Noname manuscript No.
(will be inserted by the editor)

Learning-based SPARQL Query Performance Modelling and Prediction

Wei Emma Zhang · Quan Z. Sheng · Yongrui Qin · Kerry Taylor · Lina Yao

Received: date / Accepted: date

Abstract One of the challenges of managing an RDF database is predicting performance of SPARQL queries before they are executed. Performance characteristics, such as the execution time and memory usage, can help data consumers identify unexpected long-running queries before they start and estimate the system workload for query scheduling. Queries can be rewritten to reduce time cost or rescheduled when the resource is not in contention. Extensive works address such issues in traditional SQL queries but they are not directly applicable when dealing with SPARQL queries. Further, the effort exploiting machine learning techniques is limited. In this paper, we adopt machine learning techniques to predict the performance of SPARQL queries. Our work focuses on modelling features of a SPARQL query to a vector representation and use these feature vectors to train predictive models. This method does not depend on underlying systems and any knowledge of the underlying data, but only on the nature of SPARQL queries. We adopt multiple regression models as prediction models and propose an one-step and an two-step prediction processes. Query performances in both cold and warm stages are studied. Evaluations are performed on real world SPRAQL queries, whose execution time ranges from milliseconds to hours. The results demonstrate

Wei Emma Zhang
School of Computer Science, The University of Adelaide, Australia
E-mail: wei.zhang01@adelaide.edu.au

Quan Z. Sheng
Department of Computing, Macquarie University, Australia
E-mail: michael.sheng@mq.edu.au

Yongrui Qin
School of Computing and Engineering, University of Huddersfield, United Kingdom
E-mail: y.qin2@hud.ac.uk

Kerry Taylor
Research School of Computer Science, Australian National University, Australia
E-mail: kerry.taylor@anu.edu.au

Lina Yao
School of Computer Science and Engineering, The University of New South Wales, Australia
E-mail: lina.yao@unsw.edu.au

that the proposed approach can effectively predict SPARQL query performance and outperforms state-of-the-art approaches.

Keywords SPARQ · Feature modelling · Prediction · Query performance

1 Introduction

The Semantic Web, with its underlying data model Resource Description Framework (RDF) and its query language SPARQL Protocol and RDF Query Language (SPARQL), has received increasing attention among researchers and data consumers in both academia and industry. RDF essentially represents data as a set of three-attribute tuples, i.e., triples. The attributes are *subject*, *predicate* and *object*, where *predicate* is the relationship between *subject* and *object*. Over the recent years, RDF has been increasingly used as a general data model for conceptual description and information modelling. For example, knowledge management applications such as DBpedia[1] and Freebase[2] offer large collections of facts about entities and their relations with RDF-based representations. Domain knowledge bases provide biology resources (e.g., UniProt[3], BioPortal[4]) and spatial data (e.g., LinkedGeoData[5]). Since the number of publicly available RDF datasets and their volume grow dramatically, it becomes essential for efficient querying of large scale RDF datasets. This is an important issue in the sense that whether we can obtain knowledge efficiently affects the adoption of RDF data as well as the underlying Semantic Web technologies.

1.1 Motivation

Substantial works focus on the prediction of query performance (e.g., execution time) [1,6,14,24]. Prediction of query execution performance can benefit many system management decisions, including:

- *Workload Management*: Predicting the execution performance accurately before executing incoming queries can help estimate workloads and effectively arrange available resources.
- *Query Scheduling*: Understanding the query performance metric of an incoming query can help decide whether and when to run the query to avoid system hanging. The long-running query can be rewritten in order to improve performance.
- *System Sizing*: The sizing of systems (e.g., CPU, Memory etc.) is dependent on the peak value of resources required to complete unforeseen queries.
- *Capacity Planning*: Given an expected change to a workload, the decision on whether to upgrade the system for required resources depends on the accurate estimation of query execution performance.

[1] http://dbpedia.org/
[2] https://www.freebase.com/
[3] http://www.uniprot.org/
[4] http://bioportal.bioontology.org/
[5] http://linkedgeodata.org/

Studies show that cost model based query optimizers are insufficient for query performance prediction [2,7]. Therefore, approaches that exploit machine learning techniques to build predictive models have been proposed [2,7,22]. These approaches treat the database system as a black box and focus on learning query performance prediction models, which are evaluated as feasible and effective [2]. These works extract the features of queries by exploring the query plan that can provide estimated values such as estimated execution time, estimated row count and these two estimations for each query operator (e.g., AND).

For SPARQL queries, the query engines can be grouped into two categories: RDBMS-based and RDF native triple stores. RDBMS-based engines rely on optimization techniques provided by relational databases. However, due to the absence of schematic structure in RDF, cost-based approaches show problematic query estimation which cannot effectively predict the query performance [23]. RDF native query engines typically use heuristics and statistics about the data for selecting efficient query execution plans [21]. Heuristics and statistics based optimization techniques generally work without any knowledge of the underlying data, but in many cases, statistics are often missing [23].

Hassan [9] proposes the first work on predicting SPARQL query execution time by utilizing machine learning techniques. In the work, multiple regression using Support Vector Regression (SVR) is adopted. The evaluation is performed on benchmark queries on an open source triple store Jena TDB[6]. The feature models are extracted based on *Graph Edit Distances* (GED) between each of training queries. However, in practice, we observe that the calculation of GED is very time consuming, which is not a desirable method when the training dataset is large. Moreover, their work omits the study of the cold stage of the system, where query compilation time should not be ignored. In this work, we will investigate the total elapsed time that includes both compilation time and execution time of a query. When the system is in the warm stage of query processing, i.e., the query is not executed for the first time, the compilation time is omitted, thus the elapsed time is equal to the execution time. When the query is new to the system, i.e., in the cold stage of the system, the compilation time cannot be ignored when examining total elapsed time. Although we discuss elapsed time in our work, we will use execution time hereafter for expressivity. In addition to execution time, other performance metrics such as CPU usage and memory usage also need to be considered.

1.2 Challenges

To effectively predict SPARQL query performance, we draw ideas from Hassan but adopt different machine learning techniques to address the issues of training large datasets and investigation of the cold stage of the system. The challenges in our work center on capturing characteristics of SPARQL queries and representing the characteristics as features for the application of machine learning techniques.

