Learning-Based SPARQL Query Performance Prediction

Wei Emma Zhang[1(✉)], Quan Z. Sheng[1], Kerry Taylor[2], Yongrui Qin[3], and Lina Yao[4]

[1] School of Computer Science, The University of Adelaide, Adelaide , Australia
wei.zhang01@adelaide.edu.au

[2] Research School of Computer Science, Australian National University, Canberra, Australia

[3] School of Computing and Engineering, University of Huddersfield, Huddersfield, UK

[4] School of Computer Science and Engineering, UNSW Australia, Sydney, Australia

Abstract. According to the predictive results of query performance, queries can be rewritten to reduce time cost or rescheduled to the time when the resource is not in contention. As more large RDF datasets appear on the Web recently, predicting performance of SPARQL query processing is one major challenge in managing a large RDF dataset efficiently. In this paper, we focus on representing SPARQL queries with feature vectors and using these feature vectors to train predictive models that are used to predict the performance of SPARQL queries. The evaluations performed on real world SPARQL queries demonstrate that the proposed approach can effectively predict SPARQL query performance and outperforms state-of-the-art approaches.

Keywords: SPARQL · Feature modeling · Prediction

1 Introduction

The Semantic Web, with its underlying data model RDF and its query language SPARQL, has received increasing attention from 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 modeling. Since the number of publicly available RDF datasets and their volume grows dramatically, it becomes essential to provide efficient querying of large scale RDF datasets. This is an important issue in the sense that whether to obtain knowledge efficiently affects the adoption of RDF data as well as the underlying Semantic Web technologies.

Substantial works focus on the prediction of query performance (e.g., execution time) [1,9,16]. Prediction of query execution performance can benefit many system management decisions, including workload management, query scheduling, system sizing and capacity planning. Studies show that cost model based query optimizers are insufficient for query performance prediction [2,6]. Therefore, approaches that exploit the machine learning techniques to build predictive models have been proposed [2,6]. These approaches treat the database system as a black box and focus on learning a query performance predictive model, which are evaluated as feasible and effective [2]. These works extract the features of queries by exploring the query plan with which they can provide estimations for execution time and row count.

However very few efforts have been made to predict the performance of SPARQL queries. SPARQL query engines can be grouped into two categories: RDBMS-based triple stores and RDF native triple stores. RDBMS-based triple stores 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 and cannot effectively predict the query performance [15]. RDF native query engines typically use heuristics and statistics about the data for selecting efficient query execution plans [14]. Heuristics and statistics based optimization techniques generally work without any knowledge of the underlying data, but in many cases, statistics are often missing [15]. Hassan [8] proposes the first work on predicting SPARQL query execution time by utilizing machine learning techniques. The key contribution of the work is to model a SPARQL query to a feature vector, which can be used in machine learning algorithms. However, in practice, we observe that modeling approach is very time consuming.

To address this issue, we leverage both syntactical and structural information of the graph-based SPARQL queries and propose to use the hybrid features to represent a SPARQL query. Specifically, we transform the algebra and BGPs of a SPARQL query into two feature vectors respectively and perform a feature selection process based on heuristic to build hybrid features from these two feature vectors. Our approach reduces the computation time of feature modeling in orders of magnitude. Once the features are built, we use machine learning algorithms to train the prediction model. The input of the algorithm is the feature matrix of the training queries (we concatenate the feature vectors of individual queries into a matrix) and the query performance of these queries (here we only consider the elapsed time used to perform a query and get the result). The output is the trained prediction model. When a new query q is issued, we obtain its feature vector using our feature modeling approach. Then we use the trained prediction model to predict the performance of q. K-Nearest Neighbors (k-NN) regression and Support Vector Regression (SVR) are both considered as the predictive model. We develop a two-step prediction process to improve the prediction result compared to one-step prediction. Moreover, we evaluate our approach on both cold (i.e., fresh queries) and warm (i.e., repeated queries) stages of the system. In the cold stage, elapsed time consists of both compile and execution time while in the warm stage, elapsed time equals to the execution time. The reason we can ignore the compile time is because that our work only considers static querying data. Thus a repeated query has the same execution plan each time it is issued and the system only compiles once.

