Predicting SPARQL Query Dynamics

Anonymous Author(s)

ABSTRACT

Given historical versions of an RDF graph, we propose and compare several methods to predict whether or not the results of a SPARQL query will change for the next version. Unsurprisingly, we find that the best results for this task are achievable by considering the full history of results for the query over previous versions of the graph. However, given a previously unseen query, producing historical results requires costly offline maintenance of previous versions of the data, and costly online computation of the query results over these previous versions. This prompts us to explore more lightweight alternatives that rely on features computed from the query and statistical summaries of historical versions of the graph. We evaluate the quality of the predictions produced over weekly snapshots of Wikidata and daily snapshots of DBpedia. Our results provide insights into the trade-offs for predicting SPARQL query dynamics, where we find that a detailed history of changes for a query's results enables much more accurate predictions, but other alternatives we propose have significantly less overhead.

CCS CONCEPTS

• **Information systems** → **Temporal data**; **Resource Description Framework (RDF)**; **Query languages**.

KEYWORDS

Dynamics, Linked Data, SPARQL, RDF

ACM Reference Format:

Anonymous Author(s). 2018. Predicting SPARQL Query Dynamics. In *CIKM '21: 30th ACM International Conference on Information and Knowledge Management, November 01–05, 2021, Queensland, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recent years have seen increased interest in querying graphs, particularly in the context of NoSQL systems, Linked Data, Knowledge Graphs, etc. A prominent data model for graphs is the Resource Description Framework (RDF) [43], whose standard query language is SPARQL [18]. RDF and SPARQL have been specifically designed to cater for publishing, interlinking and querying data on the Web. Hundreds of datasets have been openly published on the Web using the RDF data model [43] under the Linked Data principles [6]. These datasets often provide a SPARQL [18] service (known as an *endpoint*) that clients can query over the Web [3]. The most prominent of these datasets – including the likes of DBpedia [26], LinkedGeoData [44], Wikidata [54], and YAGO [19] – provide public access to structured descriptions of millions of entities and their relations. SPARQL endpoints over such datasets can receive in the order of millions of queries per day from the Web [28, 42].

Many SPARQL endpoints exhibit problems relating to timeouts and intermittent availability [3]. These issues are partly attributable to three factors. First, when compared with relational database systems that have enjoyed several decades of research and development, graph database systems (in which we include SPARQL engines) are still relatively young. Second, SPARQL endpoints introduce a new and challenging topic: enabling structured queries on open datasets from arbitrary clients over the Web. Third, scale is a key challenge for many SPARQL endpoints, both in terms of the data indexed, and the number of clients and requests to serve.

An established way to improve the performance of database and Web-based systems is to apply caching techniques [25, 29, 37, 57]. Caching of partial results and other forms of data – be it in a local system, on a remote server, or on a remote client – can dramatically reduce the workload on the server and improve query throughput. Such techniques can reduce server load by reusing work already done for previous requests in order to lower the costs of current and future requests, and can reduce transport costs – particularly in the context of distributed and decentralized settings – by caching data on the client side. The benefits of caching have been demonstrated for SPARQL engines in local and Web settings [23, 25, 29, 37, 57].

However, datasets are subject to change over time [21], which may render cached data stale. When deciding what results to cache, it is important to consider how long the results will stay valid. On the server-side, caching dynamic results that will soon become stale can be a waste of resources. On the client-side, refreshing cached results by resending queries to the server is necessary to avoid stale data, but sending too many queries can be costly in terms of latency and bandwidth, where frequent refresh queries can put an unnecessary load on the endpoint's server [10, 22, 23, 36].[1]

Addressing related problems, numerous works have aimed to characterise and model the *dynamics* of RDF graphs and Linked Data at varying levels of granularity: at the level of entities [11, 16, 21, 34], predicates [16, 21, 35, 36, 48], triples [36] and sources [21, 48]. Dynamics, in this context, refers to *changes in data over time*; i.e., while a change is an instantaneous event, dynamics considers changes over time [11]. An important use-case for modelling dynamics is to predict future changes [16, 36]. In recent years, a variety of works have emerged to study the dynamics of RDF graphs and datasets at the level of entities [11, 16, 21, 34], predicates [16, 21, 35, 36, 48], triples [36] and sources [21, 48].

A greater understanding of the dynamics of RDF graphs and SPARQL query results would lead to better caching systems capable of scheduling refreshes, detecting data that are likely to be stale, etc., with lower costs. However, while the dynamics of RDF

CIKM '21, November 01–05, 2021, Queensland, Australia

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

[1] An alternative is to apply a *push-based model*, where the server notifies clients of changes [24, 38], but this becomes untenable when dealing with thousands or millions of clients. Knuth et al. [24] discuss further limitations of this approach.

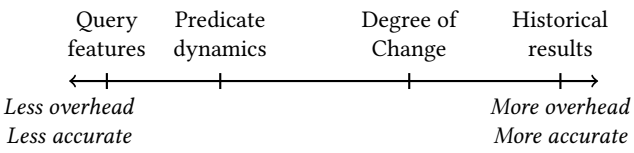

Figure 1: Hypothesised trade-off for predicting the dynamics of unseen queries

has been studied in numerous papers, few works have studied the dynamics of SPARQL query results. Umbrich et al. [50] characterise the dynamics of individual triple patterns based on predicate statistics, but do not consider more complex patterns. Knuth et al. [23] propose various scheduling policies for running refresh queries on the client, but require knowledge of the query's history of results; while such a history can be built by the client when issuing the same query repeatedly, it is not clear how to deal with previously unseen queries. One approach to deal with unseen queries is to manage and evaluate the query over the historical data [5, 14, 39, 46], but the costs in terms of time and space would obviate the benefits of making predictions: on the client side, all clients wishing to make predictions would need to duplicate the full historical data locally, making the server redundant, while on the server side, it would seem more efficient to simply evaluate the query over the current data rather than over all historical data to decide whether or not to cache the results. Moya and Hogan [32] propose a method for predicting when the results of a query will change without requiring historical results, instead proposing a machine learning model over a mixture of static query features and statistics of the dynamicity of individual predicates, but the accuracy of predictions based on these features falls far behind what is possible with historical results.

In this work, we propose various methods to predict whether or not a SPARQL query's results will change in the next version of a dynamic RDF graph. We explore a variety of methods, based on knowledge of historical query results, static query features, statistics about the dynamics of predicates used in the query, as well as estimates of the *degree of change*, i.e., the ratio of results that change considering adjacent pairs of historical versions. The latter approaches do not need historical query results, but rather use lightweight statistics extracted from the query and past versions.

We expect that the methods we propose follow a trade-off for predictions, as illustrated in Figure 1, where more accurate predictions imply greater overheads. On the left-hand side, we have query features, which require no knowledge of the data, and thus involve the least overhead, but provide less accurate predictions as a result. Next we have features about the dynamics of predicates used in the query, which require high-level statistics about how the triples involving a particular predicate change. Thereafter, the degree-of-change measures require more detailed statistics, but allow for computing more detailed meta-data regarding a particular query's sensitivity to changes in the graph. Finally, we have knowledge of historical results, which for an unseen query, requires evaluating the query on several historical versions, and maintaining indexes over those versions, thus implying a much higher overhead, but with the benefit of having much more detailed information about how its results have changed over past versions.

We incorporate these features as input into a machine learning model that is trained on the results for a selected set of queries for predicting changes over historical versions [32]. We compare the relative accuracy of predictions made with different sets of features and different models. Specifically, our experiments are based on queries evaluated over 17 weekly versions of Wikidata [54] and 18 daily versions of DBpedia Live [31].

Paper structure: Section 2 provides preliminaries for RDF/SPARQL. Section 3 then provides a formal statement of the problem we address. Section 4 discusses related works. Section 5 describes our proposed approach for predicting query dynamics. Section 6 presents the design and results of the experiments. We conclude and review limitations and future directions in Section 7.

2 PRELIMINARIES

We begin by defining preliminaries related to RDF and SPARQL.

