A Machine Learning Approach to SPARQL Query Performance Prediction

Rakebul Hasan, Fabien Gandon

A Machine Learning Approach to SPARQL Query Performance Prediction

Rakebul Hasan and Fabien Gandon
INRIA Sophia Antipolis, Wimmics, 2004 route des Lucioles - B.P. 93,
06902 Sophia-Antipolis Cedex, France,
{hasan.rakebul,fabien.gandon}@inria.fr

***Abstract*—In this paper we address the problem of predicting SPARQL query performance. We use machine learning techniques to learn SPARQL query performance from previously executed queries. Traditional approaches for estimating SPARQL query cost are based on statistics about the underlying data. However, in many use-cases involving querying Linked Data, statistics about the underlying data are often missing. Our approach does not require any statistics about the underlying RDF data, which makes it ideal for the Linked Data scenario. We show how to model SPARQL queries as feature vectors, and use *k*-nearest neighbors regression and Support Vector Machine with the nu-SVR kernel to accurately predict SPARQL query execution time.**

I. Introduction

In the recent years, we have seen a sharp growth of publishing Linked Data from community driven efforts, governmental bodies, social networking sites, scientific communities, and corporate bodies [1]. Data publishers from these different domains publish their data in an interlinked fashion[1] using RDF data model and provide SPARQL endpoints to enable querying their data. This global space presents tremendous potential for large-scale data integration over cross domain data to support a new generation of intelligent application [2]. In this context, it increasingly important to develop efficient ways of querying Linked Data [3]. Central to this problem is knowing how a query would behave prior to executing the query [4]. This enables us to adjust our queries accordingly.

Current generation of SPARQL query cost estimation approaches are based on data statistics and heuristics. Statistics-based approaches have two major drawbacks in the context of Linked Data [5]. First, the statistics (e.g histograms) about the data are often missing in the Linked Data scenario because they are expensive to generate and maintain. Second, due to the graph-based data model and schema-less nature of RDF data, what makes effective statistics for query cost estimation is unclear. Heuristics-based approaches generally do not require any knowledge of underlying data statistics. However, they are based on strong assumptions such as considering queries of certain structure less expensive than others. These assumptions may hold for some RDF datasets and may not hold for others.

We take a rather pragmatic approach to SPARQL query cost estimation. We learn SPARQL query performance metrics from already executed queries. Recent work [6], [7], [8] in database research shows that database query performance metrics can be accurately predicted without any knowledge of data statistics by applying machine learning techniques on the query logs of already executed queries. Similarly, we apply machine learning techniques to learn SPARQL query performance metrics from already executed queries. SPARQL query optimizers can use these predictions to choose among alternative query plans. In addition, they can be also used for workload allocation or to meet specific QoS targets [8]. We consider query execution time as the query performance metric in this paper.

The rest of this paper is structured as following: in section II we briefly introduce the RDF data model and the SPARQL query language. In section III we describe our approach to modeling SPARQL query execution. In section IV we present our experiments and results. In section V we describe the related work. Finally, in section VI we conclude and outline the future work.

II. Preliminaries

Before we present our approach to query performance prediction, we briefly introduce the RDF data model and the SPARQL query language. For more detailed introduction to RDF and SPARQL, please refer to the cited W3C specification documents.

The Resource Description Framework (RDF) data model is a W3C recommended standard for representing information about resources in the World Wide Web [9]. RDF is a graph-based data model where vertices represent entities and edges represent relationships between entities.

Definition 1 (*RDF graph*)**.** *Let I be the set of IRIs, L be the set of literals, and B be the set of blank nodes. An RDF triple (s,p,o) is a member of the set* $(I \cup B) \times I \times (I \cup L \cup B)$*. An RDF graph is a set of RDF triples. For an RDF triple (s,p,o), the element s is called subject, the element p is called predicate, and the element o is called object.*

SPARQL is the W3C recommended query language for RDF. As the SPARQL 1.1 specification describes [10], SPARQL query solving is based around graph pattern matching. SPARQL queries allow specifying sets of *triple patterns* know as basic graph patterns. *Triple patterns* are similar to RDF triples but the subject, predicate, and object can be variables. A basic graph pattern may match a subgraph from the RDF data and substitute the variables by RDF terms from that matched subgraph. The native SPARQL query engines perform a series of steps to execute a query [10]. First,

[1]See http://richard.cyganiak.de/2007/10/lod/ for a graph linking these datasets.

parsing the query string into an abstract syntax form. Next, transforming the abstract syntax to *SPARQL abstract query*. Finally, optimizing and evaluating the *SPARQL abstract query* on an RDF dataset.