The consideration of cold stage is useful as knowing execution performance for unseen queries is more important for system management than to previously seen queries.

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

- We adopt machine learning techniques to predict the SPARQL query performance before their execution. We transform the SPARQL queries to feature vectors that are required by the machine learning algorithms. Hybrid feature modeling is proposed based on the features that can be obtained without the information of the underlying systems.
- We consider both warm and cold stage prediction, and the latter one has not been discussed in the state-of-the-art works, but is important to the examination of execution performance of a query.
- We perform extensive experiments on real-world queries obtained from widely accessed SPARQL endpoints. The triple store we used is one of the most widely used systems in the Semantic Web community. Thus our work can benefit a large population of users. Moreover, our approach is system independent that 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 Sect. 2. In Sect. 3, the background knowledge is briefly introduced. Section 4 describes our prediction approaches in detail. Section 5 reports the experimental results. Finally, we discuss some issues we observed and conclude the paper in Sect. 6.

2 Related Work

There are limited previous works that pertain to predicting query performance via machine learning algorithms in the context of SPARQL queries. We introduce here the works of predicting SQL queries performances that we draw ideas from and discuss the work in [8], which focuses on SPARQL queries.

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. [6] consider the problem of predicting multiple performance metrics at the same time. The authors also choose query plan to build the feature matrix. Kernel Canonical Correlation Analysis (KCCA) is leveraged to build the predictive model as it is able to correlate two high-dimension 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-NN to estimate the performance of target query. Hassan [8] proposes the first work on predicting SPARQL query execution time by utilizing machine learning techniques. In the work, multiple regression using SVR is adopted. The evaluation is performed using benchmark queries on an open source triple store Jena TDB[1]. 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. Our work draws idea from this work and improves it by largely reducing the computation time.

3 Preliminaries

3.1 SPARQL Query

A SPARQL query can be represented as a graph structure, the SPARQL graph [7]. 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 [11]:

(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 widely 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 $\mathbf{x}_i$ [12].

Support Vector Regression 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 [13]. 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:

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

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

where parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyper-plane 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 [17].

K-Nearest Neighbors is a non-parametric classification and regression method [3]. 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 SPARQL queries [8]. By training the k-NN model, the predicted query time can be calculated by the average time of its k nearest neighbors.

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

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

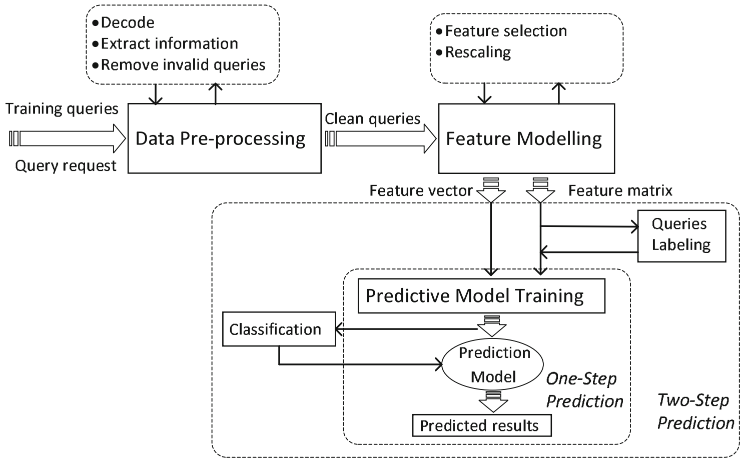

Fig. 1. Steps for query performance prediction

4 SPARQL Query Performance Prediction

Our prediction process consists of four main phases, namely *Data Pre-Processing*, *Feature Modeling*, *Predictive Model Training* and *Prediction* (one-step and two-step) (Fig. 1). Both training and new requested queries are cleaned in the *Data Pre-Processing* phase, valid queries are extracted during this phase. In the *Feature Modeling* phase, queries are represented as a set of features. In the *Predictive Model Training* phase, predictive models are derived from the training queries with observed query performance metrics. In the *Prediction* phase, trained predictive models are used to predict the performance of a new issued query. Compared to one-step prediction, the two-step prediction has labeling before predictive model training and classification step before prediction. We focus on discussion of feature modeling in Sect. 4.1 and describe the predictive models training and two-step prediction in Sect. 4.2. We ignore the description of data pre-processing due to the space constraint.