- *Feature Obtaining.* Our aim is to promote higher usability of Semantic Web and more effective consumption of RDF-represented information, thus the work on open source triple stores is more applicable. The intuitive solution for obtaining feature is to leverage the query plan provided by triple stores. Similar works

[6] https://jena.apache.org/documentation/tdb/

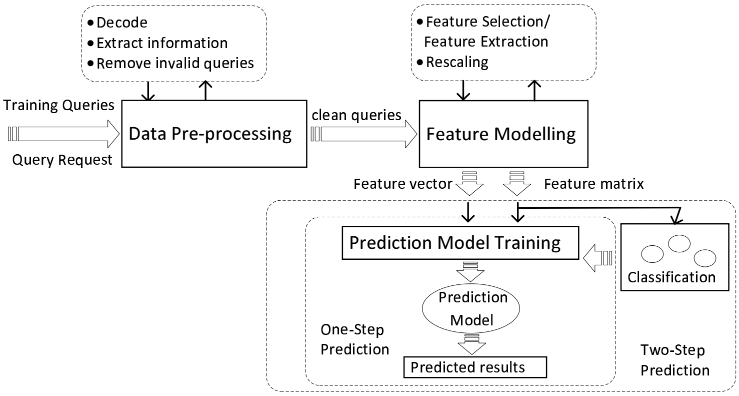


Fig. 1 SPARQL Query Performance Prediction

for SQL queries are presented in [2,7]. However, explicit plan information is hardly available in open source triple stores (we have examined four most commonly used open source triple stores Virtuoso (open source edition)[7], Fuseki[8], Sesame[9] and 4Store[10]). Furthermore, as aforementioned, if the plan information is explicit, it is still based on cost model estimations, which are shown ineffective. Therefore, we cannot leverage query plans to construct the feature of a SPARQL query. Then the question comes to: What else information can we obtain from a SPARQL query that captures the characteristics of the query?

– *Feature Representation.* Given the feature acquired from a SPARQL query, a vector representation is required for machine learning algorithms. How can we convert the features to a feature vector that effectively represents the query without losing information?
– *Feature Extraction.* The training of the prediction model is based on the query features. Irrelevant features will introduce noise in the training process, which leads to distortion of prediction results. How can we select the most predictive features?

1.3 Approaching SPARQL performance prediction

To address the above challenges, in this work, we propose three approaches for modelling features, namely *query algebra features*, *Basic Graph Pattern (BGP) features* and *hybrid features*. Specifically, we transform the algebra and BGPs into a feature vector. We propose a feature selection process based on heuristic to build hybrid features and also compare the feature selection and extraction approaches on the performance of prediction. Once the features are built, they can then be used to estimate the performance of a new requested query based on the feature values that can be obtained without executing the query. We consider both *k*-

[7] http://virtuoso.openlinksw.com/
[8] http://jena.apache.org/documentation/serving_data/
[9] http://rdf4j.org/
[10] http://4store.org/

nearest-neighbour (k-NN) regression and SVR as prediction models. Both average k-NN and weighted k-NN are investigated.

To improve the prediction performance, we develop a two-step prediction process in addition to the one-step prediction, depicted in Figure 1. The figure shows that our prediction process consists of three main components, namely *data pre-processing*, *feature modelling* and *prediction*. Both training queries and new requested queries are cleaned by the data pre-processing component. In the training component, prediction models are derived from the training queries with observed query performance metrics. In this component, queries are represented as a set of features (i.e., predictive variables) with corresponding performance metrics (i.e., target variables). A feature matrix is obtained with each row representing a feature vector of a query. The columns are instances of different features. After obtaining features, we further apply feature selection to reduce the dimension of the feature matrix to extract the most predictive features. In the one-step prediction, feature matrix of the training queries are fed into the prediction model training component, with the goal of creating a mapping between feature values and observed query performance metrics. The prediction models are then used to predict the performance of new requested queries. In the two-step prediction, a classification step is added before training the model. The aim of classification is to group queries with different ranges of execution time. Multiple models are trained for different classes and different performance metrics separately. In this work, we classify the queries into four classes and we perform prediction for execution time, CPU usage and memory usage of SPARQL queries.

For the detailed discussion of the three components, we focus on discussion of feature modelling and prediction in Section 4 and Section 5 and provide a brief introduction of data processing in experiment set-up.

1.4 Contributions and structure

Our approach can be applied in the situation that no estimations of query execution performance are provided, or such estimations are implicit or inaccurate. In practice, this applies to most triple stores that are publicly available. In a nutshell, the main contributions of this work are summarized as follows:

- We adopt machine learning techniques to effectively predict the query performance before their execution. Rather than only predicting query performance on the warm stage, we also consider the cold stage query execution, which is not discussed in the state-of-the-art works. The proposed methods are easy to reproduce as we mainly adapt the most commonly used algorithms in the machine learning field. Furthermore, little domain expertise is required.
- We propose three ways to model features of a SPARQL query. The algebra and BGP features can be acquired from the parsing of the query. The hybrid feature can be derived from algebra and BGP features. All features can be easily obtained without the information provided by the underlying systems.
- We perform extensive experiments on real queries obtained from widely accessed SPARQL endpoints. The triple store we used is one of the most used systems in the community of Semantic Web. Thus our work will benefit a large population of users. Our approach is transferable and can be applied to other triple stores.

The remainder of this paper is structured as follows. Existing research efforts on the related topics are discussed in Section 2. In Section 3, the basic techniques used in this work are briefly introduced. Section 4 presents our feature modelling approaches in detail and Section 5 describes our prediction approaches. In Section 6, we report the experimental results. Finally, we conclude the paper by discussing the issues we observed in this work and some future research directions.

2 Related Work

Although there are very limited previous works that pertain to predicting query performance via machine learning algorithms in the context of SPARQL queries, the literature of the approaches and ideas presented in this paper is extensive. In this section, we introduce some representative works that are closely related to our work.

2.1 Query performance prediction via machine learning algorithms

Predicting query execution time by leveraging machine learning techniques has recently gained significant interest in the database research community. Akdere et al. [2] propose to predict the execution time using Support Vector Machine (SVM). They build predictors by leveraging query plans provided by the PostgreSQL optimizer. The authors also choose operator-level predictors and then combine the two with heuristic techniques. The work studies the effectiveness of machine learning techniques for predicting query latency of both static and dynamic workload scenarios. Ganapathi et al. [7] consider the problem of predicting multiple performance metrics at the same time. The authors also choose query plans to build the feature matrix. Kernel Canonical Correlation Analysis (KCCA) is leveraged to build the prediction model as it is able to correlate two high-dimensional datasets. As addressed by the authors, it is hard to find a reverse mapping from feature space back to the input space and they consider the performance metric of k-nearest-neighbours to estimate the performance of target query. Li et al. [14] estimate CPU time and I/O of a query execution plan. The work also addresses the problem of robust estimation for queries that are not observed in the training stage.

2.2 SPARQL query optimization

RDBMS-based triple stores rely on the query optimization techniques of the relational database systems to evaluate SPARQL queries. Recent works focus on optimizing the joins of SPARQL queries [8, 16]. Neumann et al. [16] introduce characteristic sets, which work with dynamic programming algorithm to provide more accurate selectivity estimations for star-like SPARQL queries. Gubichev et al. [8] propose a SPARQL-tailored join-ordering algorithm aiming at large SPRAQL queries. RDF native query engines use rule-based optimization or leverage heuristics and statistics about the data for selecting efficient query plans [21]. These approaches generally work without any knowledge of the underlying data. Quilitz et al. [18] apply a set of logical rules to a query engine, to calculate all equivalent query plans for a given query and then choose the most optimised query plan to be executed. Stocker et al. [21] present optimization techniques with pre-computed statistics. Based on the statistics, triple patterns are reordered before execution to achieve efficient query processing. Tsialiamanis et al. [23] propose a heuristics-based SPARQL optimizer to maximize the number of merge-joins to speed up the query processing.