RDF. is a graph-based data model recommended for use on the Web. RDF is based on three pairwise-disjoint sets of *RDF terms*: IRIs $\mathbf{I}$, literals $\mathbf{L}$, and blank nodes $\mathbf{B}$. An RDF *triple* $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times (\mathbf{I}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ comprises of a *subject*, a *predicate*, and an *object*. An *RDF graph* is a set of RDF triples.

SPARQL. is the query language recommended for use with RDF. In this work, we focus on `SELECT` queries using core features from SPARQL 1.0, namely basic graph patterns, unions, optionals, projection and distinct. Further query features can be supported as an extension of our proposed framework, left for future work.

Queries introduce a fourth set of terms: variables from the set $\mathbf{V}$ that are disjoint with IRIs, literals and blank nodes. 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 then an RDF triple that allows variables in any position. A *basic graph pattern* is a set of triple patterns. For simplicity, we assume that blank nodes in triple patterns are rewritten to fresh variables that are projected away. We currently do not support blank nodes in the solutions of a query, but they could be handled in future by skolemising them in the input graph, or by canonically labelling them in the solutions [20].

A SPARQL *query pattern* is built recursively as follows:

(1) A *basic graph pattern* B is a *query pattern*.
(2) If Q_1 and Q_2 are query patterns, then Q_1 `AND` Q_2, Q_1 `UNION` Q_2, and Q_1 `OPTIONAL` Q_2, are query patterns.[2]
(3) Finally, if Q is a query pattern, 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 $Q(G)$, which returns a set of *solution mappings* [40]. A solution mapping $\mu : \mathbf{V} \to \mathbf{I} \cup \mathbf{L}$ is a partial mapping from variables to RDF terms. We denote the set of all such mappings by $\mathbf{M}$. We denote by $\text{dom}(\mu)$ the *domain* of the partial mapping μ, i.e., the set of variables for which μ is defined. We say that two solution mappings μ_1 and μ_2 are compatible, denoted $\mu_1 \sim \mu_2$, if and only if $\mu_1(v) = \mu_2(v)$ for all $v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$.

Let M_1 and M_2 denote two sets of solution mappings, and V a set of variables. We define the following algebra for set semantics

[2] In concrete SPARQL syntax, `AND` is a default binary operator.

(resp., join, union, anti-join, left-outer-join and projection):

$$M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2 \text{ and } \mu_1 \sim \mu_2\}$$
$$M_1 \cup M_2 = \{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}$$
$$M_1 \triangleright M_2 = \{\mu_1 \mid \mu_1 \in M_1 \text{ and } \nexists \mu_2 \in M_2 : \mu_1 \sim \mu_2\}$$
$$M_1 \bowtie\!\!\!\!\bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \triangleright M_2)$$
$$\pi_V(M) = \{\mu \mid \exists \mu' \in M : (\mu \sim \mu' \text{ and } \mathrm{dom}(\mu) = V \cap \mathrm{dom}(\mu'))\}$$

Let B denote a basic graph pattern, i.e., a set of triple patterns. Let $\mathrm{vars}(B)$ denote the variables used in B. Given a solution mapping μ, let $\mu(B)$ denote the image of B under μ, i.e., the results of replacing each variable $v \in \mathrm{vars}(B) \cap \mathrm{dom}(\mu)$ with $\mu(v)$ in B. The evaluation of a SPARQL `SELECT` query over an RDF graph G, under set semantics, is then defined as follows:

$$B(G) = \{\mu \mid \mathrm{dom}(\mu) = \mathrm{vars}(B) \text{ and } \mu(B) \subseteq G\}$$
$$Q_1 \texttt{ AND } Q_2 = Q_1(G) \bowtie Q_2(G)$$
$$Q_1 \texttt{ UNION } Q_2 = Q_1(G) \cup Q_2(G)$$
$$Q_1 \texttt{ OPTIONAL } Q_2 = Q_1(G) \bowtie\!\!\!\!\bowtie Q_2(G)$$
$$\texttt{SELECT}_V^{\text{TRUE}} Q = \pi_V(Q)$$

For a query $\texttt{SELECT}_V^{\text{FALSE}} Q$, evaluation returns bags of solution mappings. Let $\mathfrak{M}$ denote a bag (i.e., multiset) of solution mappings. We denote by $\mathfrak{M}(\mu)$ the multiplicity of μ in $\mathfrak{M}$, and say that $\mu \in \mathfrak{M}$ if and only if $\mathfrak{M}(\mu) > 0$. Given a boolean condition ϕ, we use Iverson bracket notation, where $[\![\phi]\!] = 1$ if ϕ is true, or $[\![\phi]\!] = 0$ otherwise. Noting that $B(G)$ always returns a set of solutions (recalling our assumption that B has no blank nodes), the multiplicities of solutions under bag semantics is defined as follows:

$$\mathfrak{M}_1 \bowtie \mathfrak{M}_2(\mu) = \sum_{\mu_1 \in \mathbf{M}, \mu_2 \in \mathbf{M}} \mathfrak{M}_1(\mu_1) \cdot \mathfrak{M}_2(\mu_2) \cdot [\![\mu_1 \sim \mu_2 \wedge \mu_1 \cup \mu_2 = \mu]\!]$$
$$\mathfrak{M}_1 \cup \mathfrak{M}_2(\mu) = \mathfrak{M}_1(\mu) + \mathfrak{M}_2(\mu)$$
$$\pi_V(\mathfrak{M})(\mu) = \sum_{\mu' \in \mathbf{M}} \mathfrak{M}(\mu')[\![\mu \sim \mu' \wedge \mathrm{dom}(\mu) = V \cap \mathrm{dom}(\mu')]\!]$$

where we denote by $\mathbf{M}$ the set of all possible solution mappings. The multiplicity of solutions for $\mathfrak{M}_1 \bowtie\!\!\!\!\bowtie \mathfrak{M}_2$ follows from these definitions.

3 PROBLEM STATEMENT

Before continuing, we formally state the problem that we address. We consider a *dynamic RDF graph* to be an RDF graph that changes over time. We represent a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_n)$ as a tuple of n RDF graphs, where each RDF graph denotes a discrete version. As aforementioned, we currently make the simplifying assumption that RDF graphs are ground, i.e., that they do not contain blank nodes. Given a query and a dynamic RDF graph, the central problem that interests us in this paper is as follows:

Problem:	One-Shot-Change (OSC)
Input:	a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_m)$; a SPARQL query Q.
Output:	`true` if $Q(G_m) \neq Q(G_{m+1})$ is predicted; `false` otherwise.

Note that by $Q(G_k) = Q(G_k)$, we refer to bag equality, meaning that order does not matter, but multiplicities are taken into account.

Example 3.1. Consider the dynamic RDF graph $\mathcal{G} = (G_1, G_2, G_3)$ composed of three versions of Wikidata graphs shown in Figure 2.[3] Between G_1 and G_2, the name of Swaziland is changed to Eswatini and the old name is added as an alias. Between G_2 and G_3, the claim that Luke Hall has the occupation of swimmer is changed to state that his sport is swimming. Given a query Q such as the following, which looks for names of swimmers and their countries:

```
SELECT DISTINCT ?swimmerName ?countryName {
  ?swimmer rdfs:label ?swimmerName .
  ?swimmer wdt:P106 wd:Q10843402 . # occupation swimmer
  ?swimmer wdt:P27 ?country .      # nationality
  ?country rdfs:label ?countryName .
}
```

The goal of OSC is then to predict whether or not the results for $Q(G_4)$ will change with respect to G_3. □

Given a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_n)$, we can (train and) evaluate OSC with respect to the prefix $\mathcal{G}' = (G_1, \ldots, G_m)$ such that $m < n$; i.e., the withheld version G_{m+1} in $\mathcal{G}$ can be used to generate the ground truth for a query Q over $\mathcal{G}'$ by evaluating $Q(G_m)$ and $Q(G_{m+1})$ and checking if $Q(G_m) = Q(G_{m+1})$ holds.