4.1 Feature Modeling

In order to utilize machine learning algorithms for SPARQL query performance prediction, we transform the SPARQL query into vector representation where each value in a vector is regarded as a feature instance of a query. The performance of prediction highly depends on how much information the features can represent the data. In this study, we use only static, compile time features that are extracted prior to execution. The algebra and BGP features are obtained by parsing the query text (Sects. 4.1.1 and 4.1.2). The hybrid features are generated by applying a selection algorithm on the algebra and BGP features (Sect. 4.1.3). We concatenate the feature vectors of a set of training queries and form a feature matrix as the input of learning algorithms.

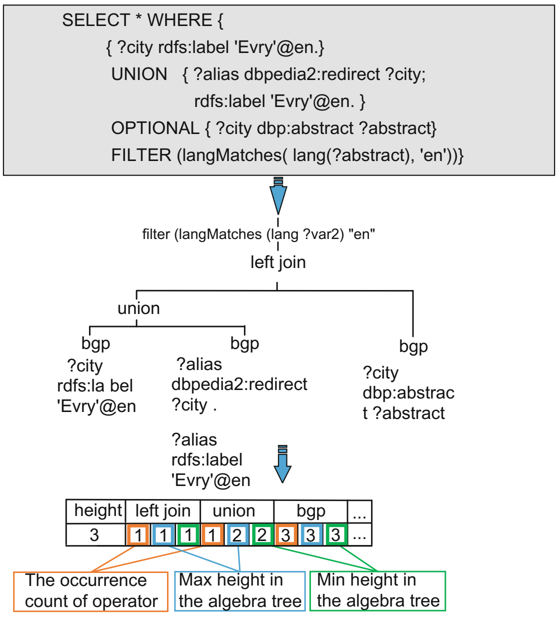
SELECT * WHERE {
{ ?city rdfs:label 'Evry'@en.}
UNION { ?alias dbpedia2:redirect ?city;
rdfs:label 'Evry'@en. }
OPTIONAL { ?city dbp:abstract ?abstract}
FILTER (langMatches(lang(?abstract), 'en'))}
filter (langMatches (lang ?var2) "en"
left join
union
bgp
?city rdfs:la bel 'Evry'@en
bgp
?alias dbpedia2:redirect ?city .
?alias rdfs:label 'Evry'@en
bgp
?city dbp:abstrac t ?abstract
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 modelling on example query

4.1.1 Algebra Feature

The algebra of a SPARQL query can be presented as a tree where the leaves are BGPs and nodes are operators presented hierarchically. The parent of each node is the parent operator of current operator. We traverse the tree to construct 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 the algebra tree, $maxh_i$ and and $minh_i$ are opt_i's maximum height and minimum height in the algebra tree, respectively. We then concatenate all the tuples sequentially to form a vector. We further insert the tree's height at the beginning of the vector. Figure 2 illustrates an example of algebra feature modelling.

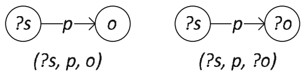

Fig. 3. Example triple patterns

4.1.2 BGP Feature

Algebra features take occurrences and some hierarchical information of operators into consideration, but fail to represent BGPs, the most widely used subset of SPARQL queries [4]. To represent BGPs, we propose to build BGP features. We examine that BGPs consist of sets of triple patterns (Sect. 3) thus can also be represented as tree structure. But we choose not to use similar transformation approach (i.e., record occurrences and maximum/minimum heights) as in algebra feature modelling. Instead, we propose a new way to transform BPGs to vector representation.

Specifically, we leverage the edit distance between graphs to build BGP features because it can capture complete information of a BGP graph. Figure 3 illustrates the graph representation of two triple patterns *(?s, p, o)* and *(?s, p, ?o)* where the *subject*(s) and *object*(o) are nodes and *predicate*(p) are edges. The question mark indicates that the corresponding component is a variable. However, it is hard to tell the differences between the two graphs, as they are structurally identical. To address this problem, we propose to map the eight types of triple patterns to eight structurally different graphs, as shown in Fig. 4(left). The black circles are inner conjunction nodes. To exemplify, we model the triple patterns of BGPs in the example query in Fig. 2, to a graph, which is depicted in Fig. 4(right). The black rectangles are outer conjunction nodes.