3 Preliminaries

3.1 SPARQL query

A SPARQL query can be represented as a graph structure, the SPARQL graph [8]. Given the notation B for blank nodes, I for Internationalized Resource Identifier (IRIs) , L for literals and V for variables, a SPARQL graph pattern expression is defined recursively (bottom-up) as follows [17]:

(i) A valid triple pattern $T \in (IVB) \times (IV) \times (IVLB)$ is a *Basic Graph Pattern* (BGP), where a triple pattern is the triple that any of its attributes is replaced by a variable (A BGP is a graph pattern represented by the conjunction of multiple triple patterns. It models the SPARQL conjunctive queries and is the most used subset of SPARQL queries [4]).

(ii) For BGP_i and BGP_j, the conjunction (BGP_i and BGP_j) is a BGP.

(iii) If P_i and P_j are graph patterns, then (P_i AND P_j), (P_i UNION P_j) and (P_i OPTIONAL P_j) are graph patterns.

(iv) If P_i is a graph pattern and R_i is a SPARQL build-in condition, then the expression (P_i FILTER R_i) is a graph pattern.

3.2 Multiple regression

Multiple regression focuses on finding the relationship between a dependent variable and multiple independent variables (i.e., predictors). It estimates the expectation of the dependent variable given the predictors. Given a training set $(\mathbf{x}_i, y_i), i = 1, ...n$, where $\mathbf{x}_i \in \mathbb{R}^m$ is a m-dimensional feature vector (i.e., m predictors), the objective of multiple regression is to discover a function $y_i{=}f(\mathbf{x}_i)$ that best predicts the value of y_i associated with each $\mathbf{x}_i$ [19].

Support Vector Regression (SVR) is to find the best regression function by selecting the particular hyperplane that maximizes the margin, i.e., the distance between the hyperplane and the nearest point, called support vectors [20]. The error is defined to be zero when the difference between actual and predicted values are within a certain amount ξ. The problem is formulated as an optimization problem:

$$\min \mathbf{w}^T\mathbf{w}, \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi, \xi \geq 0 \tag{1}$$

where parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$. If we extend the dot product of $\mathbf{x}_i \cdot \mathbf{x}_j$ to a different space of larger dimensions through a functional mapping $\Theta(\mathbf{x}_i)$, then SVR can be used in non-linear regression. $\Theta(\mathbf{x}_i) \cdot \Theta(\mathbf{x}_j)$ is called kernel function. An advantage of SVR is its insensitivity to outliers [19].

k-Nearest Neighbours (k-NN) [3] is a non-parametric classification and regression method. The k-NN regression predicts based on k nearest training data. It is often successful in the cases where the decision boundary is irregular, which applies to our training data [9]. By training the k-NN model, the predicted query time can be calculated by the average time of its k nearest neighbours:

$$t_Q = \frac{1}{k}\sum_{i=1}^{k}(t_i), \tag{2}$$

where t_i is the execution time of the i^{th} nearest query.

3.3 Dimension reduction

In machine learning, dimension reduction is the process of reducing the number of random variables to describe a large set of data while still describing the data with sufficient accuracy. Dimension reduction is often performed before other machine learning tasks, such as clustering, classification and prediction. Other benefits include enhancing the interpretatiblity of data, reducing over-fitting and shortening the training times [11].

Dimension reduction is divided into two sub-types: *feature selection* and *feature extraction*. Feature selection returns a subset of the features and the feature selection techniques are often used in the domains where representative features need to be identified, such as weight and height of subjects in healthcare. Feature extraction creates new features from original features by transforming the original features in a high-dimensional space to a space of lower dimensions. The transformation may be linear or non-linear. It is often used to avoid the effects of the curse of dimensionality [19].

4 Feature Modeling

The prediction relies on the features in the training sets, thus the performance of prediction is highly dependent on how much information the features can obtain from the data and how well the features represent the data. In order to utilize machine learning algorithms for SPARQL query performance prediction, we transform the query into vector representation. We formulate the problem as follows:

Definition 1 (SPARQL Feature Modeling) *Let $Q = (F, P)$ denote a SPARQL query, where F is the SPARQL query form in $\{$Select, Describe, Construct and Ask$\}$. P is the query pattern of Q, feature modeling is the transformation that maps $Q \rightarrow \mathbf{q}$, where $\mathbf{q} \in \mathbb{R}^m$ and m is the number of features.*

In this study, we use only static, compiling time features which can be extracted prior to execution. Algebra features and BGP features are obtained by parsing the query text (see Section 4.1 and 4.2). Hybrid features are generated by applying the selection algorithm we develop based on algebra and BGP features (see Section 4.3).

4.1 Algebra features

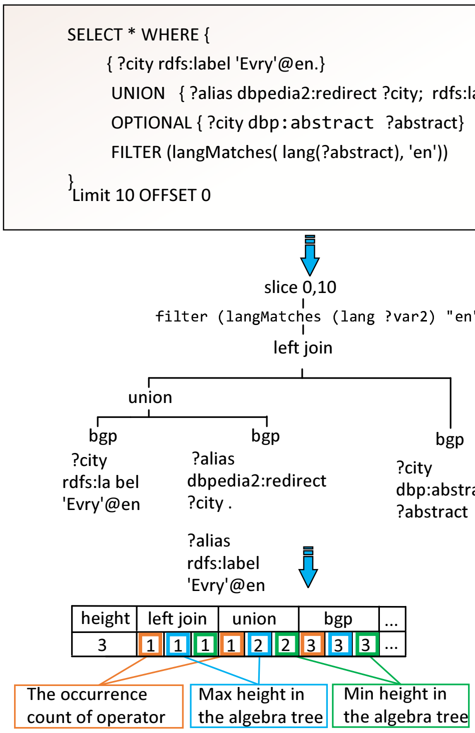
The step of parsing the query string to algebras is prior to optimization by query engines. The algebras can be presented as a tree:

Definition 2 (Algebra Tree) *Given an SPARQL query Q, algebra tree $T_{Algebra}(Q)$ is a tree where the leaves are BGPs and nodes are algebra operators presented hierarchically. The parent of each node is the parent operator of current operator.*

We obtain the SPARQL algebra tree and then traverse the tree to construct the *Algebra Set* by recording the occurrences and hierarchical information of each algebra operator.