Definition 2 (*SPARQL abstract query*)**.** *A SPARQL abstract query is a tuple (E, DS, QF) where E is a SPARQL algebra expression, DS is an RDF dataset, QF is a query form.*

The algebra expression E is evaluated against *RDF graphs* in the RDF dataset DS. The query form QF (*SELECT*, *CONSTRUCT*, *ASK*, *DESCRIBE*) uses the solutions from pattern matching to provide result sets or *RDF graphs*. The algebra expression E includes graph patterns and operators such as *FILTER*, *JOIN*, and *ORDER BY* [2]. SPARQL allows forming graph patterns by combining smaller patterns: basic graph patterns, group graph patterns, optional graph patterns, alternative graph patterns, and patterns on named graphs. A basic graph pattern contains a set of *triple patterns*.

Definition 3 (*Triple pattern*)**.** *A triple pattern is a member of the set: $(T \cup V) \times (I \cup V) \times (T \cup V)$. The set of RDF terms T is the set $I \cup L \cup B$. The set V is the set of query variables where V is infinite and disjoint from T.*

A group graph pattern combines all other types of graph patterns. An optional graph pattern contains a graph pattern which is optional to match for a query solution. Alternative graph patterns provide a means to take union of the solutions of two or more graph patterns. Patterns on named graphs provide a means to match patterns against graphs when querying a collection of graphs. The outer-most graph pattern in a SPARQL query is known as the query pattern. A query pattern is a group graph pattern.

III. Modeling SPARQL Query Execution

As in the common machine learning approaches, our query performance prediction approach includes two main phases: training and testing. In the training phase, we derive a prediction model from a training dataset containing previously executed queries and the observed performance metric values (execution times) for those queries. We represent the queries as feature vectors. The goal of the training phase is to create an accurate model that maps the feature vectors to the performance metric data points. We use regression for this purpose. We define feature vectors, $x = (x_1, x_2, ...x_n)$, where $x \in \mathbb{R}^n$ and each x_i is a SPARQL query feature. The performance metric, query execution time, is the variable y. We learn a function $f(x) = y$, *i.e.* the function maps a feature vector x to y, using regression. We provide more details on the types of regression we use in section IV-C. In the testing phase, we use the trained model to predict query performance metric values for unforeseen queries. Additionally, we tune our model parameters using cross-validation. We use two types of query features: SPARQL algebra features and graph pattern features.

A. SPARQL Algebra Features

We use the frequencies of all the SPARQL algebra operators except the *SLICE* operator as query features. The *SLICE* operator is the combination of *OFFSET* and *LIMIT* SPARQL keywords. We use the sum of all the *SLICE* operator cardinalities appearing in the algebra expression as the feature representing the *SLICE* operator. In addition, we use two more features: the depth of the algebra expression tree and the number of *triple patterns*. Figure 1 shows an example of extracting the SPARQL algebra features vector from a SPARQL query. First we transform a query into an algebra expression tree. Then we extract the features and represent the query as a feature vector. We use the Jena ARQ SPARQL parser[3] to transform query strings to SPARQL algebra expressions.

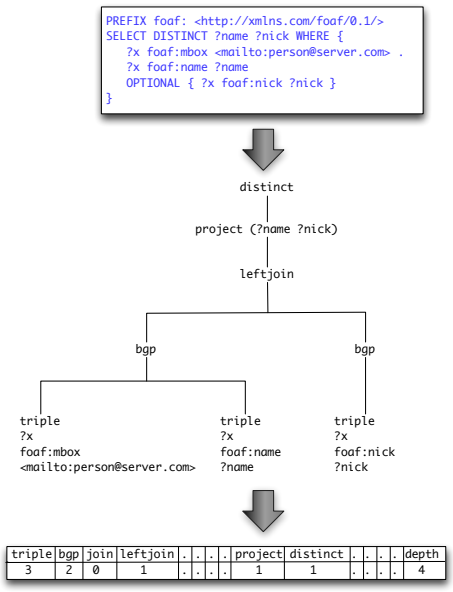


Fig. 1. Extracting SPARQL algebra features from a SPARQL query.

B. Graph Pattern Features

The SPARQL algebra features do not represent graph patters appearing in SPARQL queries. Transforming graph patterns to vector space is not trivial because the space is infinite. To address this, we create a query pattern vector representation relative to the query patterns appearing in the training data. First, we cluster the structurally similar query patterns in the training data into K_{gp} number of clusters. The query pattern in the center of a cluster is the representative of query patterns in that cluster. Second, we represent a query pattern as a K_{gp} dimensional vector where the value of a dimension is the structural similarity between that query pattern and the corresponding cluster center query pattern. To compute the structural similarity between two query patterns, we first construct two graphs from the two query patterns, then compute the graph edit distance [11] between these two graphs. We compute the structural similarity by inverting the edit distance. The graph edit distance between two graphs is the minimum amount of distortion needed to transform one graph to another. The minimum amount of distortion is the sequence of edit operations with minimum cost. The edit operations are deletions, insertions, and substitutions of nodes and edges. A well known method for computing graph edit distance is using the A* search algorithm to explore the state space of possible