In terms of desiderata, the procedure for this problem should provide accurate predictions (which we can evaluate by passing a sub-sequence of known versions, making predictions for held-out versions). Rather than requiring the full dynamic RDF graph $\mathcal{G}$ as input, ideally the procedure would accept $\mathrm{d}(\mathcal{G})$ and Q as input, where $\mathrm{d}(\mathcal{G})$ is a description of the dynamic graph that is computed from $\mathcal{G}$ but is (far) more concise than $\mathcal{G}$. A server wishing to implement OSC would then have less data to manage and could even publish $\mathrm{d}(\mathcal{G})$ so that clients can independently predict when results will change. For example, $\mathrm{d}(\mathcal{G})$ might be a statistical summary of the graphs in $\mathcal{G}$ and what changes between its versions.

Example 3.2. Consider the query Q from Example 3.1 and the dynamic RDF graph $\mathcal{G} = (G_1, G_2, G_3)$ from Figure 2. In order to implement OSC, a natural strategy would be to evaluate and compare $Q(G_1)$ with $Q(G_2)$ and $Q(G_2)$ with $Q(G_3)$. Since the results change in both pairs of versions, we may view it as likely they will change again between $Q(G_3)$ and the future version $Q(G_4)$. However, evaluating $Q(G_1)$, $Q(G_2)$ and $Q(G_3)$ requires maintaining the full history of changes – i.e., storing all data for $\mathcal{G} = (G_1, G_2, G_3)$ – which may be costly. Furthermore, even if we maintain a full history, if Q is a previously unseen query, we incur the runtime cost of evaluating $Q(G_1)$, $Q(G_2)$ and $Q(G_3)$. This approach is thus costly.

Alternatively, we may look at other clues for OSC that do not require evaluating $Q(G_1)$, $Q(G_2)$ and $Q(G_3)$. First, we may note that the query has four variables and two projected variables with distinct; we might expect a query with more variables (particularly projected variables) to be more sensitive to change, a query with distinct to be less sensitive to change, etc.; hence *query features* may be useful for OSC. Second, we may note that the query mentions two predicates – `rdfs:label` and `wdt:P106` whose triples are involved in changes between historical versions, which may suggest that the query is more sensitive to change; hence *predicate dynamicity features* may be useful for OSC. Finally, we may use statistical summaries of G_1, G_2 and G_3 to estimate the cardinality

[3] The prefixes used in this paper are as defined on http://prefix.cc

Legend	wdt:P27/nationality wdt:P31/instance of wdt:P106/occupation wdt:P641/sport wd:Q1050/Eswatini/Swaziland wd:Q31920/swimming wd:Q2686854/Luke Hall wd:Q3624078/sovereign state wd:Q10843402/swimmer		
Version	G_1	G_2	G_3
Triples	wd:Q1050 rdfs:label "Swaziland"@en . wd:Q1050 wdt:P31 wd:Q3624078 . wd:Q2686854 rdfs:label "Luke Hall"@en . wd:Q2686854 wdt:P27 wd:Q1050 . wd:Q2686854 wdt:P106 wd:Q10843402 .	wd:Q1050 rdfs:label "Eswatini"@en . wd:Q1050 skos:altLabel "Swaziland"@en . wd:Q1050 wdt:P31 wd:Q3624078 . wd:Q2686854 rdfs:label "Luke Hall"@en . wd:Q2686854 wdt:P27 wd:Q1050 . wd:Q2686854 wdt:P106 wd:Q10843402 .	wd:Q1050 rdfs:label "Eswatini"@en . wd:Q1050 skos:altLabel "Swaziland"@en . wd:Q1050 wdt:P31 wd:Q3624078 . wd:Q2686854 rdfs:label "Luke Hall"@en . wd:Q2686854 wdt:P27 wd:Q1050 . wd:Q2686854 wdt:P641 wd:Q31920 .
Added		wd:Q1050 rdfs:label "Eswatini"@en . wd:Q1050 skos:altLabel "Swaziland"@en .	wd:Q2686854 wdt:P641 wd:Q31920 .
Removed		wd:Q1050 rdfs:label "Swaziland"@en .	wd:Q2686854 wdt:P106 wd:Q10843402 .

Figure 2: Example of a dynamic RDF graph $\mathcal{G} = (G_1, G_2, G_3)$ with three versions based on Wikidata

of $Q(G_1)$, $Q(G_2)$ and $Q(G_3)$ (i.e., the number of results returned) without actually evaluating these queries; comparing these cardinality estimates across versions allows us to estimate a *degree of change* across versions, where if we notice major changes in these estimated cardinalities over historical versions, this may suggest that the query is more sensitive to change.

Given the diversity of such clues that may influence different aspects of OSC predictions, and the potential interactions between them, we propose to encode such clues as features and use them to train a binary classifier for computing the final prediction. □

Though our current focus is on the OSC task, an implementation for this task can also be used to address the related Time To Live (TTL) task, which predicts the next version $Q(G_k)$ in which the results are expected to change (if any). Specifically, we can address TTL by applying OSC with different gaps between versions. For example, given a query Q and a dynamic RDF dataset $\mathcal{G} = (G_1, \ldots, G_m)$, we can call OSC with $\mathcal{G}$ to predict a change for $Q(G_{m+1})$; if true, we return $m + 1$; if false, we can use $\mathcal{G} = (G_1, G_3, \ldots, G_{m-\text{mod}(m-1,2)})$, returning $m + 2$ if OSC returns true, or moving onto an interval of three if it returns false, and so forth, outputting the lowest value $k < m$ such that OSC on $\mathcal{G} = (G_1, G_{k+1}, \ldots, G_{m-\text{mod}(m-1,k)})$ and Q returns true, or returning that there is no foreseeable change if no such value for k is found.[4]

4 RELATED WORK

Before presenting our proposals for implementing OSC, we discuss works on RDF/SPARQL dynamics. We also briefly discuss works in the relational database literature.

RDF/Linked Data dynamics: refers to how RDF graphs (on the Web) change over time, and has been studied from numerous perspectives. While some works have focused on studying changes over time within a particular RDF graph [16], others have focused on changes across different datasets [12, 13, 21, 35, 48, 51], including new datasets being added or removed [21, 51]. Dynamics have been studied at the level of entities [16, 35, 51], predicates [21], schema [13], documents [12, 21, 48, 51] and entire datasets [21]. Many of these works have been observational in terms of trying to capture and analyse patterns in dynamic Linked Data content [12, 13, 21, 35, 48]. Such works have also proposed models to characterise dynamics – and ultimately predict changes – based on Poisson distributions [48], abstract schemata [13], Markov chains [53], empirical distributions [33], machine learning models [36], formal concept analysis [16], among other techniques.

Synchronisation: refers to the problem of keeping multiple clients up-to-date with remote changes on a server as efficiently (for both client and server) as possible. There are two main approaches to synchronisation. *Push-based* approaches are typically based on servers notifying clients of relevant changes [15, 24, 30, 38, 47]. Such approaches enable stronger levels of consistency, but centralise the burden of synchronisation on the server. Conversely, in *pull-based approaches*, the responsibility lies with clients to request updated data from the server [1, 12, 22, 23]. Such approaches involve weaker levels of consistency, but preserve the traditional style of client–server interaction that has enabled the Web to scale.

RDF Archives: host and enable access to historical versions of a dynamic RDF graph [5, 7, 14, 39, 46]. Such archives can support single-version queries that retrieve the answers at a particular point in time, or delta queries that retrieve answers changing between two versions [14]. Indexing techniques are often used to compress and provide efficient access over historical data [5, 14, 39, 46].