Definition 3 (Algebra Set) *Given an algebra tree denoted as $T_{Algebra}(Q)$, the Algebra Set is a set of tuples $\{(opt_i, c_i, maxh_i, minh_i)\}$, where opt_i is the operator name, c_i is the occurrence count of opt_i in $T_{Algebra(Q)}$, $maxh_i$ and and $minh_i$ are opt_i's max height and min height in $T_{Algebra(Q)}$, respectively.*

We then transform the algebra set to a vector by concatenating all the tuples in the algebra set sequentially. We further insert the tree's height at the beginning of the vector. The values of each position is considered as an instance of a feature, thus we obtain a feature vector for a query. Figure 2 illustrates an example of algebra feature modelling. We have a 40-dimensional feature vector for the example SPARQL query.



```
SELECT * WHERE {
      { ?city rdfs:label 'Evry'@en.}
       UNION  { ?alias dbpedia2:redirect ?city;  rdfs:label 'Evry'@en. }
       OPTIONAL { ?city dbp:abstract  ?abstract}
       FILTER (langMatches( lang(?abstract), 'en'))
}Limit 10 OFFSET 0
```

height	left join			union			bgp			...
3	1	1	1	1	2	2	3	3	3	...

The occurrence count of operator
Max height in the algebra tree
Min height in the algebra tree

Fig. 2 Algebra Feature Modeling

4.2 BGP features

Algebra features take occurrences and hierarchical information of operators into consideration. For complementary, we propose to leverage graph structure of BGPs, the most used subset of SPARQL queries [4] to form BGP features. We propose to capture the characteristics of BGPs and transform them into a vector representation. We firstly examine BGPs that consist of sets of triple patterns. We could follow the way in the algebra feature modeling to count the number of occurrences of triple patterns. However, in this way, we fail to represent the hierarchy of triple patterns that are rooted at the target BGP. We further consider that a triple pattern is essentially a graph. Figure 3 illustrates the graph representation of two triple patterns *(?s, p, o)* and *(?s, p, ?o)*. The question mark indicates that the corresponding component is a variable. However, it is hard to tell the differences of the two graphs, as they are structurally identical.

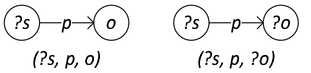


Fig. 3 Example Triple Patterns

In our work, we formulate the problem as follows:

Definition 4 (BGP Graph Modeling) *Let $bgp_i = \{tp_1, tp_2, ..., tp_n\}$ denote a BGP of an SPARQL Query. The $tp_k, k \in (1, n)$ is a triple pattern rooted at bgp_i. $ged(g_o, g_d)$ represents the graph edit distance between graph g_o and graph g_d. BGP graph modeling is the task that models each tp_k to a graph g_{tp_k} satisfying $ged(g_{tp_k}, g_{tp_l}) > 0$ when $k \neq l$.*

To address the above problem, we propose to map all the eight types of triple patterns to eight structurally different graphs, as shown in Figure 4 (left). To exemplify, we model the triple patterns of BGPs in the example query in Figure 2 to a graph, which is depicted in Figure 4 (right). The black rectangles in the figure are conjunction nodes. Triple patterns of a query are connected according to the hierarchy of these patterns in the query.

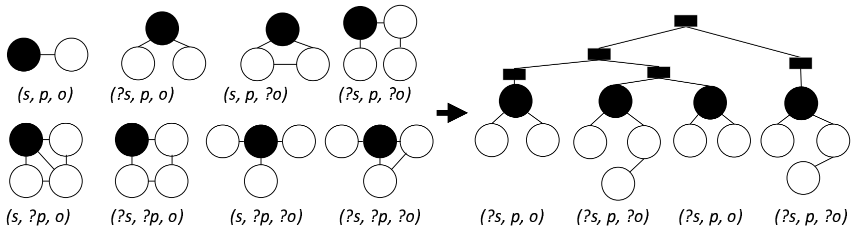


Fig. 4 Mapping Triple Patterns to Graphs. Left: eight types of triple patterns are mapped to eight structurally different graphs. Right: mapping example query in Figure 2 to a graph.

After transforming the query to a graph, we propose to use the graph edit distances between queries to build feature vectors for queries. Graph edit distance between two graphs is the minimum amount of edit operations (i.e., deletion, insertion and substitutions of nodes and edges) needed to transform one graph to the other. We take the edit path from *(?s, p, o)* to *(?s, p, ?o)* in Figure 4 (left) as an example, the steps are shown in Figure 5.

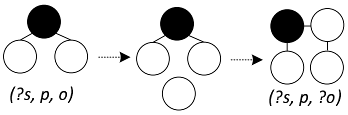


Fig. 5 Graph Edit Path from *(?s, p, o)* to *(?s, p, ?o)*

For a query q, the GEDs between q and other queries could form a feature vector, in which each feature instance is a GED. In [9], the authors propose to use GEDs between q and all training queries (used for training prediction models) to form the feature vector of q. However, this calculation is very time consuming and lack of scalability. To address this issue, we propose to select some representative queries as target queries and calculate the GEDs between q and the target queries. These GEDs form a feature vector for q. The computation is thus largely reduced. DBPSB benchmark query templates [15] represent the most used patterns in real-world queries. Therefore, we choose queries generated from its templates as the representative queries (or target queries). Specifically, we choose 18 valid ones out of 25 templates in the DBPSB benchmark (templates that cannot generate queries are excluded: Templates 1, 2, 3, 10, 16, 21 and 23) and generate one target query for each template. We model the BGPs of q and the 18 target queries based on graph mappings depicted in Figure 4 (left). Then the GEDs between graph of q and target graphs are calculated. By recording the GEDs between q with 18 target graphs, we obtain a 18-dimensional feature vector for q.

4.3 Hybrid features

We build a hybrid feature by selecting the most predictive features based on algebra and BGP features. Most feature selection approaches rank the candidate features and use this ranking to guide a heuristic search to identify the most predictive features. In this paper, we will use a similar forward feature selection algorithm, but we choose the contribution to overall prediction performance as the heuristic. The algorithm performs a best-first search in the feature space. It starts with building predictive models using a small number of features and iteratively builds more complex and accurate models by using more features. In each iteration, a new feature is constructed, tested, and added to the feature set. If it improves the overall prediction performance, the feature is selected. We do not consider building multiple models of different types of features for solving the model selection problem. Instead, we use a single type of prediction model, k-NN, because of its excellent performance. Finally, we simply consider the completion of traversing all features as the stopping condition.

Algorithm 1 presents the feature selection algorithm. Firstly, we choose from feature described in Section 4.1 and Section 4.2 (line 2). We put BGP features

Algorithm 1 Plan Feature Selection Algorithm

```
Input: Training Queries:data
Input: Prediction Model:model
Input: Feature Models:feature_models
Output: Prediction performance metric value:val
Output: Selected feature list:list
 1: Initialize: val← zero; list← ∅
 2: for fm in feature_models
 3:     while feature←get_feature_next(fm,data)
 4:     do
 5:         list=list.add(feature)
 6:         [predictions, new_val]=apply_model(data,list)
 7:         if new_val>val then
 8:           val←new_val
 9:        else
10:           list.remove(feature)
11:     end
12: end for
```

ahead of algebra features. Then we forwardly choose single feature from these features and evaluate the performance of prediction with current chosen feature (line 3-6). When a new chosen feature contributes to the overall performance, we add it to the candidate list (line 7-8). Otherwise, it is not selected (line 10). The output of the algorithm is a list of selected features.