[2]Algebra operators: http://www.w3.org/TR/sparql11-query/#sparqlAlgebra

[3]https://jena.apache.org/documentation/query/algebra.html

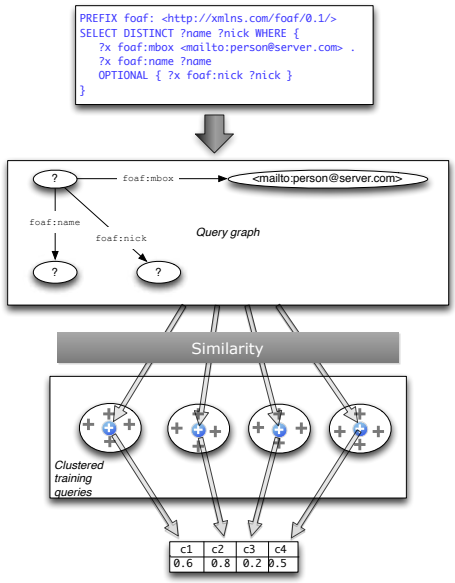


Fig. 2. Example of extracting graph pattern features.

mappings of the nodes and edges of the source graph to the nodes and edges of the target graph. However, the computational complexity of this edit distance algorithm is exponential in the number of nodes of the involved graphs, irrespective of using A* search with a heuristic function to govern the tree traversal process. Therefore we use the polynomial time suboptimal solution of graph edit distance that Riesen and Bunke [12], [13] propose. The computational complexity this polynomial time suboptimal solution is O $\left(n^3\right)$, where n is the number of nodes of the involved graphs. To construct a graph from a query pattern, we take all the *triple patterns* in the query pattern and construct a graph from these *triple patterns*. As in *RDF graphs*, the subject and the object of a *triple pattern* represent vertices of the graph and the predicate represents an edge of the graph. After constructing such a graph, we replace the labels of vertices and edges representing variables by a fixed symbol - the symbol '?'. This ensures that the graph has separate vertices and edges for each variable appearing in the query but an unified labeling. We call such a graph a *query graph*. Figure 2 shows an example of extracting graph pattern features for a query. First step shows the constructed *query graph*. The clustered queries box shows the clusters of training queries where each circle is a cluster of *query graphs* with their cluster centers shown in blue color. We use the *k*-mediods [14] clustering algorithm to cluster the *query graphs* of training data. We use *k*-mediods because it chooses data points as cluster centers and allows using an arbitrary distance function. As we mention before, we use the suboptimal graph edit distance algorithm as the distance function for *k*-mediods. For the K_{gp} dimensional vector representation of query pattern, we compute the structural similarity between a *query graph* p_i and the k^{th} cluster center *query graph* $C(k)$ as below:

$$sim(p_i, C(k)) = \frac{1}{1 + d(p_i, C(k))} \quad (1)$$

The term $d(p_i, C(k))$ is the graph edit distance between *query graphs* p_i and $C(k)$. This formulation gives us a similarity score within the range of 0 to 1. A similarity score of 0 being the least similar and a score of 1 being the most similar. The extracted feature vector in figure 2 shows the computed similarity values using equation 1 for the example query.

IV. Experiments and Results

We use the DBPSB benchmark [15] queries on a Jena-TDB triple store [16] to evaluate our approach. DBPSB includes 25 query templates which cover most commonly used SPARQL query features in the queries sent to DBPedia[4]. We generate our training, validation, and test queries from these query templates. We use query execution time as the query performance metric. The details of our experimental setup is as below.

A. Triple Store and Hardware

We use Jena-TDB 1.0.0 as a triple store. We allow Jena-TDB to use 16 GB of memory. We execute all the queries in a commodity server machine with a 4 core Intel Xeon 2.53 GHz CPU, 48 GB system RAM, and Linux 2.6.32 operating system.

B. Datasets

As the RDF dataset, we use the DBpedia 3.5.1 dataset with 100% scaling factor – provided by the DBPSB benchmark framework. We generate our training, validation, and test queries from the 25 DBPSB query templates. To generate queries, we assign randomly selected RDF terms from the RDF dataset to the placeholders in the query templates. We generate 205 queries for each template and then execute them to build our training, validation, and test datasets. Before executing the queries, we restart the triple store to clear the caches. Then we execute total 125 queries in our warm-up phase to measure query performance under normal operational conditions. Our warm-up queries include the first 5 queries from each of the 25 templates. To generate the training queries, we execute the next 120 queries from each template and take the first 60 queries for each template which return at least 1 result and finish executing within a reasonable time. We specify a 300 second timeout for a query execution. We follow the same process to generate 20 validation queries from the next 40 queries for each template and 20 test queries from the last 40 queries for each template. In this setting, none of the queries from template 2, 16, and 21 returned any result. All the queries from template 20 were interrupted because of timeout. This process resulted 1260 training queries, 420 validation queries, and 420 test queries. We execute each of these training, validation, and test queries 5 times and record the average execution time in milliseconds (ms) for each query. Figure 3 shows the average, minimum, and maximum execution times for the queries from our test dataset. As the figure shows, we have a mix of long and short running queries. Queries belonging to templates 4, 10, and 24 have more than 1000 ms of average execution time. The queries from the other query templates have less than 1000 ms of average execution time.