SPARQL query caching: aims to reuse the results of SPARQL (sub-)queries for subsequent requests. Numerous server-side caching techniques for SPARQL have been proposed, based on similarity measures for graph patterns [27, 45], the types of join used [25], generalised and canonicalised sub-queries [37, 56], etc. Rather than caching (sub-)query results, [57] cache frequently accessed triples. Only the works of Martin et al. [29] and Williams and Weaver [55] consider changes in the RDF graph; the former work invalidates cached results for a query when a triple matching one of its query patterns changes, while the latter work adds modification times to the RDF index to quickly detect changes. On the client-side, Knuth et al. [23] propose a service in which clients can register queries that are scheduled for refresh according to policies such as how

[4] More advanced reductions from TTL to OSC might consider applying binary search to find k; or using a voting scheme over (up to) $j - 1$ dynamic graphs with gaps of j, such as $(G_1, G_4, \ldots)$, $(G_2, G_5, \ldots)$, $(G_3, G_6, \ldots)$ for $j = 3$, etc.

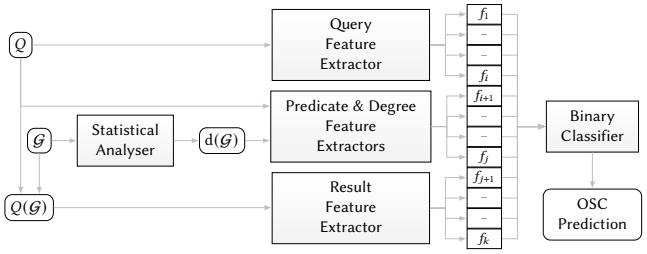

Figure 3: Proposed architecture for predicting change OSC given a query Q and dynamic RDF graph $\mathcal{G}$

long ago the last refresh took place, how long the query takes to run, how often the results change, how many results change, etc.

Caching and dynamics in relational databases: have been studied in detail, and while our focus is on SPARQL, there are relevant works in the database literature on topics such as *semantic caching* in client–server settings, where a client maintains semantic descriptions of cached data that facilitates its future reuse [8, 41]; *incremental view maintenance* [17], where a materialised view, storing the results of a query, can be updated to reflect changes in the underlying data rather than being recomputing from scratch, etc. We are not, however, aware of works from the relational literature that try to *predict* changes in query results.

Novelty: We aim to predict whether or not the results of a SPARQL query will change in the next version of a dynamic RDF graph. Previous works in this direction have been primarily based on dynamic predicate features [9, 32, 49, 50, 52] and/or historical results for the query [23, 32]. We consider these methods, and further propose novel features based on estimating the degree of change using cardinality estimations, which we show to offer an interesting trade-off, offering better predictions than other features based on statistical summaries, at a fraction of the overhead for unseen queries versus historical versions (though with less accurate predictions). Our prediction framework can be used, for example, by a server to decide for which queries to cache results, or by a client to decide when to refresh the query's results on the remote server.

5 PREDICTING OSC QUERY DYNAMICS

We now describe our approach for predicting changes in the results of a particular query. Given a dynamic RDF graph and a query, the general idea is to extract a set of features for the query relative to the dynamic RDF graph that can be used for the purposes of binary classification, predicting an answer for the OSC task.

5.1 Architecture

We present the architecture for our proposed system for predicting OSC in Figure 3. The inputs are a query Q and a dynamic RDF graph $\mathcal{G}$. The system then extracts a feature vector $(f_1, \ldots, f_k)$. from these inputs and feeds them into a pre-trained binary classifier to make the OSC prediction. The query features $(f_1, \ldots, f_i)$ are extracted online from the query itself. The predicate and degree features $(f_{i+1}, \ldots, f_j)$ are extracted from a statistical description $\mathrm{d}(\mathcal{G})$ of $\mathcal{G}$, whose details will be described later; in practice, $\mathrm{d}(\mathcal{G})$ can be computed and maintained offline (independently of the query) in an incremental manner, requiring only the two most recent versions of $\mathcal{G}$ to be updated. Finally, the results features $(f_{j+1}, \ldots, f_k)$ require as input the full historical results of Q for each version of $\mathcal{G}$ (which we denote by $Q(\mathcal{G})$); this must be computed online. The binary classifier is pre-trained over a given set of queries $\mathcal{Q}$ for which ground truths are computed over withheld versions.

5.2 Features

We now present the details of our query features, predicate features, degree-of-change features, and results features.

*Query features (*Q*):* include statistics about the query. Formally, given a query Q, we define that $\mathrm{Q}(Q) = (n_T, n_V, n'_V, n_P, \vec{o})$, where n_T denotes the number of triple patterns in Q, n_V denotes the number of variables in Q, n'_V denotes the number of projected variables in Q, n_P denotes the number of unique predicates in Q, and $\vec{o}$ is a one-hot encoded vector that denotes the SPARQL operators (`AND`, `DISTINCT`, `OPTIONAL`, `UNION`) used by Q. We hypothesise that such features may be useful for predicting dynamics, where triples with fewer triple patterns, variables, etc., will have fewer "opportunities" to be affected by changes in query results. The limitation of these features is that there is no guarantee that query dynamics will correlate with them; for example, additional triple patterns may constrain the results of a query by making it more specific, and thus making its results less likely to change. The value of query features for predicting dynamics will need to be studied empirically.

*Predicate features (*P*):* include statistics about how frequently and how many triples matching the predicates used in the query change. More formally, let $G_1 \oplus G_2 = G_1 \setminus G_2 \cup G_2 \setminus G_1$ denote the set of triples in one graph, but not in the other. Further let $\sigma_{\mathrm{P}=p}(G) = \{(x, y, z) \in G \mid p = y\}$ denote the set of triples in G using the predicate p. Now, given a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_m)$, we denote by $\Delta(\mathcal{G}, p) = \sum_{i=1}^{m-1} \frac{|\sigma_{\mathrm{P}=p}(G_1 \oplus G_2)|}{|\sigma_{\mathrm{P}=p}(G_1 \cup G_2)|}$ the sum of the ratios of triples for p that changed between each consecutive pair of versions, such that the higher the value for $\Delta(\mathcal{G}, p)$, the more dynamic the triples associated with the predicate. Next let $\mathrm{preds}(Q)$ denote the set of IRIs used as predicates in Q. Now given a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_m)$, we define $\mathrm{P}(\mathcal{G}, Q) = \frac{\sum_{p \in \mathrm{preds}(Q)} \Delta(\mathcal{G}, p)}{|\mathrm{preds}(Q)|}$, taking the mean value of $\Delta(\mathcal{G}, p)$ for all predicates in Q. The benefit of predicate features is that they require lightweight statistics about the dynamic RDF graph since the number of unique predicates – even in large RDF graphs – tends to be relatively small. The limitation of these features is that they take into account few details of the query; for example, a query searching for all labels and a query searching for the label of a specific subject will not be distinguished.

*Degree-of-change features (*D*):* include statistics about the variability in the number of results returned by the query across the historical versions. Specifically, given a query Q and an RDF graph G, we denote by $\mathrm{card}(\cdot, \cdot)$ a *cardinality estimation function*, where $\mathrm{card}(Q, G)$ estimates the (bag) cardinality of $|Q(G)|$. Any such function can be used; we describe the one we use in Section 5.4. We use the function $\mathrm{card}(Q, G)$ to estimate the dynamics of a query by comparing the number of results for the query that depend on data that did not change between two versions, versus the number of

results generated over the union of the two versions. Specifically, given a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_m)$, we use the sum of the ratios $\mathrm{D}(\mathcal{G}, Q) = \sum_{i=1}^{m-1} 1 - \frac{\mathrm{card}(Q, G_i \cap G_{i+1})}{\mathrm{card}(Q, G_i \cup G_{i+1})}$ to estimate the degree-of-change in terms of what ratio of results for the query are sensitive to changes between pairs of versions, where low values indicate that fewer results are affected by the changes, while high values indicate that more results are affected by the changes.[5] When compared with query features, cardinality features take into account changes in the graph. When compared with predicate features, degree features take into account more details of the query, but require additional statistics to estimate cardinalities.