5 Predicting SPARQL Query Performance

To predict query performance metrics prior to query execution, we apply machine learning techniques on historical executed queries (the training set). We work with query execution time, CPU usage and memory usage as the query performance metrics. Once a prediction model is derived from the training queries, it can then be used to estimate the performance of new requested queries based on the query feature values that can be obtained without executing the query. We train separate prediction models for each of the performance metrics. Our approach does not require prior knowledge of the underlying RDF data, thus it is treated as a black box that the behaviour of queries are learned only from the executed queries.

5.1 Predictive models

We choose two regression approaches in this work, SVR and k-NN Regression because SVR is insensitive to outliers [25] and k-NN is suitable for irregular data points [9]. The models are trained with features of training queries as well as the actual query performances of these queries. Then the models can be used to estimate the performance of a new issued query through extracting its features.

5.1.1 SVR

Four commonly used kernels are considered in our prediction: namely *Linear*, *Polynomial*, *Radial Basis* and *Sigmoid*, with different kernel parameters γ and r:

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$.
- Polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^r$.
- Radial Basis: $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2), \gamma > 0$.
- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\gamma\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j) + r$.

5.1.2 KNN regression

We apply four kinds of k-NN regression by considering different weighting schemes.

- *Nearest.* The execution time of the nearest (k=1) neighbour is considered as the predicted time for the new query.
- *Average.* We assign equal weights to each of the k nearest neighbours and use the average of their execution time as the predicted time. The calculation is based on Equation (2) (Section 3.2).
- *Power.* The weights in *Power* is the power value of weighting scale α. The predicted query time is calculated as follows:

$$t_Q = \frac{1}{k}\sum_{i=1}^{k}(\alpha^i * t_i), \tag{3}$$

  where α^i is the weight of the *i-th* nearest query.
- *Exponential.* We apply an exponential decay function with decay scale α to assign weights to neighbors with different distance.

$$t_Q = \frac{1}{k}\sum_{i=1}^{k}(e^{-d_i*\alpha} * t_i), \tag{4}$$

  where d_i is the distance between target query and its *i-th* nearest neighbor.

5.2 Two-step prediction

We observe that prediction process with one-step, i.e., all the training data are input into the model training stage, gives undesirable performance. A possible reason is the fact that our training dataset has queries with various time ranges. Fitting a curve in such irregular data points is often inaccurate. Then we follow a two-step prediction process. We firstly split the training data according to execution time ranges, then we train different prediction for different time ranges. Specifically we put the training queries in four bins (or classes), namely *short*, *medium short*, *medium*, and *long*. The time ranges in these four bins are <0.1 second, 0.1 to 10 seconds, 10 to 3,600 seconds, and >3,600 seconds respectively. We correspondingly label training queries into four labels. Then 12 prediction models are trained, with three models for execution time, CPU usage and memory usage respectively for each of the four classes. When a new query comes, we first classify it to the possible class, then apply the corresponding prediction models of the target class. In this way, the performance improves significantly (see Section 6.5).

6 Experiments

6.1 Setup

Real world queries. We use real world queries gathered from USEWOD 2014 challenge[11], which provides query logs from DBPedia's SPARQL endpoint[12] (DBpedia3.9). These logs are formatted in Apache Common Log Format and are encoded. In the data preprocessing step, we process the log files and extract valid queries by decoding, extracting interesting values (IP, date, query string), identifying SPARQL queries from query strings and removing duplicated and invalid queries. Here, invalid queries include all incomplete queries, queries in languages rather than English and queries with syntax errors according to the SPARQL1.1 specification. We work on SELECT queries in the experiments as more than 98% of queries are SELECT queries [26]. We finally retrieve 198,235 valid queries from DBpedia3.9. We randomly choose 10,000 valid queries in our prediction evaluation. We execute these queries 11 times as suggested in [8] and record their execution execution time, CPU usage and memory usage. We consider the first time as the cold stage and the remaining 10 times as the warm stage. We calculate the average of the remaining 10 times as the actual performance of each query for warm stage prediction. We split the collection to training and test sets according to the 4:1 tradition.

System. The backing system of our local triple store is Virtuoso 7.2, installed on 64-bit Ubuntu 14.04 Linux operation system with 32GB RAM and 16 CPU cores. We set up a local mirror of DBpedia3.9 English dataset on the Virtuoso server. Table 1 shows summary statistics of the dataset. The query performance (execution time, CPU usage and memory usage) and query plans are obtained from executing the queries when enabling the profile function of Virtuoso. All the machine learning algorithms are performed on a PC with 64-bit Windows 7, 8GB RAM and 2.40GHZ Intel i7-3630QM CPU.

Table 1 Statistics for DBpedia3.9 (English)

#.Triples	#.Subject	#.Predicate	#.Object
463,342,557	27,706,241	53,338	133,397,629

Implementation. We use SVR for kernel and linear regression available from LIBSVM [5]. We also use SVM supported by LIBSVM for the classification stage in two-step prediction. k-NN and weighted k-NN regression is designed and implemented using Matlab programming. The heuristic based feature selection algorithm is also implemented in Matlab. The algebra tree used for extracting algebra features is parsed using Apache Jena-2.11.2 library, Java API. Graph edit distance used for building BGP features is calculated using the Graph Matching Toolkit[13].

[11] http://usewod.org/

[12] http://dbpedia.org/sparql/

[13] http://www.fhnw.ch/wirtschaft/iwi/gmt

Table 2 Relative Error (%) of prediction on multiple performance metrics. SVR-L denotes SVR-Linear, SVR-P is SVR-Polynomial, SVR-R is SVR-RadialBasis and SVR-S is SVR-Sigmoid.

	SVR-L	**SVR-P**	**SVR-R**	**SVR-S**	***k*-NN ($k = 1$)**
Execution time (Cold)	99.69	99.46	99.74	99.68	21.94
Execution time (Warm)	97.59	97.33	97.86	97.57	20.89
CPU usage (Cold)	111.39	110.53	112.25	111.36	38.22
CPU usage (Warm)	106.72	105.46	107.33	106.68	36.25
Memory usage (Cold)	103.39	103.34	103.85	103.41	26.85
Memory usage (Warm)	101.39	101.25	101.93	101.37	23.49

Evaluation metric. We follow the suggestion in [2] and use the *mean relative error* as our prediction metric:

$$relativeerror = \frac{1}{N}\sum_{i=1}^{N}\frac{|actual_i - estimate_i|}{actual_{mean}} \quad (5)$$

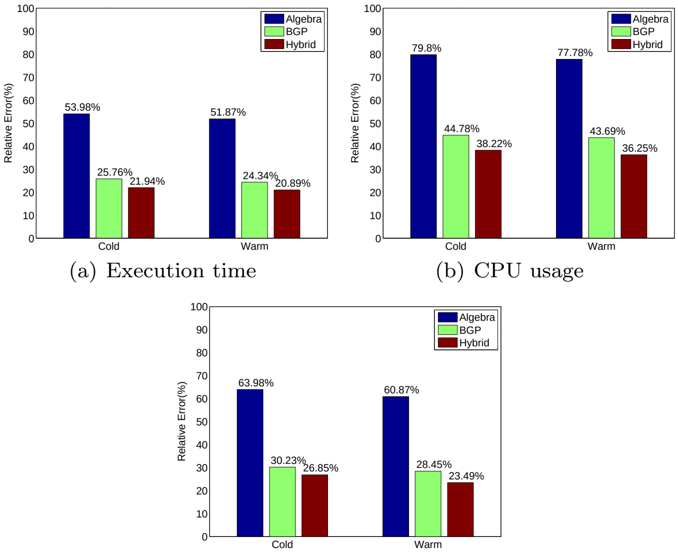
The difference with the calculation in [2] is that we divide $actual_{mean}$ instead of $actual_i$ because we observe there are zero values for $actual_i$.