[4]http://dbpedia.org

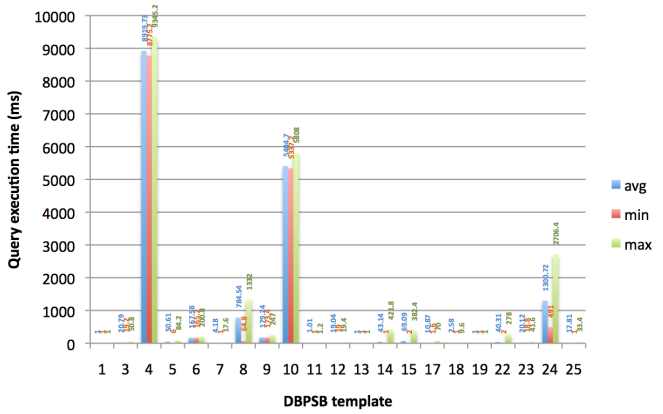


Fig. 3. Average, minimum, and maximum execution times for the queries belonging to different query templates in the test dataset.

C. Prediction Models

To predict query execution time, we experiment with two regression models. We first experiment with Weka's [17] implementation of *k*-nearest neighbors (*k*-NN) regression [18], [19]. The *k*-NN algorithm predicts based on the closest training data points. It uses a distance function to compute these closest data points. We use Euclidean distance as the distance function in our experiments. For predictions, we use the weighted average of the k nearest neighbors - weighted by the inverse of the distance from the querying data point. This ensures that the nearby neighbors contribute more to the prediction than the faraway neighbors. We use the *k*-dimensional tree (*k*-d tree) [20] data structure to compute the nearest neighbors. For N training samples, *k*-d tree can find the nearest neighbor of a data point with $O(log\, N)$ operations. We also experiment with the libsvm [21] implementation of Support Vector Machine (SVM) with the nu-SVR kernel for regression [22]. The approach in SVM regression is to map the features to a higher dimensional space and perform a regression in that space. The predictions in SVM are based on a subset of data points known as support vectors.

D. Evaluation Metrics

We use the coefficient of determination, denoted as R^2, to evaluate our models. R^2 is a widely used evaluation measure for regression. R^2 measures how well future samples are likely to be predicted. We compute R^2 as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

The vectors y and $\hat{y}$ represent the actual values and predicted values respectively for n queries. $\bar{y}$ is the mean of actual values. An R^2 score close to 1 indicates near perfect prediction. R^2 scores however can be misleading in many cases. As R^2 depends on the scale and statistical characteristics of the whole dataset, it can have low errors even if the predictions have high errors [8]. Therefore we use another evaluation metric, root mean squared error (RMSE), as our error metric:

$$RMSE(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

E. Predicting Query Execution Time

We show the results of our experiments in Figure 4 and Figure 6. The results include R^2 and RMSE values using *k*-NN and SVM with SPARQL algebra features and graph pattern features. Below we discuss these results.

1) Predicting with SPARQL Algebra Features: For *k*-NN with SPARQL algebra features, we select k, the number of neighbors, by cross-validation. As Table I shows, different values of k do not have any effect on RMSE and R^2 on our validation dataset. Therefore we select $k = 2$. We achieve an R^2 value of 0.96645 and an RMSE value of 395.5125 on the test dataset using *k*-NN with SPARQL algebra features. Figure 4(a) shows the comparison between predicted and actual execution times using *k*-NN with SPARQL algebra features. Figure 4(b) shows that the queries from template 15 has the highest RMSE. The execution time for queries from template 15 range from 2 ms to 382.4 ms with an average of 69.09 ms. Because of the high error for queries from template 15, there are overestimated data points in this interval in Figure 4(a).