*Results features (*R*):* include statistics about the historical results for the query. These features are conceptually the simplest, where we simply count the number of times the results for the query Q changed between two consecutive historical versions of the graph. Formally, given a query Q and a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_m)$ we define this feature as follows: $\mathrm{R}(\mathcal{G}, Q) = |\{i \in \{1, \ldots, m-1\} : Q(G_i) \neq Q(G_{i+1})\}|$. Though it is the simplest, we expect that it will also provide the most useful information for OSC prediction. Conversely, as aforementioned, it incurs by far the highest overhead for an unseen query Q, requiring maintaining all data for $G_1, \ldots, G_m$ and evaluating $Q(G_1), \ldots Q(G_m)$.

5.3 Dynamic graph description

In the case of query features, we do not require knowledge of the dynamic graph. In the case of predicate and degree-of-change features, while we require some knowledge of the dynamic graph, we can also compute these features from a dynamic graph description that provides key statistics relating to a graph.

Specifically, given an RDF graph G, we define its key statistics as a tuple $\mathrm{s}(G) = (n_T, n_S, n_P, n_O)$ where $n_T = |G|$ denotes the number of triples in G, $n_S \coloneqq |\{x : \exists y, z\ (x, y, z) \in G\}|$ denotes the number of unique subjects in G, $n_P \coloneqq |\{y : \exists x, z\ (x, y, z) \in G\}|$ denotes the number of unique predicates in G, and $n_O \coloneqq |\{z : \exists x, y\ (x, y, z) \in G\}|$ denotes the number of unique objects in G. By $\mathrm{d}(G) = (\mathrm{s}(G), \{(p_1, \mathrm{s}(\sigma_{\mathrm{P}=p_1}(G))), \ldots, (p_n, \mathrm{s}(\sigma_{\mathrm{P}=p_k}(G)))\})$ we denote the description of G, consisting of the key statistics of G, and the key statistics for the partitions of G by predicate, where $\mathrm{preds}(G) = \{p_1, \ldots, p_k\}$.[6] For brevity, we will write $n_T[p]$, $n_S[p]$ and $n_O[p]$ to denote the values for n_T, n_S and n_O in $\mathrm{s}(\sigma_{\mathrm{P}=p}(G))$, i.e., the number of unique triples, subject and objects, respectively for the sub-graph of G with predicate p; we will similarly write n_T, n_S and n_O to more concisely denote the values in $\mathrm{s}(G)$.

We can now generalise this idea to a dynamic graph description. Given a dynamic graph $\mathcal{G} = (G_1, \ldots, G_m)$, we compute $6(m-1)$ graph descriptions, or in other words two graph descriptions for each adjacent pair of graphs G_i, G_{i+1}, specifically, $\mathrm{d}(G_i \cap G_{i+1})$, $\mathrm{d}(G_i \oplus G_{i+1})$, $\mathrm{d}(G_i \cup G_{i+1})$. Rather than describing each individual graph, this allows us to capture and compare statistics about the triples that stayed the same in both versions, that changed between versions, and that appeared in either version, respectively. A dynamic graph description $\mathrm{d}(\mathcal{G})$ is then a 3-tuple of the form:

$$\begin{aligned}\mathrm{d}(\mathcal{G}) = ((&\mathrm{d}(G_1 \cap G_2), \ldots, \mathrm{d}(G_{m-1} \cap G_m))\\ &(\mathrm{d}(G_1 \oplus G_2), \ldots, \mathrm{d}(G_{m-1} \oplus G_m))\\ &(\mathrm{d}(G_1 \cup G_2), \ldots, \mathrm{d}(G_{m-1} \cup G_m)))\end{aligned}$$

The predicate features described earlier are then trivial to compute from $\mathrm{d}(\mathcal{G})$. This leaves us to discuss the estimations of cardinalities needed for the degree-of-change features.

Table 1: Cardinality estimation for a single triple pattern

t	$\mathrm{card}(t, \mathrm{d}(G))$
(x, y, z)	1
(X, y, z)	$n_T[y]/n_O[y]$
(x, Y, z)	1
(x, y, Z)	$n_T[y]/n_S[y]$
(X, Y, z)	n_T/n_O
(X, y, Z)	$n_T[y]$
(x, Y, Z)	n_T/n_S
(X, Y, Z)	n_T

5.4 Cardinality estimations

Any cardinality estimation function $\mathrm{card}(\cdot, \cdot)$ can be used within our framework, where the more accurate the function is in terms of predicting the actual number of results returned by Q over G, the better the results we would expect for predicting dynamics. An obvious implementation is to use $\mathrm{card}(Q, G) = |Q(G)|$, i.e., to evaluate the query on the graph and count the results. However, this is akin to using historical query results, and will be costly. We instead define a function $\mathrm{card}(\cdot, \cdot)$ that uses (only) our dynamic-graph description. Specifically, given an RDF graph G, we use $\mathrm{d}(G)$ to estimate the cardinality of triple patterns per standard cardinality estimations used by relational databases (e.g., by Postgres[7]), considering each predicate as a binary relation. We adopt typical assumptions in these scenarios, such as the assumption that every join is complete, and that (subject/object) values are uniformly distributed.

In the following let $X, Y, Z \in \mathbf{V}$ and $x, y, z \in \mathbf{I} \cup \mathbf{L}$. We begin by estimating the cardinality of a triple pattern t as shown in Table 1 following the aforementioned assumptions. Note that only in the case of (X, y, Z) and (X, Y, Z) will the exact cardinality be known. Otherwise we assume that when subject and object are constant, that the cardinality is 1; and that when one of the subject or object is constant, that the cardinality is a ratio of the total number of triples in the corresponding graph divided by the number of unique subject/object terms in the graph (uniform distribution).

However, given a particular predicate, the distribution of subject and object values may also be far from the uniform distribution. For example, a country such as India may appear much more often as the object of the predicate *birthPlace* than a country such as Tuvalu. We thus extend $\mathrm{s}(G)$ in order to store the k most common subject

[5] While it may be simpler – and thus tempting – to rather compare $\mathrm{card}(Q, G_i)$ and $\mathrm{card}(Q, G_{i+1})$, they may generate the same number of results even though the results themselves change. This issue would arise, for example, if we were to query for the objects of `rdfs:label` over Figure 2, where the number of results per version remains precisely the same even though the results change between G_1 and G_2.

[6] Note that $n_P = 1$ for any $\mathrm{s}(\sigma_{\mathrm{P}=p_i}(G))$ with $1 \leq i \leq k$, and is not stored.

[7] https://www.postgresql.org/docs/current/static/planner-stats-details.html

and object values and their frequencies, both for the full graph and its predicate partitions. Thus if we have a pattern (x, y, Z), where x is among the top-k most common subjects for the sub-graph with predicate y, we can compute the exact cardinality. The estimations for non top-k values are also adjusted accordingly to subtract the cardinalities of the top-k values from the calculations.

The estimations for other query operators are then layered on top of the estimates for triple patterns in a standard way. Specifically, for each bag of solution mappings produced at each stage, we maintain an estimated cardinality for the entire bag, as well as an estimated number of unique values for each variable. When we compute the cardinality estimate for a join $M_1 \bowtie M_2$ on the variable V, if we estimate that V takes v_1 unique values in M_1 and v_2 unique values in M_2, then we assume that V will have $\min\{v_1, v_2\}$ unique values in $M_1 \bowtie M_2$ based on the assumption of complete joins. Without loss of generality, if we assume that $v_1 \leq v_2$, then the estimated number of unique values for (other) variables in M_2 will be reduced by a factor of $\frac{v_1}{v_2}$. Finally, the number of mappings in $M_1 \bowtie M_2$ will be computed as $\frac{m_1 m_2}{v_2}$, where m_1 and m_2 denote the estimated cardinalities for M_1 and M_2 respectively. We follow a similar process for estimating the cardinalities of the left-join $M_1 \bowtie M_2$, where we do not lower the estimates for unique values of variables in M_1. For the union $M_1 \cup M_2$, we sum (under bag semantics) the estimates for M_1 and M_2 and sum the estimates of unique values per variable (effectively assuming all values to be distinct in M_1 and M_2). For projection combined with distinct, the overall cardinality is given by the maximum number of unique values for an individual projected variable, with each variable maintaining its estimate.