Relative error is useful when we aim to minimize the relative prediction error for all queries regardless of the actual value. Non-relative error metrics such as square error would be better for minimizing the absolute difference (or its square) in actual and predicted values. One other most widely used metric R^2 is usually computed on the training data [7], but we want to evaluate the fitting of test data.

6.2 Prediction models comparison

We compared the Linear SVR and SVR with three kernels, namely Polynomial, Radial Basis and Sigmoid. We also calculated the relative error for k-NN when k=1. The feature model used in the experiments was the hybrid feature.

Table 2 gives the relative error of predictions of the targeted performance metrics. From the result we can see that the SVR models with various kernels have higher relative errors than k-NN. All the relative errors exceed 97%, indicating the predictions are far from the true values. For cold stage prediction of execution time, k-NN model performs much better with 21.94% in relative error. In warm stage, the relative error goes down to 20.89%. For CPU and memory usage, k-NN model performs much better than SVR models with relative error under 40%, whereas the values exceed 100% using SVR models. We further investigate this result and find two possible reasons. One is that the execution time has a broad range and SVR considers all the data points in the training set to fit the real value, whereas k-NN only considers the points close to the test point. The other reason is we use mean of actual value in Equation (5), and the values that are far from average will lead to distortion of mean value. Given this result, we chose to use k-NN model by default in the following evaluations. Only in the two-step prediction evaluation, we compared SVR with k-NN.



(a) Execution time

(b) CPU usage

(c) Memory usage

Fig. 6 One-step prediction for multiple performances with different features.

6.3 Feature modelling comparison

We evaluate the prediction ability of three proposed feature modelling, namely algebra features, BGP features and hybrid features. We further adopt dimensional reduction on hybrid features to evaluate the performance of three most used dimensional reduction algorithms.

6.3.1 Performance of three types of features

As the hybrid feature model is built on the feature selection algorithm (see Algorithm 1), we compared its performance with the algebra and BGP feature models to demonstrate the performance comparison with and without feature selection. Figure 6 gives the result, showing the relative errors for these three approaches in both cold and warm stages.

From the figure we can see for all three performance metrics, the hybrid feature performs the best and the BGP feature performs better than the algebra feature. The prediction of execution time gives the best result, with 21.94% relative error in the warm stage and 20.89% relative error in the cold stage. CPU usage is the percentage of CPU used for executing a query. The prediction of CPU usage is slightly poorer. The best prediction is 36.25% relative error when using the hybrid features in the warm stage, and 38.22% is achieved in the cold stage. The reason of prediction on CPU usage having higher relative error is that the CPU scheduling of the underlying operating system for each thread is not the same. Therefore, even for the same query, each time it is executed, the CPU usage might be different.

6.3.2 Performance of dimensional reduction algorithms on hybrid features

Based on the selected hybrid features, we further applied three feature extraction algorithms to extract the most predictive information. We examine tree most used dimension reduction techniques in this work, namely Principle Component Analysis (PCA)[12], Canonical Component Analysis (CCA) [10] and Nonnegative Matrix Factorization (NMF) [13]. We implement them to reduce the dimension of query feature matrix.

Figure 7 presents the prediction results for three performance metrics on both warm and cold stages. We observe that NMF shows the worst result, CCA gives medium performance and PCA has the best performance among the three. However, the performance difference between PCA and with or without dimensional reduction is not obvious, indicating that dimension reduction is not suitable for our data. The reason is that dimension reduction algorithms perform well when the original dimension is high, but the dimension of our feature matrices and vectors is relatively low (less than 100). Therefore, we do not apply dimension reduction algorithms in our following evaluations.

6.4 Comparison of different weighting schemes in k-NN regression

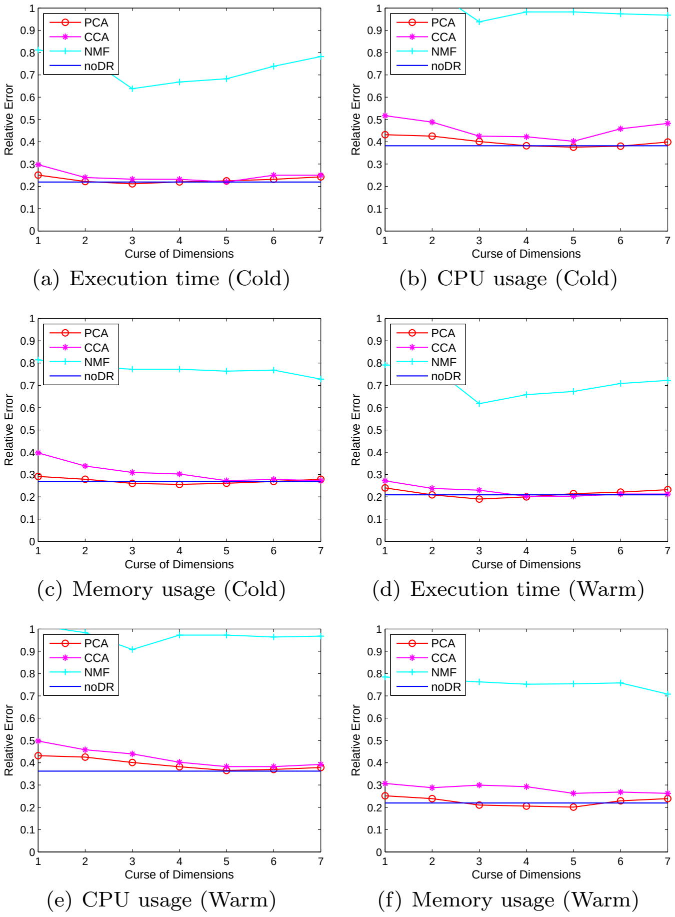
We evaluated three weighting schemes discussed in Section 5.1.2, namely *Average*, *Power* and *Exponential.* All the scaling parameters were chosen through five-fold cross-validation. We used hybrid features in this evaluation. Both warm and cold stages were evaluated. We presented only the result for execution time because other performance metrics provide similar results. We observe from Figure 8 that the power weighting achieves the best performance. In the warm stage, the 15.32% relative error is achieved when k=5. The trend of relative error returns to upward after k=5. *Average* is the worst weighting method for our data. Exponential weighting does not perform as well as we expected although it is better than average weighting. Weighting schemes show similar performances when the query execution is in the cold stage, i.e., when k=5, the power weighting achieves the lowest relative error of 18.29%. We therefore used k=5 power weighting in following evaluations.