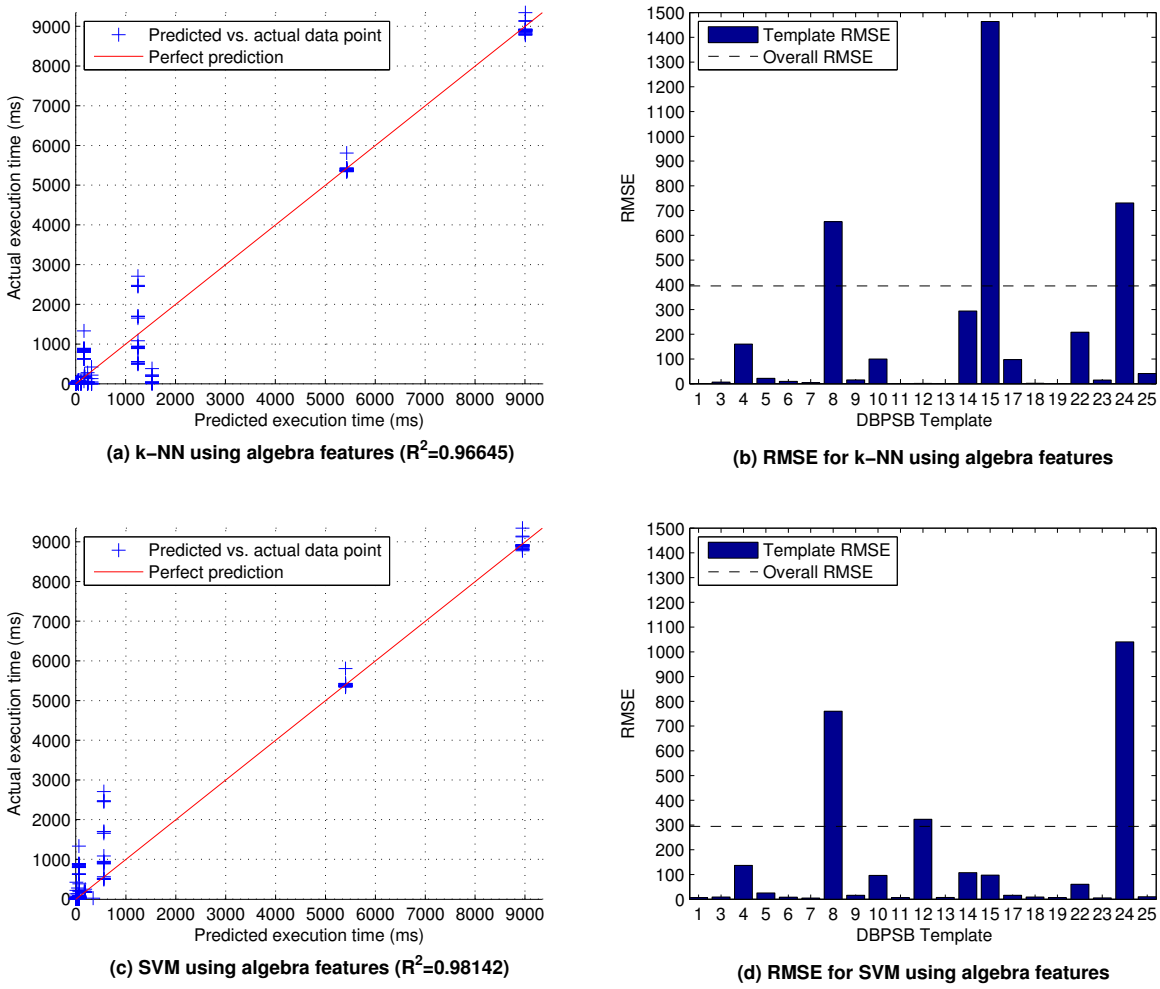
	k=2	*k*=3	*k*=4	*k*=5
RMSE	588.2004	588.2004	588.2004	588.2004
R^2	0.9286	0.9286	0.9286	0.9286

TABLE I. RMSE AND R^2 VALUES FOR DIFFERENT k FOR k-NN ON THE VALIDATION DATASET.

We achieve an improved R^2 value of 0.98142 and a lower RMSE value of 294.3532 on the test dataset using SVM with SPARQL algebra features. Figure 4(c) shows the comparison between predicted and actual execution times using SVM with SPARQL algebra features. Figure 4(d) shows the RMSE values by query template for this model. As the figures show, the error for queries from template 15 decreases. Therefore the overestimated data points in the interval 2 ms to 382.4 ms move towards the perfect prediction line. However, the error for template 8 and 24 slightly increases.

2) Predicting with SPARQL Algebra and Graph Pattern Features: For *k*-NN with SPARQL algebra features and graph pattern features, we have two parameters: the number of clusters K_{gp} and the number of neighbors k. Again we select them by cross-validation. Figure 5(a) shows the RMSE values on the validation dataset for different K_{gp} and k, and Figure 5(b) shows the R^2 values on the validation dataset for different K_{gp} and k. The Figure 5 shows, k again does not have any impact. We get lowest K_{gp} and highest R^2 values at $K_{gp} = 10$ and $K_{gp} = 25$ for all k values. Therefore we select $K_{gp} = 10$ and $k = 2$ for our predictions with *k*-NN on the test dataset. Figure 6(a) and Figure 6(b) shows the prediction results on the test dataset using *k*-NN with $K_{gp} = 10$ and $k = 2$. We get a slightly less R^2 value for this model than *k*-NN with SPARQL algebra features. This is because of the increase in RMSE values for queries from template 9, 17, and 24.

For SVM with SPARQL algebra features and graph pattern features, we select the value of K_{gp} by cross-validation.