For computing the degree-of-change features, the dynamic-graph description $\mathrm{d}(\mathcal{G})$ contains the individual graph descriptions needed to compute $\mathrm{card}(Q, G_i \cap G_{i+1})$ and $\mathrm{card}(Q, G_i \cap G_{i+1})$, namely $\mathrm{d}(G_i \cap G_{i+1})$ and $\mathrm{d}(G_i \cap G_{i+1})$ for $1 \leq i < m$.

5.5 Classification

In order to make the final OSC prediction for Q and $\mathcal{G}$, we use a binary classifier trained over example queries. Specifically, given a dynamic RDF graph $\mathcal{G} = (G_1, \ldots, G_m)$ and a set of training queries $\mathbf{Q}$, we use a sub-sequence of versions $\mathcal{G}' = (G_i, \ldots, G_j)$ (where $1 \leq i < j < m$) in order to build a feature set for each query $Q \in \mathbf{Q}$, and evaluate $Q(G_j) = Q(G_{j+1})$ in order to label the query. Given an unseen query Q', we can then build the feature set with respect to $\mathcal{G}$ (or any sub-sequence thereof if we do not wish to use all versions), and use the trained classifier to predict whether or not the results for Q' will change for the next version of the graph G_{m+1}.

5.6 Complexity

With respect to the trade-off mentioned in Figure 1, we now analyse the complexity of extracting the different sets of features for an unseen query Q and dynamic graph $\mathcal{G}$ in order to draw insights about the overheads involved. We assume here that counting n elements takes $O(n)$ time. For the query Q, we denote by q the number of triple patterns it contains, and for a dynamic graph $\mathcal{G}$, we denote by g the sum of the number of triples in each graph it contains. For extracting query features, we only require statistics over Q, which can be counted in $O(q)$. For predicate features and degree-of-change features, we need to extract statistics over Q and $\mathcal{G}$, which can be extracted in $O(q + g)$; under the natural assumption that $g \gg q$, we could simplify this to $O(g)$; furthermore, in practice, the counting is pre-computed and the counts indexed, so the amortised complexity approximates $O(q)$. For the results features, evaluating each $Q(G_i)$ is intractable (even using worst-case algorithms, the complexity to enumerate $Q(G_i)$ is given by the AGM bound [4]). We note that training the classifier is similarly intractable in all cases as it requires evaluating the training queries; however, the training process can take place offline.

In summary, when extracting features online to predict OSC for an unseen query, query features have the least overhead; predicate features and degree-of-change features have slightly more overhead, but assuming that statistics are precomputed and indexed, the amortized overhead is similar to that for query features; finally, extracting results features is intractable due to having evaluate the query at runtime over historical versions, which is a significant leap in terms of complexity compared to the other alternatives.

6 EVALUATION

While the previous theoretical analysis establishes a comparison in terms of overhead, the accuracy of the OSC features for different types of features must be established empirically. We thus now evaluate our proposal for estimating the dynamics of queries per the different sets of features proposed in the previous sections. Our evaluation aims to ascertain the quality of predictions (in terms of precision, recall and F_1-score) as to whether the results for a query will change in the next version considering these features. We begin by describing the two datasets used.

Datasets. We first consider a dataset of 17 weekly truthy versions of Wikidata, spanning from 2019/11/14 (containing 4.4 billion triples) to 2020/03/05 (containing 4.8 billion triples). The number of triples grew by 9.2% during the time period, where approximately 4 times more triples are added, on average, than removed. A total of 78 billion triples are present in all versions. We use the 141 queries previously generated by Moya and Hogan [32], which were sourced from the user-contributed example queries published by the Wikidata query service. Of these queries, the results for 56 change between each pair of versions, while the results for 20 queries never change, with the remaining queries changing intermittently.

We consider a second dataset based on DBpedia Live, where we use the changesets to build 18 daily versions, spanning from 2019-07-01 (containing 593.6 million triples), until 2019-07-18 (containing 590.3 million triples). In this case we see more deletions than insertions, and fewer changes overall, when compared with Wikidata. This is to be expected considering that the interval between versions is 7 times shorter in the case of DBpedia. Overall, the DBpedia versions contain 10.7 billion triples. We use the set of 10,000 queries extracted by Knuth et al. [23] from the LSQ dataset [42], composed of queries evaluated by the DBpedia SPARQL endpoint. We filtered these queries to remove those that returned empty results over all versions. However, given the origin of these queries, we noted a very high number of very similar queries, which we assume are due to applications using the endpoints. For example, we found 2,425 queries asking for the label, latitude and longitude of various

resources; 722 asking for the abstract of a resource; 575 asking for the type of a resource; etc. To avoid the results being dominated by such queries, we partitioned the queries by their predicates, and applied a logarithmic downsampling for the number of queries in each partition such that there were no more than 20 queries in each partition. The end result was 256 queries, where there were 416 changes in results between pairs of versions (out of a possible 4,318 comparisons). Again we note fewer changes than in the case of Wikidata, likely due to the narrower interval between versions.

In order to extract features from the different versions, we used Unix sorts and custom Java code for counting. In order to evaluate queries for producing ground truths and results features, we loaded the corresponding graphs into Virtuoso instances.

Binary classification models. We experiment with a number of well-known machine learning classifiers, including Decision Trees, Naive Bayes, Nearest Neighbours and Linear SVM. We also include a Random Baseline for comparison. We train and compare classifiers for different window sizes (specifically 3, 5, 9, corresponding to 2, 4 and 8 pairs of consecutive versions).

Results. In Tables 2 and 3, we present the results for predicting one-shot changes in queries for the Wikidata and DBpedia datasets, respectively. For reasons of space, we include only F_1 scores, where we refer to the accompanying online material for precision and recall results [2]. In these tables, we experiment with four well-known classifiers, as well as a random baseline for comparison. We compare the results for six feature sets: query (Q), property (P), degree-of-change (D), results (R), all features without historical results (QPD) and all features (QPDR).

Comparing models, we see that Nearest Neighbours (NN) offers the best performance overall across different feature sets and datasets, being slightly outperformed by other models in few cases.

Comparing window sizes, we see that the best results overall are given for larger window sizes. This can be explained by the fact that variances of individual versions are smoothed out when longer intervals are considered. However, it is interesting to note that the performance improvement is not very pronounced, where smaller window sizes provide similar prediction quality.

Comparing different feature sets, we see that statistics based on historical query results (R) are the most important, and enable (by far) the most accurate predictions. In fact, the predictions with just the results features even tend to outperform the predictions combining all features. On the other hand, we find that query features and predicate features provide minimal contribution to prediction quality, where the best models are only slightly better than the random baseline. In the case of Wikidata, degree-of-change statistics tend to offer predictions when combined with the Nearest Neighbours classifier. However, the quality of predictions drops for DBpedia, which we believe to be due to the relative sparsity of changes in the query results of the dataset.

