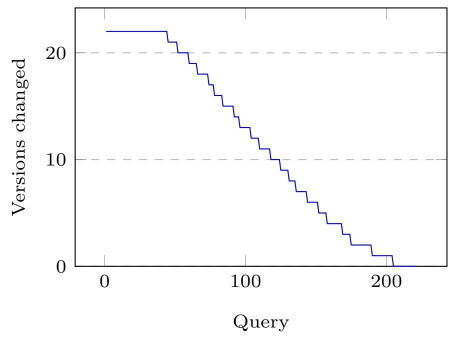

Fig. 7. Number of versions in which some results changed for each query

Feature extraction: To compute the query features, we use Apache Jena to parse the query and extract the necessary statistics and determine the presence of the relevant query operators. In order to extract statistics on the dynamics of individual predicates in the dynamic graph, a challenge here is scalability, since we work with a total of 32.3 billion triples; hence we (1) sort each version of Wikidata, (2) apply a merge-sort iterator over each pair of consecutive versions (G_i and G_{i+1}) to detect triples that changed ($G_i \oplus G_j$), writing a separate file for triples that were deleted ($G_i - G_j$) and added ($G_j - G_i$), (3) from these files, we can then compute and sum the number of triples changed for each predicate between each pair of versions. Finally, in order to create the ground truth in terms of which results change between which versions, we index each version in Virtuoso and compute the results for each query against each version and write them to disk; we then compare consecutive pairs of results with a merge-sort.

6 Evaluation

Our experiments evaluate the (relative) quality of One Shot Change (OSC) predictions that can be made based on query features, predicate features, result features, as well as combinations thereof, using a selection of binary classifiers.

Setting: To build our final datasets, we must define a past interval to consider. With 23 versions, we must hold out at least one version to predict, leaving a maximum interval of 22 past versions to use for computing our features. However, the more past versions we consider for a prediction, the fewer examples we can generate; for example, if we consider 22 versions, we will only have one ground truth label for each query. In the end, we opted to consider intervals of 3, 5, 9 and 17 previous versions, for example, in the case of an interval of three versions, we use $(G_{i-3}, G_{i-2}, G_{i-1})$ to predict changes in queries results for the subsequent version (G_i), which allows us to predict OSC from G_4 to G_{23} inclusive, providing 20 labeled examples per query. We thus have 4,420 labeled examples for intervals of 3, ranging down to 1,326 examples for intervals of 17. No feature explicitly indicates the versions given or the version to be predicted.

In the case of predicate dynamics, as discussed in Section 4, there are potentially many predicates per query, each with its own value for $\Delta(\mathcal{G}, p)$, where to reduce (and fix) dimensionality, we may apply an aggregate function to choose the min, max or mean value over all predicates. In preliminary experiments, we found the mean value to offer the best results, followed by the max value; hence in what follows we adopt the mean value in our experiments.

For OSC prediction, we test four binary classifiers: Decision Trees, Linear SVM, Naive Bayes, and Nearest Neighbors. We split the data to use 80% for training and 20% for tests. To avoid overfitting, we use stratified 5-fold cross-validation. We consider three types of features, as previously described – query features (Q), predicate features (P) and result features (R) – considering historical data (only) from the fixed interval. We also consider combinations of these features: QP, QR, PR and QPR.

Results: Table 4 shows the F_1-score for the predictions made considering the union of different combinations of the sets of features (Q, P, R) and varying the window sizes (3, 5, 9, 17); for reference, we include a baseline that randomly guesses `yes`/`no` respecting the observed class distribution. Given the number of configurations presented, we only include F_1 scores for reasons of space; however, we remark that the Precision and Recall scores were in general quite balanced. The best results were obtained using the features from R, where, as can be expected, prediction performs best when knowledge of changes in the historical results of the input query is available. The features from sets Q and P – which do not assume the availability of such information – obtain a significantly lower F_1-score; the best result including R was with PR ($F_1 = 0.831$) using Linear SVM and window size 17, while the best results without R was with Q ($F_1 = 0.60$) using Linear SVM and window size 5.

Furthermore, in Figure 8, we show that the size of the window influences the quality of the results when considering R features (blue), being better as the window increases. However, there is no clear trend in the results for increasing window sizes when considering models without using R features (red). (As aforementioned, we can see that Precision and Recall results are comparable.)