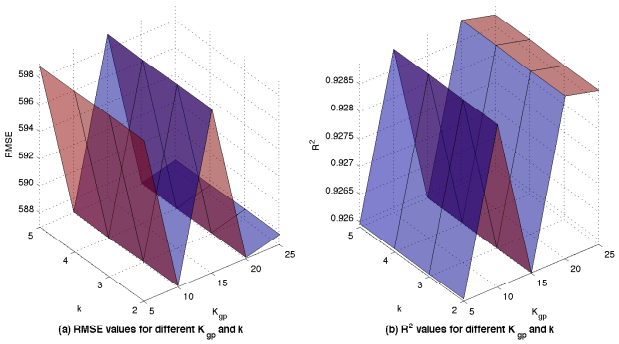
(a) k–NN using algebra features (R^2=0.96645)

(b) RMSE for k–NN using algebra features

(c) SVM using algebra features (R^2=0.98142)

(d) RMSE for SVM using algebra features

Fig. 4. Query execution time predictions with SPARQL algebra features using k-NN (with $k = 2$) and SVM models.



(a) RMSE values for different K_{gp} and k

(b) R^2 values for different K_{gp} and k

Fig. 5. RMSE and R^2 values on the validation dataset for different K_{gp} and k.

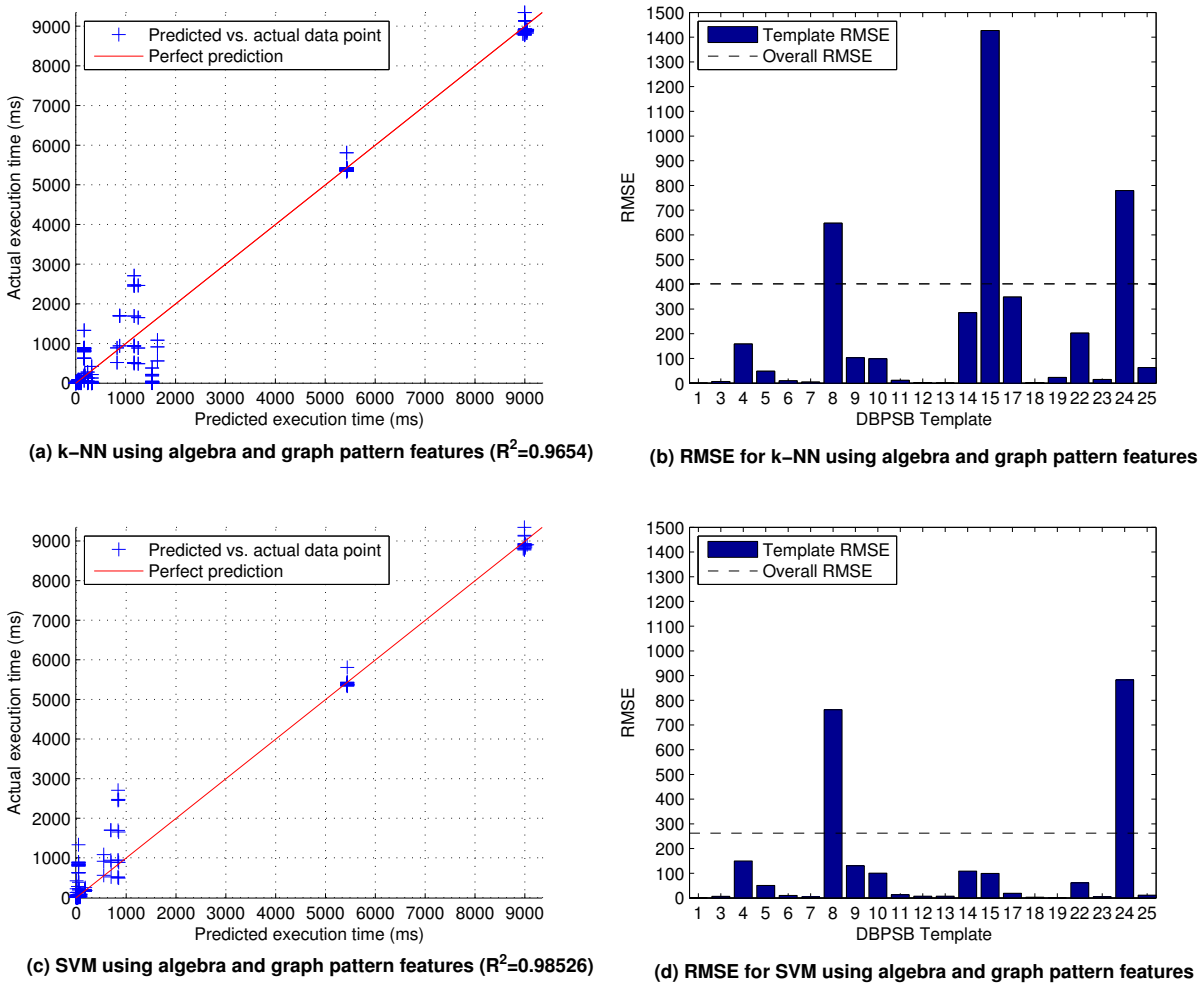
Table II shows RMSE and R^2 values on the validation dataset for different K_{gp} using SVM. We select $K_{gp} = 25$ because it gives use the lowest RMSE value 528.9321 and highest R^2 value 0.9422 on the validation dataset. Figure 6(c) and Figure 6(d) shows the prediction results on the test dataset using SVM with $K_{gp} = 25$. We get the overall best R^2 value 0.98526 and the overall lowest RMSE value 262.1869 with this model. This is an improvement from the SVM with SPARQL algebra features model. The main reason for this is the decrease in RMSE for queries from template 12 and 24.