We then calculate the Graph Edit Distance (GED)[2] between the graph of a query q and graphs of some representative queries and regard each distance as an instance of a feature. Thus we obtain a n-dimensional feature vector for q, where n is the number of representative queries. We choose to use the 18 valid out of

[2] Graph edit distance 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.

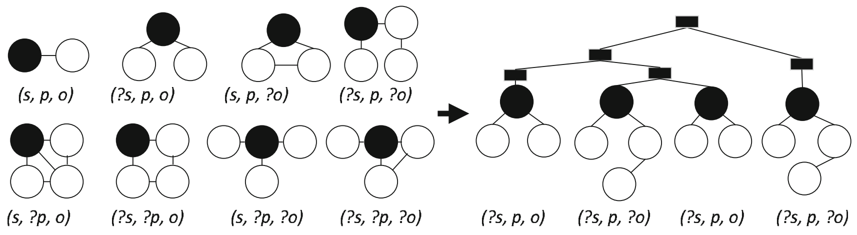

Fig. 4. Mapping triple patterns to graphs. Left: eight types of triple patterns are mapped to eight structurally different graphs. Right: mapping example query in Fig. 2 to a graph.

22 templates from DBPSB benchmark [10] to generate representative queries. We build the graph for each of the 18 queries and compute the GED between q and these graphs. Thus we obtain a 18-dimension feature vector for q.

4.1.3 Hybrid Feature

We build hybrid feature vector for q by selecting the most predictive features based on the algebra and BGP features. Most feature selection approaches rank the candidate features (often based on their correlations) and use this ranking to guide a heuristic search to identify the most predictive features. In this paper, we use a similar forward feature selection algorithm, but we choose the contribution to overall prediction performance as the heuristic. The algorithm starts with building predictive model (we use k-NN as the predictive model here) using a small number of features and iteratively build more complex and accurate model by using more features. In each iteration, a new feature is tested and added to the feature set. If it improves the overall prediction performance, the feature is selected. Otherwise, it is removed from the feature set. Finally, we simply consider the completion of traversing all features as the stopping condition. In this work, we considers BGP features first and then select features from algebra features. The output of the algorithm is the list of selected features that form the feature vector for each query.

4.2 Prediction

We propose two prediction processes, namely one-step prediction and two-step prediction. In the one-step prediction, feature vector of a new query is input into the trained predictive model obtained in the predictive model training phase. The output is the predicted value of the query performance metrics. The two-step prediction differs with one-step prediction by adding classification step. We present the predictive models used in this work in Sect. 4.2.1 and describe how we do two-step prediction in Sect. 4.2.2.

4.2.1 Predictive Models

We choose two regression approaches SVR and k-NN regression in this work (Sect. 3.2). The models are trained with the actual query performance of training queries and then be used to estimate the performance of a new issued query. Both models require the features vector-represented. We compare several variations of these two models in this work. The description is as follows.

SVR. Four commonly used kernels are considered in our model: 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: $K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\gamma\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j) + r$

k-NN. We apply three variations of k-NN regression by considering different weighting methods to the neighbors.

- *Average.* We assign equal weights to each of the k nearest neighbors and get the average of their elapsed time as the predicted time:

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

  where t_i is the elapsed time of the i^{th} nearest query.
- *Power.* The weights in *Power* is the power value of weighting scale α. The predicted query time is calculated as follows:

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

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

$$t_Q = \frac{1}{k}\sum_{i=1}^{k}(e^{-d_i*\beta} * t_i) \tag{5}$$

  where d_i is the distance between target query and its i^{th} nearest neighbor.

All the scaling parameters are chosen through 5-fold cross-validation.