7 Summary, Conclusions and Future Work

In this paper we evaluate methods to predict whether or not (OSC) the results of an input query will change at a fixed point in the near future. More specifically, we evaluate a framework based on binary classifiers that accept features extracted from the query, from past versions of the data, and/or from the combination of both. Considering this framework, we hypothesize that there is a conceptual trade-off between the cost of computing features and their value for prediction: features extracted from queries alone are the most efficient to extract, not requiring historical data, but are quite coarse-grained for prediction; on the

Table 4. F_1-score for tested classifiers considering different sets of features (Q, P, R) and window sizes (3,5,9,17)

	Classifier	Q	P	R	QP	QR	PR	QPR
$w = 3$	*Random Baseline*	0.499	0.489	0.502	0.489	0.495	0.495	0.516
	Decision Trees	0.497	0.519	0.781	0.497	0.569	0.706	0.600
	Naive Bayes	0.432	0.419	0.758	0.432	0.478	0.759	0.479
	Nearest Neighbors	0.505	0.517	0.776	0.509	0.679	0.781	0.680
	Linear SVM	0.596	0.372	0.758	0.593	0.775	0.762	0.774
	Best	**0.596**	**0.519**	**0.781**	**0.593**	**0.775**	**0.781**	**0.774**
$w = 5$	*Random Baseline*	0.480	0.496	0.499	0.505	0.495	0.497	0.499
	Decision Trees	0.485	0.514	0.782	0.494	0.581	0.711	0.603
	Naive Bayes	0.431	0.486	0.782	0.432	0.508	0.797	0.500
	Nearest Neighbors	0.517	0.535	0.795	0.525	0.676	0.790	0.673
	Linear SVM	0.600	0.384	0.784	0.590	0.787	0.789	0.787
	Best	**0.600**	**0.535**	**0.795**	**0.590**	**0.787**	**0.797**	**0.787**
$w = 9$	*Random Baseline*	0.497	0.505	0.493	0.483	0.497	0.511	0.515
	Decision Trees	0.456	0.517	0.793	0.48	0.668	0.735	0.651
	Naive Bayes	0.434	0.461	0.797	0.435	0.531	0.790	0.531
	Nearest Neighbors	0.514	0.528	0.787	0.517	0.653	0.800	0.648
	Linear SVM	0.578	0.397	0.797	0.571	0.795	0.800	0.796
	Best	**0.578**	**0.528**	**0.797**	**0.571**	**0.795**	**0.800**	**0.796**
$w = 17$	*Random Baseline*	0.486	0.494	0.482	0.512	0.524	0.511	0.490
	Decision Trees	0.511	0.511	0.827	0.469	0.712	0.712	0.71
	Naive Bayes	0.416	0.465	0.826	0.414	0.555	0.810	0.554
	Nearest Neighbors	0.548	0.480	0.826	0.554	0.744	0.820	0.731
	Linear SVM	0.554	0.438	0.826	0.560	0.819	0.831	0.815
	Best	**0.554**	**0.511**	**0.827**	**0.560**	**0.819**	**0.831**	**0.815**

other hand, features based on changes in results over previous versions for an (unseen) input query are often the most costly to acquire, but offer fine-grained information for prediction; finally, features based on the dynamics of predicates offer a balance between the two, allowing to summarize historical data into succinct statistics, thus offering more fine-grained information than static query features but more coarse-grained information than historical results.

We thus explore this trade-off experimentally using 23 versions and 221 user-contributed queries for Wikidata to form a gold standard dataset. We use this gold standard dataset to evaluate the trade-off identified between different types of features for OSC predictions. Our results show that the features based on historical changes to query results perform best (reaching $F_1 = 0.831$ in the best configuration), whereas considering static query features and predicate dynamics alone is less competitive (reaching $F_1 = 0.600$ in the best configuration) when historical results are not available or are prohibitively costly to compute. Com-