	K_{gp}=5	K_{gp}=10	K_{gp}=15	K_{gp}=20	K_{gp}=25
RMSE	530.9169	546.7406	547.6764	547.4219	528.9321
R^2	0.9418	0.9383	0.9381	0.9381	0.9422

TABLE II. RMSE AND R^2 VALUES ON THE VALIDATION DATASET FOR DIFFERENT K_{gp} USING SVM.

F. Required Time for Training and Prediction

Table III shows the total training time and average prediction time per query for the models we experimented with. Models with SPARQL algebra features take very low prediction time per query. Training time is also low. Models with



(a) k–NN using algebra and graph pattern features (R^2=0.9654)

(b) RMSE for k–NN using algebra and graph pattern features

(c) SVM using algebra and graph pattern features (R^2=0.98526)

(d) RMSE for SVM using algebra and graph pattern features

Fig. 6. Query execution time predictions with SPARQL algebra features and graph pattern features using k-NN ($K_{gp} = 10$ and $k = 2$) and SVM ($K_{gp} = 25$).

graph pattern features take longer time to train. This is because the training time includes generating the distance matrix using approximated graph edit distance. This process itself takes 3293 seconds on average for 1260 queries. Also it includes the time required to cluster the training queries. However the average prediction time per query using models with graph pattern features is within the limit of 100 milliseconds, which is reasonable especially for query solving over Linked Data. The average prediction time per query using models with graph pattern features increase from the models with only algebra features because of the similarity computations using approximated edit distance. It is important to note that the training phase is an offline process and hence it does not influence query prediction time.

Model	Training time	Avg. prediction time per query
k-NN + algebra	7.14 sec	3.42 ms
SVM+ algebra	26.26 sec	3.53 ms
k-NN + algebra + graph pattern	3300.33 sec	47.25 ms
SVM + algebra + graph pattern	3390.71 sec	98.1 ms

TABLE III. REQUIRED TIME FOR TRAINING AND PREDICTIONS.

V. Related Work

Recent work on predicting database query performance [8], [6], [7] has argued that the cost models used by the current generation query optimizers are good for comparing alternative query plans, but ineffective for predicting actual query performance metrics such as query execution time. These cost models are unable to capture the complexities of modern database systems [8]. To address this, database researchers have experimented with machine learning techniques to learn query performance metrics. Ganapathi *et al.* [6] use Kernel Canonical Correlation Analysis (KCCA) to predict a set of performance metrics. For the individual query elapsed time performance metric, they were able to predict within 20% of the actual query elapsed time for 85% of the test queries. Gupta *et al.* [7] use machine learning for predicting query execution time ranges on a data warehouse and achieve an

accuracy of 80%. Akdere *et al.* [8] study the effectiveness of machine learning techniques for predicting query latency of static and dynamic workload scenarios. They argue that query performance prediction using machine learning is both feasible and effective.

Related to the Semantic Web query processing, SPARQL query engines can be categorized into two categories: SQL-based and RDF native query engines [5]. SQL-based query engines rely on relational database systems storage and query optimization techniques to efficiently evaluate SPARQL queries. They suffer from the same problems mentioned above. Furthermore, due to the absence of schematic structure in RDF, cost-based approaches – successful in relational database systems – do not perform well in SPARQL query processing [5]. RDF native query engines typically use heuristics and statistics about the data for selecting efficient query execution plans [23]. Heuristics-based optimization techniques include exploiting syntactic and structural variations of *triple patterns* in a query [23], and rewitting a query using algebraic optimization techniques [24] and transformation rules [4]. Heuristics-based optimization techniques generally work without any knowledge of the underlying data. Stocker *et al.* [23] present optimization techniques with pre-computed statistics for re-ordering triple patters in a SPARQL query for efficient query processing. However, in many use-cases involving querying Linked Data, statistics are often missing [5]. This makes these statistics-based approaches ineffective in the Linked Data scenario. Furthermore, as in the case of relation database systems, these existing approaches are unable to predict actual query performance metrics such as query execution time for a given configuration.

VI. Conclusion and Future Work

We present an approach to predict SPARQL query execution time using machine learning techniques. We learn query execution times from already executed queries. This approach can be useful where statistics about the underlying data are unavailable We discuss how to model SPARQL queries as feature vectors, and show highly accurate results.