6.5 Performance of two-step prediction

We observe that the performance of one-step prediction is not desirable with the lowest relative error 15.32% when predicting execution time for warm stage querying. A possible reason is the broad execution time range of actual SPARQL queries. The long time queries will distort the mean of actual observations that make the

Table 3 Relative errors (%) of two-Step prediction with k-NN and SVR. The values delimited by comma are for (cold, warm) respectively.

Prediction model	Execution time (sec.)	CPU usage	Memory usage
5NN($\alpha = 0.3$)	11.06,9.81	37.78,35.34	23.58,18.94
SVR-Polynomial	22.39,20.30	60.15,58.56	40.34,36.62



(a) Execution time (Cold)

(b) CPU usage (Cold)

(c) Memory usage (Cold)

(d) Execution time (Warm)

(e) CPU usage (Warm)

(f) Memory usage (Warm)

Fig. 7 Comparison of dimension reduction algorithms on hybrid features in one-step prediction. X-axis is the number of dimensions.

relative error inaccurate. We thus propose the two-step prediction process as discussed in Section 5.2. Here we evaluate the performance of two-step prediction. We used Support Vector Machine for the classification task in two-step prediction and achieved accuracy of 98.36%, indicating that we can accurately predict the time range in the first step. For comparison, we use log-log plotting for the one-step and two-step prediction in Figure 9. We only show the results for execution time in both cold and warm stages here. The one-step warm stage prediction is worse than the two-step cold stage prediction and the performance gap to two-step warm stage prediction is larger.

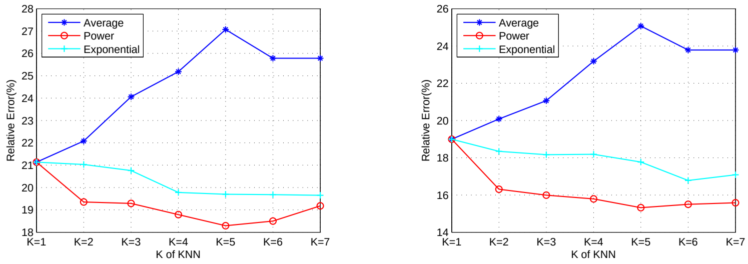
We also apply SVR-Polynomial in two-step prediction for comparison. We select polynomial here because it shows the best performance among all the kernels we considered in this paper (See Table 2). We used the power weighting of k-NN with k=5 and $\alpha = 0.3$ as it shows the best performance (Section 6.4). Table 3 shows the result of comparison. It shows that SVR regression model still does not perform desirably.

6.6 Comparison to the State-of-the-art

The only work that exploits machine learning algorithms to predict SPARQL query is the work in [9]. In our evaluation, we implement their approach and compare their work with ours. To implement the approach in [9], we calculate GED for all pairs of 1,000 randomly chosen queries and cluster the training queries using X-means per description in [9], then build distance feature vectors for each test query, and each instance is the distance between the test query and the center query of each cluster. We then train prediction models for each cluster. Finally we use the trained model to predict the performance of test queries. Table 4 shows the result of comparison on warm stage querying. The training time includes feature modelling, clustering and classification for work in [9]. The first part takes most time because the calculation of GED for all training queries is time-consuming. In our approach, we reduce the GED calculation dramatically. But this calculation still takes most of the time in the prediction process. The time gap of training process between ours and theirs will be enlarged when more training queries are involved because their approach takes squared time. We do not have a clustering process that further reduces the time. Our approach also shows better prediction performance with lower relative error for all three prediction performances.

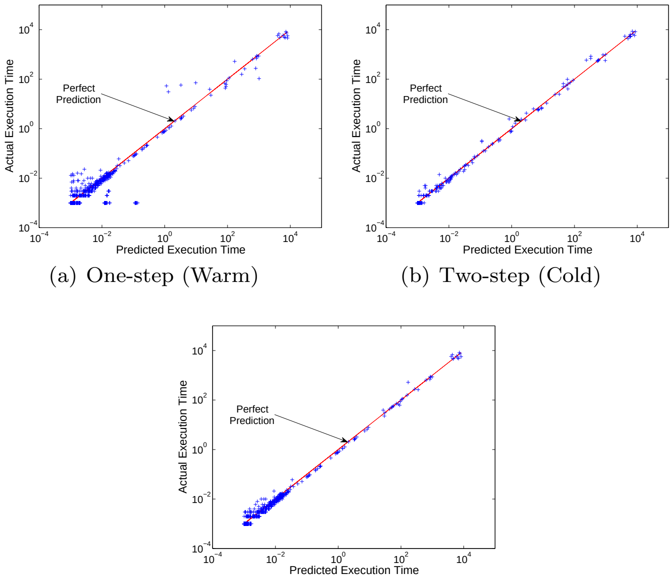
Table 4 Comparison to the state-of-the-art work. Training time for 1,000 queries are compared as well as the relative errors for each performance metric.

	Our approach	**Approach in [9]**
Models	SVN+Weighted KNN	X-means+SVM+SVR
Training time 1k queries(sec.)	51.36	1548.45
err% of Execution time	9.81	14.39
err% of CPU usage	35.34	38.67
err% of Memory usage	18.94	22.24



(a) Cold (b) Warm

Fig. 8 One-step execution time prediction of different weighting method on k-NN



(a) One-step (Warm)

(b) Two-step (Cold)

(c) Two-step (Warm)

Fig. 9 One-step and two-step prediction fitting for execution time (seconds)

7 Discussions

In this section, we discuss the observations and issues from this work.

7.1 Plan features

Some works use machine learning techniques that leverage the information provided by the query plan, which is given by a query optimizer. The information that the plan provides in these works includes estimated total execution time, estimated result row count, estimated time and row count for each operators. However, there are two obstacles for using such information in our work. Firstly, this information is based on the cost model estimation, which has been proven ineffective [2,7]. It is unlikely to achieve desirable performances based on inaccurate estimations. Secondly, as we target open source triple stores to benefit more data consumers, we are only able to obtain information from these systems. However, most of them fail to provide an explicit query plan. Thus we turn to choose structure-based features that can be obtained directly from query text. As argued in many works, similarly structured queries may have huge performance gaps due to the irregular distribution of values in the queried data. But from our practical experience in this work, we observed that although it leads to distortion of the prediction, the error rate is acceptable based on the limited features we can acquire.