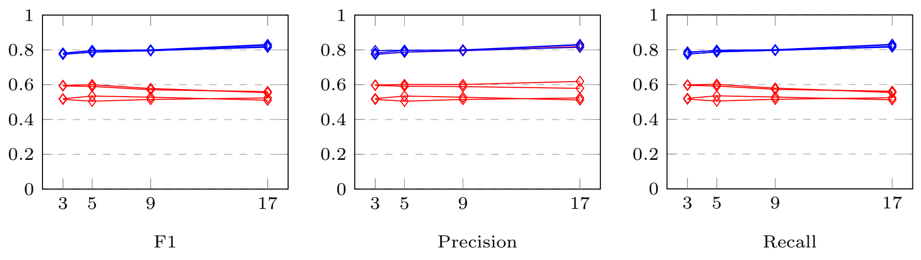

Fig. 8. Best results for combination of features in term of F_1-measure (left), Precision score (center) and Recall (right) with increasing windows sizes; blue indicates results using features from R (R, QR, PR and QPR) while red indicates results not using features from R (Q, P, QP and the Baselines)

paring query and predicate features, in fact, query features performed better, where predicate features alone only barely outperformed a random baseline.

In conclusion, our results show that having knowledge of the historical changes of the results of a query is important for improving the quality of OSC predictions using binary classifiers. However, in many scenarios, such knowledge is often not available or may not be practical to compute. Considering a real-world caching use-case, for example, historical changes in results may be maintained for queries that are frequently repeated with relatively low-cost, but for a previously unseen query, computing results for past versions at runtime would incur a prohibitive cost. Hence, we identify an open research question: is it possible to estimate the dynamics of query results without relying on (costly) knowledge about historical changes of query results while staying competitive with the quality of prediction possible when such knowledge is available?

Regarding future work, our gold standard based on Wikidata could be extended to consider more versions spanning a longer period of time and/or more queries; a very large query dataset for Wikidata was recently published [17], which may be of use here. Building gold standards based on other datasets would also help to diversify the evaluation process. Concerning the prediction tasks themselves, one promising direction may be to apply a more fine-grained analysis of the query, considering (for example) the selectivity of particular triples patterns as well as their dynamicity. There may also be better ways to perform such predictions without relying on binary classifiers, but rather using more analytical methods. Finally, it would be interesting to investigate the effectiveness of these techniques in practice, developing caching systems, synchronization schedules, and other applications, based on the evaluated predictions.

Material We make supplementary material (queries, data, results, etc.) available at `https://users.dcc.uchile.cl/~amoya/quweda2019/`.

Acknowledgements This work was supported by the Millennium Institute for Foundational Research on Data (IMFD), by Fondecyt Grant No. 1181896 and by CONICYT PFCHA/Doctorado Nacional/2017 - 21171070.

References

1. Akhtar, U., Amin, M.B., Lee, S.: Evaluating scheduling strategies in LOD based application. In: Asia-Pacific Network Operations and Management Symposium, APNOMS. IEEE (2017)
2. Alici, S., Altingövde, I.S., Ozcan, R., Cambazoglu, B.B., Ulusoy, Ö.: Adaptive time-to-live strategies for query result caching in web search engines. In: European Conference on IR Research, ECIR. Springer (2012)
3. Arenas, M., Pérez, J.: Querying semantic web data with SPARQL. In: Principles of Database Systems (PODS). ACM (2011)
4. Arenas, M., Ugarte, M.: Designing a query language for RDF: marrying open and closed worlds. ACM Trans. Database Syst. **42**(4) (2017)
5. Dehghanzadeh, S., Parreira, J.X., Karnstedt, M., Umbrich, J., Hauswirth, M., Decker, S.: Optimizing SPARQL query processing on dynamic and static data based on query time/freshness requirements using materialization. In: Joint International Conference, JIST. Springer (2014)
6. Dividino, R.Q., Gottron, T., Scherp, A.: Strategies for efficiently keeping local linked open data caches up-to-date. In: International Semantic Web Conference ISWC. Springer (2015)
7. Dividino, R.Q., Gottron, T., Scherp, A., Gröner, G.: From changes to dynamics: Dynamics analysis of linked open data sources. In: Extended Semantic Web Conference, PROFILES@ESWC 2014 (2014)
8. Dividino, R.Q., Kramer, A., Gottron, T.: An investigation of HTTP header information for detecting changes of linked open data sources. In: European Semantic Web Conference (ESWC). Springer (2014)
9. Dividino, R.Q., Scherp, A., Gröner, G., Grotton, T.: Change-a-lod: Does the schema on the linked data cloud change or not? In: Workshop on Consuming Linked Data, COLD. CEUR-WS.org (2013)
10. Genestoux, J., Fitzpatrick, B., Slatkin, B., Atkins, M.: WebSub. W3C Recommendation (Jan 2018), `https://www.w3.org/TR/websub/`
11. González, L., Hogan, A.: Modelling dynamics in semantic web knowledge graphs with formal concept analysis. In: World Wide Web Conference. ACM (2018)
12. Ibáñez, L.D., Skaf-Molli, H., Molli, P., Corby, O.: Col-graph: Towards writable and scalable linked open data. In: International Semantic Web Conference (ISWC). pp. 325–340. Springer (2014)
13. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing linked data dynamics. In: Extended Semantic Web Conference, ESWC. Springer (2013)
14. Kjernsmo, K.: A survey of HTTP caching implementations on the open semantic web. In: European Semantic Web Conference, ESWC. Springer (2015)
15. Knuth, M., Hartig, O., Sack, H.: Scheduling refresh queries for keeping results from a SPARQL endpoint up-to-date (short paper). In: Confederated International Conferences: CoopIS, C&TC, and ODBASE. Springer (2016)
16. Mader, C., Martin, M., Stadler, C.: Facilitating the exploration and visualization of linked data. In: Linked Open Data - Creating Knowledge Out of Interlinked Data - Results of the LOD2 Project. Springer (2014)
17. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of Wikidata: Semantic technology usage in Wikipedia's knowledge graph. In: International Semantic Web Conference (ISWC) (2018)
18. Missier, P., Alper, P., Corcho, Ó., Dunlop, I., Goble, C.A.: Requirements and services for metadata management. IEEE Internet Computing (2007)