In future, firstly we would like to compare our approach to traditional query cost estimation techniques in the Linked Data scenario – e.g. join order optimization in federated query processing. State of the art Linked Data query processing approach FedX [2] uses variable count selectivity estimation [23] optimization for efficient join ordering of grouped triple pattern execution. We would like to compare our approach to such approaches. Second, we plan to systematically generate training queries for two scenarios: (a) given query logs of real queries (b) given a small set of sample queries. We plan to apply query log mining techniques to systematically generate training queries. Recent [25] work on query log mining shows that the majority of SPARQL queries share some common characteristics. We plan to consider those statistically significant common characteristics in refining training queries from massive query logs and generating training queries from a small set of sample queries. We would also explore how these common characteristics can be used as query features. Third, we would like to investigate online machine learning techniques for our models. Our goal would be to refine our prediction models based on the new predictions and their actual values. Finally, we would like to include load and availability related features. In this direction, we plan to execute the training queries every hour and include features such as time, day, and month. This would help us to model workload patterns for public SPARQL endpoints.

Acknowledgment

This work is supported by the ANR CONTINT program under the Kolflow project (ANR-2010-CORD-021-02).

References

[1] P. Bonatti, A. Hogan, A. Polleres, and L. Sauro, "Robust and scalable linked data reasoning incorporating provenance and trust annotations," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 9, no. 2, pp. 165–201, 2011.

[2] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt, "Fedx: Optimization techniques for federated query processing on linked data," in *Proc. of the 10th International Semantic Web Conference*, ser. ISWC 2011. Springer, 2011.

[3] J. Huang, D. J. Abadi, and K. Ren, "Scalable SPARQL querying of large RDF graphs," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1123–1134, 2011.

[4] O. Hartig and R. Heese, "The sparql query graph model for query optimization," in *Proceedings of the 4th European Conference on The Semantic Web: Research and Applications*, ser. ESWC '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 564–578.

[5] P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, and P. Boncz, "Heuristics-based query optimisation for SPARQL," in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12. New York, NY, USA: ACM, 2012, pp. 324–335.

[6] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ser. ICDE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 592–603.

[7] C. Gupta, A. Mehta, and U. Dayal, "PQR: Predicting query execution times for autonomous workload management," in *Proceedings of the 2008 International Conference on Autonomic Computing*, ser. ICAC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 13–22.

[8] M. Akdere, U. Cetintemel, M. Riondato, E. Upfal, and S. Zdonik, "Learning-based query performance modeling and prediction," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, 2012, pp. 390–401.

[9] "RDF 1.1 concepts and abstract syntax," *W3C recommendation*, 2014.

[10] "SPARQL 1.1 Query Language," *W3C recommendation*, 2013.

[11] H. Bunke and B. Messmer, "Similarity measures for structured representations," in *Topics in Case-Based Reasoning*, ser. Lecture Notes in Computer Science, S. Wess, K.-D. Althoff, and M. Richter, Eds. Springer Berlin Heidelberg, 1994, vol. 837, pp. 106–118.

[12] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image Vision Comput.*, vol. 27, no. 7, pp. 950–959, Jun. 2009.

[13] K. Riesen, S. Emmenegger, and H. Bunke, "A novel software toolkit for graph edit distance computation," in *Graph-Based Representations in Pattern Recognition*, ser. LNCS, W. Kropatsch, N. Artner, Y. Haxhimusa, and X. Jiang, Eds. Springer Berlin Heidelberg, 2013, vol. 7877, pp. 142–151.

[14] L. Kaufman and P. Rousseeuw, "Clustering by means of medoids," in *Statistical Data Analysis based on the L1 Norm*, Y. Dodge, Ed., 1987, p. 405416.

[15] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo, "Dbpedia SPARQL benchmark performance assessment with real queries on real data," in *The Semantic Web ISWC 2011*, ser. LNCS, L. Aroyo *et al.*, Eds. Springer Berlin Heidelberg, 2011, vol. 7031, pp. 454–469.

[16] A. Owens, A. Seaborne, N. Gibbins, and mc schraefel, "Clustered TDB: A clustered triple store for Jena," November 2008.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[18] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[19] N. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[20] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977.

[21] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[22] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to the SMO algorithm for SVM regression," *Neural Networks, IEEE Transactions on*, vol. 11, no. 5, pp. 1188–1193, 2000.

[23] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "SPARQL basic graph pattern optimization using selectivity estimation," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 595–604.

[24] F. Frasincar, G.-J. Houben, R. Vdovjak, and P. Barna, "RAL: An algebra for querying RDF," *World Wide Web*, vol. 7, no. 1, pp. 83–109, 2004.

[25] M. Arias, J. D. Fernndez, M. A. Martnez-Prieto, and P. de la Fuente, "An empirical study of real-world SPARQL queries," *1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011), in Conjunction with WWW 2011*, 2011.