4.2.2 Two-Step Prediction

We observe that the one-step prediction, where all the training data are fed into a single predictive model, gives undesirable performance. A possible reason is the fact that our training dataset has queries with various different elapsed time ranges. Fitting a curve for such irregular data points is often inaccurate. Then we propose a two-step prediction process, where we split queries according to their elapsed time and train different predictive models. Specifically, we firstly put the training queries in four bins, namely *short*, *medium short*, *medium*, and *long*. The time ranges in these four bins are <0.1 s, 0.1 to 10 s, 10 to 3,600 s, and >3,600 s respectively. We correspondingly label all the training queries with these four labels. Then we train four predictive models and one for each bin (or class). When a new query q arrives, we perform classification for q and obtain its label (or class). Here we use Support Vector Machine (SVM) as classification algorithm as it is the mostly used classification algorithm. Then we use the trained predictive model for the class that q is labelled to predict q's performance.

5 Experiments

5.1 Setup

Data. We used real world queries gathered from USEWOD challenge[3], which provides query logs from DBPedia's SPARQL endpoint[4] (DBpedia3.9). We randomly chose 10,000 valid queries in our prediction evaluation. Then these queries were executed 11 times as suggested in [7], including the first time as cold stage, and the remaining 10 times as the warm stage. Finally, we split the collection to training and test sets according to the 4:1 tradition. We set up a local mirror of DBpedia3.9 English dataset to execute the queries.

System. The backing system of our local triple store is Virtuoso 7.2, installed on 64-bit Ubuntu 14.04 Linux operating system with 32 GB RAM and 16 CPU. All the machine learning algorithms are performed on a PC with 64-bit Windows 7, 8 GB RAM and 2.40 GHZ Intel i7-3630QM CPU.

Implementation. We used SVR for kernel and linear regression available from LIBSVM [5]. k-NN and weighted k-NN regression was designed and implemented using Matlab. The algebra tree used for extracting algebra features was parsed using Apache Jena-2.11.2 library, Java API. Graph edit distance was calculated using the Graph Matching Toolkit[5].

Evaluation Metric. We followed the suggestion in [2] and used the *mean relative error* as our prediction metric:

$$relativeerror = \frac{1}{N}\sum_{i=1}^{N}\frac{|actual_i - estimate_i|}{actual_{mean}} \tag{6}$$

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 $actual_i$.

[3] http://usewod.org/.
[4] http://dbpedia.org/sparql/.
[5] http://www.fhnw.ch/wirtschaft/iwi/gmt.

Table 1. Relative error of elapsed time prediction (one-step)

	Elapsed time (cold)	Elapsed time (warm)
SVR-Linear	99.69 %	97.59 %
SVR-Polynomial	99.46 %	97.33 %
SVR-RadialBasis	99.74 %	97.86 %
SVR-Sigmoid	99.68 %	97.57 %
k-NN ($k = 1$)	21.94 %	20.89 %

5.2 Models Comparison

We compared the Linear SVR and SVR with three kernels, namely Polynomial, Radial Basis and Sigmoid with k-NN when $k = 1$. The feature model used in the experiments was the hybrid feature. Table 1 shows the performance of the four models in one-step prediction. SVR models perform poorer than k-NN. We investigate this phenomenon and find two possible reasons. One is that the elapsed 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 that we use mean of actual values in Eq. 6, 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.

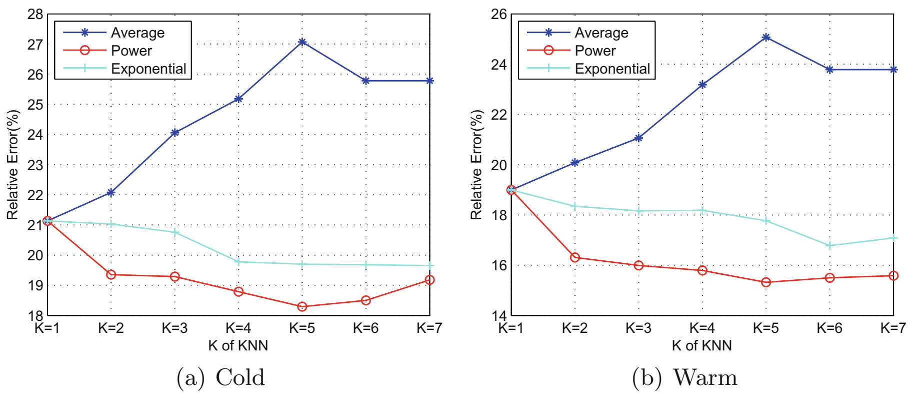

Fig. 5. Performance comparison of different weighted k-NN model (one-step)