Table 2: F_1-score for OSC on the Wikidata dataset considering different features sets and window sizes (w)

	Classifier	Q	P	D	R	QPD	QPDR
w=3	*Random Baseline*	0.509	0.499	0.497	0.482	0.496	0.5
	Decision Trees	0.549	0.554	0.66	0.855	0.61	0.709
	Naive Bayes	0.522	0.36	0.478	0.825	0.498	0.67
	Nearest Neighbours	0.547	0.532	0.691	0.837	0.537	0.83
	Linear SVM	0.582	0.441	0.387	0.825	0.603	0.827
	Best	**0.582**	**0.554**	**0.691**	**0.855**	**0.61**	**0.83**
w=5	*Random Baseline*	0.487	0.482	0.495	0.501	0.515	0.504
	Decision Trees	0.539	0.541	0.677	0.868	0.622	0.735
	Naive Bayes	0.516	0.382	0.484	0.841	0.513	0.76
	Nearest Neighbours	0.54	0.536	0.656	0.837	0.524	0.851
	Linear SVM	0.594	0.442	0.472	0.841	0.612	0.838
	Best	**0.594**	**0.541**	**0.677**	**0.868**	**0.622**	**0.851**
w=9	*Random Baseline*	0.497	0.501	0.499	0.517	0.487	0.508
	Decision Trees	0.544	0.568	0.634	0.876	0.631	0.699
	Naive Bayes	0.53	0.464	0.498	0.859	0.528	0.781
	Nearest Neighbours	0.551	0.568	0.675	0.868	0.521	0.863
	Linear SVM	0.595	0.445	0.546	0.86	0.626	0.85
	Best	**0.595**	**0.568**	**0.675**	**0.876**	**0.631**	**0.863**

Table 3: F_1-score for OSC on the DBpedia dataset considering different feature sets and window sizes (w)

	Classifier	Q	P	D	R	QPD	QPDR
w=3	*Random Baseline*	0.494	0.498	0.501	0.503	0.506	0.511
	Decision Trees	0.57	0.488	0.556	0.928	0.557	0.879
	Naive Bayes	0.475	0.532	0.244	0.902	0.279	0.774
	Nearest Neighbours	0.519	0.474	0.479	0.928	0.479	0.904
	Linear SVM	0.474	0.474	0.48	0.928	0.496	0.925
	Best	**0.57**	**0.532**	**0.556**	**0.928**	**0.557**	**0.925**
w=5	*Random Baseline*	0.494	0.492	0.514	0.507	0.494	0.489
	Decision Trees	0.571	0.497	0.555	0.938	0.536	0.895
	Naive Bayes	0.475	0.51	0.407	0.936	0.429	0.837
	Nearest Neighbours	0.48	0.485	0.55	0.936	0.499	0.912
	Linear SVM	0.475	0.475	0.475	0.938	0.491	0.937
	Best	**0.571**	**0.51**	**0.555**	**0.938**	**0.536**	**0.937**
w=9	*Random Baseline*	0.505	0.481	0.501	0.509	0.484	0.505
	Decision Trees	0.509	0.502	0.578	0.946	0.554	0.884
	Naive Bayes	0.473	0.507	0.392	0.931	0.428	0.848
	Nearest Neighbours	0.474	0.503	0.508	0.942	0.476	0.914
	Linear SVM	0.475	0.475	0.475	0.94	0.489	0.94
	Best	**0.509**	**0.507**	**0.578**	**0.946**	**0.554**	**0.94**

7 CONCLUSIONS

In this paper, we have proposed a framework for predicting whether or not the results of a query will change in the next version. We propose four categories of features for this task, based on queries, predicates, degree-of-change estimates and historical results. In terms of overhead, while the complexity of the later three categories is quite similar in terms of extracting features online, the fourth category is intractable. On the other hand, taking into account the quality of predictions, features based on historical results provide by far the best predictions. In the case of unseen queries, where the historical results are costly to compute, degree-of-change estimates provide the next best alternative. From these results we see a clear trade-off in terms of the overhead of prediction versus the quality of those predictions. In terms of future work, it would be of interest to explore further features, and also to explore TTL predictions.

We refer to the online material for the datasets and queries used, along with additional results [2].

REFERENCES

[1] Usman Akhtar, Muhammad Bilal Amin, and Sungyoung Lee. 2017. Evaluating scheduling strategies in LOD based application. In *Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 255–258.

[2] Anonymous. 2021. Online material. GitHub Repository. https://github.com/iswc2021/querydynamics.github.io.

[3] Carlos Buil Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. 2013. SPARQL Web-Querying Infrastructure: Ready for Action?. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8219)*. Springer, 277–293. https://doi.org/10.1007/978-3-642-41338-4_18

[4] Albert Atserias, Martin Grohe, and Dániel Marx. 2013. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767. https://doi.org/10.1137/110859440

[5] Afef Bahri, Meriem Laajimi, and Nadia Yacoubi Ayadi. 2018. Distributed RDF Archives Querying with Spark. In *European Semantic Web Conference (ESWC)*. 451–465.

[6] Tim Berners-Lee. 2006. Linked Data. W3C Design Issues. From https://www.w3.org/DesignIssues/LinkedData.html; retr. 2010/10/27.

[7] Ignacio Cuevas and Aidan Hogan. 2020. Versioned Queries over RDF Archives: All You Need is SPARQL?. In *Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW) (CEUR Workshop Proceedings, Vol. 2821)*, Fabrizio Orlandi, Damien Graux, Maria-Esther Vidal, Javier D. Fernández, and Jeremy Debattista (Eds.). CEUR-WS.org, 43–52. http://ceur-ws.org/Vol-2821/paper6.pdf

[8] Shaul Dar, Michael J. Franklin, Björn Þór Jónsson, Divesh Srivastava, and Michael Tan. 1996. Semantic Data Caching and Replacement. In *International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 330–341.

[9] Soheila Dehghanzadeh, Josiane Xavier Parreira, Marcel Karnstedt, Jürgen Umbrich, Manfred Hauswirth, and Stefan Decker. 2014. Optimizing SPARQL Query Processing on Dynamic and Static Data Based on Query Time/Freshness Requirements Using Materialization. In *Semantic Technology - 4th Joint International Conference, JIST 2014, Chiang Mai, Thailand, November 9-11, 2014. Revised Selected Papers*. Springer, 257–270.

[10] Renata Queiroz Dividino, Thomas Gottron, and Ansgar Scherp. 2015. Strategies for Efficiently Keeping Local Linked Open Data Caches Up-To-Date. In *International Semantic Web Conference (ISWC)*. Springer, 356–373.

[11] Renata Queiroz Dividino, Thomas Gottron, Ansgar Scherp, and Gerd Gröner. 2014. From Changes to Dynamics: Dynamics Analysis of Linked Open Data Sources. In *Proc. of the 1st International Workshop on Dataset PROFIling & fEderated Search for Linked Data co-located with the 11th Extended Semantic Web Conference, PROFILES@ESWC 2014, Anissaras, Crete, Greece, May 26, 2014*.

[12] Renata Queiroz Dividino, André Kramer, and Thomas Gottron. 2014. An Investigation of HTTP Header Information for Detecting Changes of Linked Open Data Sources. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*. Springer, 199–203.

[13] Renata Queiroz Dividino, Ansgar Scherp, Gerd Gröner, and Thomas Grotton. 2013. Change-a-LOD: Does the Schema on the Linked Data Cloud Change or Not?. In *Proc. of the Fourth International Workshop on Consuming Linked Data, COLD 2013, Sydney, Australia, October 22, 2013*. CEUR-WS.org.

[14] Javier D. Fernández, Axel Polleres, and Jürgen Umbrich. 2015. Towards Efficient Archiving of Dynamic Linked Open Data. In *DIACHRON Managing the Evolution and Preservation of the Data Web*. 34–49.

[15] Julien Genestoux, Brad Fitzpatrick, Brett Slatkin, and Martin Atkins. 2018. WebSub. W3C Recommendation. https://www.w3.org/TR/websub/.

[16] Larry González and Aidan Hogan. 2018. Modelling Dynamics in Semantic Web Knowledge Graphs with Formal Concept Analysis. In *Proc. of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. ACM, 1175–1184.

[17] Ashish Gupta and Inderpal Singh Mumick. 1995. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.* 18, 2 (1995), 3–18.

[18] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. SPARQL 1.1 Query Language. W3C Recommendation. https://www.w3.org/TR/sparql11-query/.

[19] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.* 194 (2013), 28–61.

[20] Aidan Hogan. 2015. Skolemising Blank Nodes while Preserving Isomorphism. In *Proc. of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*. ACM, 430–440.

[21] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. 2013. Observing Linked Data Dynamics. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*. Springer, 213–227.

[22] Kjetil Kjernsmo. 2015. A Survey of HTTP Caching Implementations on the Open Semantic Web. In *European Semantic Web Conference (ESWC)*. Springer, 286–301.

[23] Magnus Knuth, Olaf Hartig, and Harald Sack. 2016. Scheduling Refresh Queries for Keeping Results from a SPARQL Endpoint Up-to-Date (Short Paper). In *On the Move to Meaningful Internet Systems (OTM)*. Springer, 780–791.