19. Neumaier, S., Umbrich, J.: Measures for assessing the data freshness in open data portals. In: Open and Big Data, OBD. IEEE Computer Society (2016)
20. Neumann, T., Moerkotte, G.: Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In: International Conference on Data Engineering, ICDE. IEEE Computer Society (2011)
21. Nishioka, C., Scherp, A.: Temporal patterns and periodicity of entity dynamics in the linked open data cloud. In: Conference on Knowledge Capture, K-CAP. ACM (2015)
22. Nishioka, C., Scherp, A.: Information-theoretic analysis of entity dynamics on the linked open data cloud. In: International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES '16) ESWC. CEUR-WS.org (2016)
23. Nishioka, C., Scherp, A.: Keeping linked open data caches up-to-date by predicting the life-time of RDF triples. In: Conference on Web Intelligence. ACM (2017)
24. Passant, A., Mendes, P.N.: sparqlpush: Proactive notification of data updates in RDF stores using pubsubhubbub. In: Workshop on Scripting and Development for the Semantic Web. CEUR-WS.org (2010)
25. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. (2009)
26. Tramp, S., Frischmuth, P., Ermilov, T., Auer, S.: Weaving a social data web with semantic pingback. In: Knowledge Engineering and Management by the Masses - EKAW. Springer (2010)
27. Umbrich, J., Hausenblas, M., Hogan, A., Polleres, A., Decker, S.: Towards dataset dynamics: Change frequency of linked open data sources. In: WWW2010 Workshop on Linked Data on the Web, LDOW. CEUR-WS.org (2010)
28. Umbrich, J., Karnstedt, M., Hogan, A., Parreira, J.X.: Freshening up while staying fast: Towards hybrid SPARQL queries. In: Knowledge Engineering and Knowledge Management, EKAW. Springer (2012)
29. Umbrich, J., Karnstedt, M., Hogan, A., Parreira, J.X.: Hybrid SPARQL queries: Fresh vs. fast results. In: International Semantic Web Conference,ISWC. Springer (2012)
30. Umbrich, J., Karnstedt, M., Land, S.: Towards understanding the changing web: Mining the dynamics of linked-data sources and entities. In: LWA 2010 - Lernen, Wissen & Adaptivität, Workshop (2010)
31. Umbrich, J., Karnstedt, M., Parreira, J.X., Polleres, A., Hauswirth, M.: Linked data and live querying for enabling support platforms for web dataspaces. In: International Conferenceon Data Engineering, ICDE. IEEE (2012)
32. Umbrich, J., Mrzelj, N., Polleres, A.: Towards capturing and preserving changes on the web of data. In: European Semantic Web Conference ESWC (2015)
33. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)