We evaluated three weighting schemes for k-NN regression discussed in Sect. 4.2.1, namely *Average*, *Power* and *Exponential*. From Fig. 5 we observe that the power weighting gives 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 weighting is the worst weighting method for our data. Exponential weighting does not perform as well as we expected although it

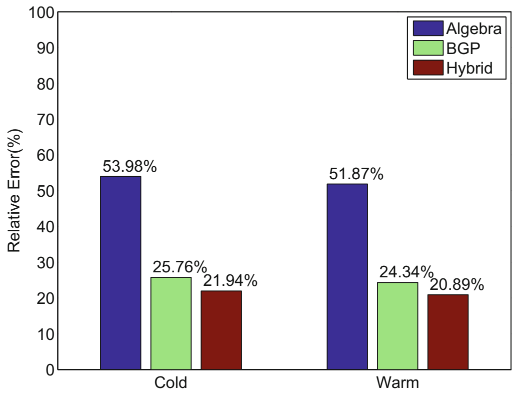

Fig. 6. Feature model selection (one-step)

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.

We also compared the three feature models: *Algebra*, *BGP* and *Hybrid*. Figure 6 shows the prediction performance of elapse time on both warm and cold stages. The hybrid feature performs the best and the BGP feature performs better than the algebra feature. Thus we chose hybrid features in following evaluations.

5.3 Performance of Two-Step Regression

We used SVM for the classification task and achieved accuracy of 98.36 %, indicating that we can accurately predict the time range. Table 2 shows the result of two-step prediction comparison between k-NN and SVR-Polynomial on both warm and cold stage. It shows that SVR regression model still does not perform desirably. It also shows the two-step prediction performs better than one-step prediction. In Fig. 7 we compare one-step and two-step prediction on elapsed time in warm stage using log-log plotting.

Table 2. Relative errors (%) of two-step prediction with k-NN and SVR. In the parentheses are the values from one-step prediction

Predictive model	Elapsed time (cold)	Elapsed time (warm)
5-NN ($\alpha = 0.3$)	11.06 (21.94)	9.81 (20.89)
SVR-Polynomial	22.39	20.30

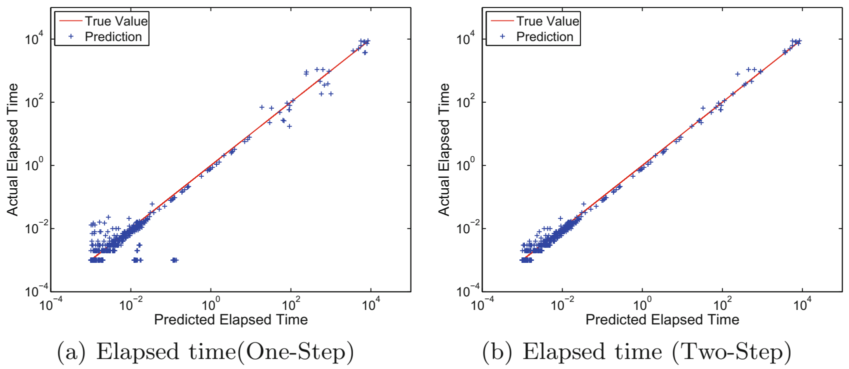

Fig. 7. Elapsed time prediction fitting in warm stage (log-log)

5.4 Comparison to the State-of-the-Art

We compare the approach in the work [8] with our approach, as it is the only work that exploits machine learning algorithms to predict SPARQL query. Table 3 shows the result of comparison on warm stage querying. The training time includes feature modelling, clustering and classification for work in [8]. The first part takes the most time because the calculation of GED for all training queries is time-consuming. In our approach (Sect. 4.1.2), we reduce the GED calculation drastically. But this calculation still takes most time in the prediction process. The time gap of training process between ours and the approach in [8] will be enlarged when more training queries are involved because their approach takes squared time. We do not have clustering process, which further reduces the time used. Our approach also shows better prediction performance with lower relative error for the prediction metric.

Table 3. Comparison to the state-of-the-art work. Training times for 1000 queries (Time1k) are compared as well as the relative errors for elapsed time.