[24] Magnus Knuth, Dinesh Reddy, Anastasia Dimou, Sahar Vahdati, and George Kastrinakis. 2015. Towards Linked Data Update Notifications Reviewing and Generalizing the SparqlPuSH Approach. In *Workshop on Negative or Inconclusive Results in Semantic Web (NoISE) (CEUR Workshop Proceedings, Vol. 1435)*, Anastasia Dimou, Jacco van Ossenbruggen, Miel Vander Sande, and Maria-Esther Vidal (Eds.). CEUR-WS.org.

[25] Tomas Lampo, Maria-Esther Vidal, Juan Danilow, and Edna Ruckhaus. 2011. To Cache or Not To Cache: The Effects of Warming Cache in Complex SPARQL Queries. In *On the Move to Meaningful Internet Systems (OTM)*. Springer, 716–733.

[26] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[27] Johannes Lorey and Felix Naumann. 2013. Caching and Prefetching Strategies for SPARQL Queries. In *ESWC Satellite Events*. Springer, 46–65.

[28] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. 2018. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II*. Springer, 376–394.

[29] Michael Martin, Jörg Unbehauen, and Sören Auer. 2010. Improving the Performance of Semantic Web Applications with SPARQL Query Caching. In *Extended Semantic Web Conference (ESWC)*. Springer, 304–318.

[30] Paolo Missier, Pinar Alper, Óscar Corcho, Ian Dunlop, and Carole A. Goble. 2007. Requirements and Services for Metadata Management. *IEEE Internet Computing* 11, 5 (2007), 17–25.

[31] Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, and Sebastian Hellmann. 2012. DBpedia and the live extraction of structured data from Wikipedia. *Program* 46, 2 (2012), 157–181.

[32] Alberto Moya Loustaunau and Aidan Hogan. 2019. Estimating the Dynamics of SPARQL Query Results Using Binary Classification. In *Workshop on Querying and Benchmarking the Web of Data (QuWeDa)*. 5–20.

[33] Sebastian Neumaier and Jürgen Umbrich. 2016. Measures for Assessing the Data Freshness in Open Data Portals. In *2nd International Conference on Open and Big Data, OBD 2016, Vienna, Austria, August 22-24, 2016*. IEEE Computer Society, 17–24.

[34] Chifumi Nishioka and Ansgar Scherp. 2015. Temporal Patterns and Periodicity of Entity Dynamics in the Linked Open Data Cloud. In *Proc. of the 8th International Conference on Knowledge Capture, K-CAP 2015, Palisades, NY, USA, October 7-10, 2015*. ACM, 22:1–22:4.

[35] Chifumi Nishioka and Ansgar Scherp. 2016. Information-theoretic Analysis of Entity Dynamics on the Linked Open Data Cloud. In *Proc. of the 3rd International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES '16) co-located with the 13th ESWC 2016 Conference, Anissaras, Greece, May 30, 2016*. CEUR-WS.org.

[36] Chifumi Nishioka and Ansgar Scherp. 2017. Keeping Linked Open Data caches up-to-date by predicting the life-time of RDF triples. In *Proc. of the International Conference on Web Intelligence, Leipzig, Germany, August 23-26, 2017*. ACM, 73–80.

[37] Nikolaos Papailiou, Dimitrios Tsoumakos, Panagiotis Karras, and Nectarios Koziris. 2015. Graph-Aware, Workload-Adaptive SPARQL Query Caching. In *SIGMOD International Conference on Management of Data*. ACM, 1777–1792.

[38] Alexandre Passant and Pablo N. Mendes. 2010. sparqlPuSH: Proactive Notification of Data Updates in RDF Stores Using PubSubHubbub. In *Proc. of the Sixth Workshop on Scripting and Development for the Semantic Web, Crete, Greece, May 31, 2010*. CEUR-WS.org.

[39] Olivier Pelgrin, Luis Galárraga, and Katja Hose. 2021. Towards Fully-fledged Archiving for RDF Datasets. *Semantic Web Journal* (2021). http://www.semantic-web-journal.net/content/towards-fully-fledged-archiving-rdf-datasets-0 In Press.

[40] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009), 16:1–16:45.

[41] Qun Ren, Margaret H. Dunham, and Vijay Kumar. 2003. Semantic Caching and Query Processing. *IEEE Trans. Knowl. Data Eng.* 15, 1 (2003), 192–210.

[42] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. LSQ: The Linked SPARQL Queries Dataset. In *International Semantic Web Conference (ISWC)*. Springer, 261–269.

[43] Guus Schreiber and Yves Raimond. 2014. RDF 1.1 Primer. W3C Working Group Note. https://www.w3.org/TR/rdf11-primer/.

[44] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. 2012. LinkedGeoData: A core for a web of spatial open data. *Semantic Web* 3, 4 (2012), 333–354.

[45] Heiner Stuckenschmidt. 2004. Similarity-Based Query Caching. In *International Conference on Flexible Query Answering Systems (FQAS)*. Springer, 295–306.

[46] Ruben Taelman, Miel Vander Sande, Joachim Van Herwegen, Erik Mannens, and Ruben Verborgh. 2019. Triple storage for random-access versioned querying of

RDF archives. *J. Web Semant.* 54 (2019), 4–28.

[47] Sebastian Tramp, Philipp Frischmuth, Timofey Ermilov, and Sören Auer. 2010. Weaving a Social Data Web with Semantic Pingback. In *Knowledge Engineering and Management by the Masses - 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*. Springer, 135–149.

[48] Jürgen Umbrich, Michael Hausenblas, Aidan Hogan, Axel Polleres, and Stefan Decker. 2010. Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources. In *Proc. of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*. CEUR-WS.org.

[49] Jürgen Umbrich, Marcel Karnstedt, Aidan Hogan, and Josiane Xavier Parreira. 2012. Freshening up while Staying Fast: Towards Hybrid SPARQL Queries. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*. Springer, 164–174.

[50] Jürgen Umbrich, Marcel Karnstedt, Aidan Hogan, and Josiane Xavier Parreira. 2012. Hybrid SPARQL Queries: Fresh vs. Fast Results. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*. Springer, 608–624.

[51] Jürgen Umbrich, Marcel Karnstedt, and Sebastian Land. 2010. Towards Understanding the Changing Web: Mining the Dynamics of Linked-Data Sources and Entities. In *LWA 2010 - Lernen, Wissen & Adaptivität, Workshop Proceedings, Kassel, 4.-6. Oktober 2010*. 159–162.

[52] Jürgen Umbrich, Marcel Karnstedt, Josiane Xavier Parreira, Axel Polleres, and Manfred Hauswirth. 2012. Linked Data and Live Querying for Enabling Support Platforms for Web Dataspaces. In *Workshops Proc. of the IEEE 28th International Conference on Data Engineering, ICDE 2012, Arlington, VA, USA, April 1-5, 2012*. IEEE Computer Society, 23–28.

[53] Jürgen Umbrich, Nina Mrzelj, and Axel Polleres. 2015. Towards capturing and preserving changes on the Web of Data. In *Proc. of the First DIACHRON Workshop on Managing the Evolution and Preservation of the Data Web co-located with 12th European Semantic Web Conference (ESWC 2015), Portorož, Slovenia, May 31, 2015*. 50–65.

[54] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[55] Gregory Todd Williams and Jesse Weaver. 2011. Enabling Fine-Grained HTTP Caching of SPARQL Query Results. In *International Semantic Web Conference (ISWC)*. Springer, 762–777.

[56] Gang Wu and Mengdong Yang. 2012. Improving SPARQL query performance with algebraic expression tree based caching and entity caching. *J. Zhejiang Univ. Sci. C* 13, 4 (2012), 281–294.

[57] Wei Emma Zhang, Quan Z. Sheng, Kerry Taylor, and Yongrui Qin. 2015. Identifying and Caching Hot Triples for Efficient RDF Query Processing. In *Database Systems for Advanced Applications (DASFAA)*. Springer, 259–274.