7.2 Cost model vs machine learning

Cost model based optimization estimates execution time based on the cost. Although it is arguably less accurate than machine learning based prediction, it is faster than machine learning prediction. The downside is that those estimations are always inaccessible for personal users. To promote the usability of Semantic Web techniques and RDF data, it is better to be more consumer friendly. Therefore, the machine learning approaches are a better choice, as the publicly accessible tools are easy to use for training and testing.

7.3 Training size

In the training process, the larger the size of the training data, the better performance we can get. The reason is that more data variety is seen and the model will be less sensitive to unforeseen queries. However, in practice, it is time consuming to obtain the query execution time of a large collection of queries. That is the possible reason why many other works only use small sizes of queries in their evaluation. This fact will cause the bias of the prediction result and makes similar works hard to compare. Although our experimental query set is larger than theirs, we still consider to further enlarge the size of our query set to cover more various queries in the future.

7.4 Dynamic vs static data

In dynamic query workloads, the queried data is frequently updated. Therefore, the prediction might perform poorly due to lack of update of training data. Our work focuses on prediction on static data and we expect training to be done in a periodical manner. In the future we plan to investigate the techniques that can make prediction more available for continuous retraining, which reflects recently executed queries.

Moreover, the query performance would vary when execution environment changes. Thus we perform the evaluations on fixed dataset and run 10 times to get the warm stage performance. Our approach shows desirable prediction performance in this scenario. Although our approach is not designed to predict query execution performance under changing environments (e.g., updating of data, changing of resources etc.), it can be an indicator of the query performance compared to other queries.

8 Conclusion

To conclude, we leverage machine learning techniques to predict multiple performance metrics for SPARQL queries. We transform a given SPARQL query to a vector representation. Feature vectors are built by exploiting the syntactic and structural characteristics of SPARQL queries. SVR and KNN regression models are adopted as prediction models in this work. We observe that KNN performs better than SVR for our data because of the irregular distribution of query performance. The dimension reduction technique is not suitable for our low-dimension feature matrices and vectors. The proposed two-step prediction performs much better because it considers the broad range of observed execution time. The prediction of execution time is more accurate, however for CPU usage, the prediction is not desirable. The reason is that the CPU usage is rarely consistent even for identical queries executed in the warm stage. The prediction in the warm stage is generally better than in the cold stage. We observe that the reason comes from same structured queries. We also observe that many queries are issued by programmatic users, who tend to issue queries using query templates. In the future, we plan to consider the dynamic workload when data is not static. Techniques that can incorporate new training data into an existing model will also be considered.

References

1. Ahmad, M., Duan, S., Aboulnaga, A., Babu, S.: Predicting completion times of batch query workloads using interaction-aware models and simulation. In: Proc. of the 14th International Conference on Extending Database Technology (EDBT 2011), pp. 449–460. Uppsala, Sweden (2011)
2. Akdere, M., Çetintemel, U., Riondato, M., Upfal, E., Zdonik, S.B.: Learning-based query performance modeling and prediction. In: Proc. of the 28th International Conference on Data Engineering (ICDE 2012), pp. 390–401. Washington DC, USA (2012)
3. Altman, N.S.: An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. The American Statistician **46**(3), 175–185 (1992)
4. Bursztyn, D., Goasdoué, F., Manolescu, I.: Optimizing reformulation-based query answering in RDF. In: Proc. of the 18th International Conference on Extending Database Technology (EDBT 2015), pp. 265–276. Brussels, Belgium (2015)
5. Chang, C., Lin, C.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**(3), 27 (2011)
6. Duggan, J., Çetintemel, U., Papaemmanouil, O., Upfal, E.: Performance prediction for concurrent database workloads. In: Proc. of the 2011 International Conference on Management of Data (SIGMOD 2011), pp. 337–348. Athens, Greece (2011)
7. Ganapathi, A., Kuno, H.A., Dayal, U., Wiener, J.L., Fox, A., Jordan, M.I., Patterson, D.A.: Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In: Proc. of the 25th International Conference on Data Engineering (ICDE 2009), pp. 592–603. Shanghai China (2009)
8. Gubichev, A., Neumann, T.: Exploiting the query structure for efficient join ordering in SPARQL queries. In: Proc. of the 17th International Conference on Extending Database Technology (EDBT 2014), pp. 439–450. Athens, Greece (2014)
9. Hasan, R.: Predicting SPARQL Query Performance and Explaining Linked Data. In: Proc. of the 11th Extended Semantic Web Conference (ESWC 2014), pp. 795–805. Anissaras, Crete, Greece (2014)
10. Hotelling, H.: Relations between two sets of variates. Biometrika **28**(3/4), 321–377 (1936)
11. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning. Springer (2013)
12. Jolliffe, I.: Principal Component Analysis. Wiley Online Library (2002)
13. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755), 788–791 (1999)
14. Li, J., König, A.C., Narasayya, V.R., Chaudhuri, S.: Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques. The VLDB Endowment (PVLDB) **5**(11), 1555–1566 (2012)
15. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.N.: Usage-Centric Benchmarking of RDF Triple Stores. In: Proc. of the 26th AAAI Conference on Artificial Intelligence. Toronto, Canada (2012)
16. Neumann, T., Moerkotte, G.: Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In: Proc. of the 27th International Conference on Data Engineering (ICDE 2011), pp. 984–994. Hannover, Germany (2011)

17. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. ACM Transactions on Database Systems **34**(3), 16:1–16:45 (2009)
18. Quilitz, B., Leser, U.: Querying distributed rdf data sources with sparql. In: Proc. of the 5th Extended Semantic Web Conference (ESWC 2008), pp. 524–538. Tenerife, Spain (2008)
19. Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press (2011)
20. Smola, A., Vapnik, V.: Support Vector Regression Machines. Advances in neural information processing systems **9**, 155–161 (1997)
21. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In: Proc. of the 17th International World Wide Web Conference (WWW 2008), pp. 595–604. Beijing, China (2008)
22. Tozer, S., Brecht, T., Aboulnaga, A.: Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In: Proc. of the 26th International Conference on Data Engineering (ICDE 2010), pp. 397–408. Long Beach, USA (2010)
23. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P.A.: Heuristics-based query optimisation for SPARQL. In: Proc. of the 15th International Conference on Extending Database Technology (EDBT 2012), pp. 324–335. Uppsala, Sweden (2012)
24. Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigümüs, H., Naughton, J.F.: Predicting query execution time: Are optimizer cost models really unusable? In: Proc. of the 29th International Conference on Data Engineering (ICDE 2013), pp. 1081–1092. Brisbane Australia (2013)
25. Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A.F.M., Liu, B., Yu, P.S., Zhou, Z., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 algorithms in data mining. Knowledge and Information Systems **14**(1), 1–37 (2008)
26. Zhang, W.E., Sheng, Q.Z., Taylor, K., Qin, Y.: Identifying and Caching Hot Triples for Efficient RDF Query Processing. In: Proc. of the 20th International Conference on Database Systems for Advanced Applications (DASFAA 2015), pp. 259–274. Hanoi, Vietnam (2015)