	Models	Time1k	Relative error
Ours	SVM + Weighted k-NN	51.36 s	9.81 %
[8]	X-means + SVM + SVR	1548.45 s	14.39 %

6 Discussions and Conclusion

In this section, we first discuss some observations and issues of this work. Then we conclude this paper.

Plan Features. There are two obstacles for using query plan as features in our work. Firstly, this information is based on the cost model estimation, which has been proven ineffective [2,6]. Secondly, most of the open source triple stores fail to provide explicit query plans. Thus we turn to choose structure-based features that can be obtained directly from query texts. 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 limited features we can acquire.

Training Size. Larger size of the training data would lead to better prediction performance. 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 elapsed time of a large collection of queries. That is the possible reason why many other works only use small size 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 will consider to further enlarge the size of our query set to cover more various queries in the future.

Dynamic vs Static Data. In dynamic query workloads, the queried data is updated. Therefore, the prediction might perform poorly due to lack of update of the 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 to make prediction more available for continuous retraining which reflects recently executed queries.

To conclude, in this paper, we build feature vectors for SPARQL queries by exploiting the syntactic and structural characteristics of the queries. We observe that k-NN performs better than SVR on predicting the elapsed time of real-world SPARQL queries. The proposed two-step prediction performs better than one-step prediction because it considers the broad range of observed elapsed time. The prediction in the warm stage is generally better than in the cold stage. We identify the reason comes from same structured queries because many queries are issued by programmatic users, who tend to issue queries using query templates. Our work is on static data and we will consider dynamic workload in the future. 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: Proceedings of the 14th International Conference on Extending Database Technology (EDBT 2011), Uppsala, pp. 449–460, March 2011
2. Akdere, M., Çetintemel, U., Riondato, M., Upfal, E., Zdonik, S.B.: Learning-based query performance modeling and prediction. In: Proceedings of the 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, pp. 390–401, April 2012

3. Altman, N.S.: An introduction to kernel and nearest-neighbor nonparametric regression. Am. Stat. **46**(3), 175–185 (1992)
4. Bursztyn, D., Goasdoué, F., Manolescu, I.: Optimizing reformulation-based query answering in RDF. In: Proceedings of the 18th International Conference on Extending Database Technology (EDBT 2015), Brussels, pp. 265–276, March 2015
5. Chang, C., Lin, C.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 27 (2011)
6. 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: Proceedings of the 25th International Conference on Data Engineering (ICDE 2009), Shanghai, pp. 592–603, March 2009
7. Gubichev, A., Neumann, T.: Exploiting the query structure for efficient join ordering in SPARQL queries. In: Proceedings of the 17th International Conference on Extending Database Technology (EDBT 2014), Athens, pp. 439–450, March 2014
8. Hasan, R.: Predicting SPARQL query performance and explaining linked data. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 795–805. Springer, Heidelberg (2014). doi:10.1007/978-3-319-07443-6_53
9. Li, J., König, A.C., Narasayya, V.R., Chaudhuri, S.: Robust estimation of resource consumption for SQL queries using statistical techniques. VLDB Endow. (PVLDB) **5**(11), 1555–1566 (2012)
10. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.N.: Usage-centric benchmarking of RDF triple stores. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence, Toronto, July 2012
11. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16 (2009)
12. Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2011)
13. Smola, A., Vapnik, V.: Support vector regression machines. Adv. Neural Inf. Process. Syst. **9**, 155–161 (1997)
14. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL basic graph pattern optimization using selectivity estimation. In: Proceedings of the 17th International World Wide Web Conference (WWW 2008), Beijing, pp. 595–604, April 2008
15. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P. A.: Heuristics-based query optimisation for SPARQL. In: Proceedings of the 15th International Conference on Extending Database Technology (EDBT 2012), Uppsala, pp. 324–335, March 2012
16. Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigümüs, H., Naughton, J.F.: Predicting query execution time: are optimizer cost models really unusable? In: Proceedings of the 29th International Conference on Data Engineering (ICDE 2013), Brisbane, pp. 1081–1092, April 2013
17. 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. Knowl. Inf. Syst. **14**(1), 1–37 (2008)