Mathematical and Computational Applications

MDPI

Article

A Recommendation System for Execution Plans Using Machine Learning

Jihad Zahir [1,*] and Abderrahim El Qadi [2]

[1] LRIT-CNRST (URAC No. 29), Faculty of Sciences, Mohammed V University in Rabat, Rabat 10000, Morocco
[2] Team TIM, High School of Technology, Moulay Ismail University in Meknes, Meknes 50050, Morocco; elqadi_a@yahoo.com
* Correspondence: jihad.zahir@gmail.com; Tel.: +212-664-927-421

Academic Editor: Mehmet Ali Ilgın
Received: 13 April 2016; Accepted: 7 June 2016; Published: 15 June 2016

Abstract: Generating execution plans is a costly operation for the DataBase Management System (DBMS). An interesting alternative to this operation is to reuse the old execution plans, that were already generated by the optimizer for past queries, to execute new queries. In this paper, we present an approach for execution plan recommendation in two phases. We firstly propose a textual representation of our SQL queries and use it to build a *Features Extractor module*. Then, we present a straightforward solution to identify query similarity.This solution relies only on the comparison of the SQL statements. Next, we show how to build an improved solution enabled by machine learning techniques. The improved version takes into account the features of the queries' execution plans. By comparing three machine learning algorithms, we find that the improved solution using Classification Based on Associative Rules (CAR) identifies similarity in 91% of the cases.

Keywords: execution plan reuse; CAR; SQL queries; recommendation

1. Introduction

To execute a given SQL query, the query optimizer needs to choose an execution plan for it. An execution plan is a tree such that each node of the tree is a physical operator (e.g., sequential scan, sort, or hash join) [1]. Needless to say, generating execution plans is an "expensive" process for servers. Equivalent results can be obtained by considering different plans, but selecting the less resource consuming plan is crucial. Accordingly, the task of query optimizers is to generate a plan with the sequence of actions that minimizes the consumption of resources. One solution is to reuse existing plans [2], that were generated in the past for previous queries, to execute new similar queries. This solution is based on similarity identification between queries. In our context, two queries are said to be similar if the same execution plan could be used to execute both of them.

In this paper, our main objective is to present a recommendation process that uses similarity identification between SQL queries and machine learning techniques to recommend a previously generated execution plan to the optimizer. We firstly present a straightforward approach that relies only on SQL query statements, and presents an improved approach enabled by machine learning techniques afterwards.

In the improved approach, as we represent our queries in two different spaces, we want to use classification methods to try to discover relationships between these representations. Features of the first representation are derived from the SQL statement of the query, while features of the second representation are extracted from the execution plan. More specifically, we are interested in discovering association rules between the textual features of SQL queries and their execution plans' features. These rules are then used to learn models that recommend execution plans for new queries as shown in Figure 1.

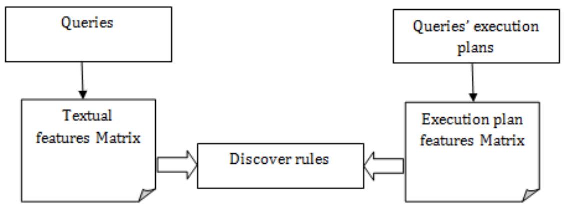

Figure 1. Using association rules to plan recommendation.

The remainder of this paper is organized as follows: In the second section, we give an overview of work related to our study. Next, we describe our approach in the third section, and the features extraction process in the fourth section. The straightforward solution and the solution improved by machine learning techniques are presented in Sections 5 and 6, respectively. Finally, we discuss our experimental results in the last section and provide a brief conclusion afterwards.

2. Related Work

Related work falls into three categories: Machine learning, Recommender systems and Query Optimization. Machine learning techniques are a good alternative to building complex cost-based models and are often easier to adapt and use with new configurations and query types. Accordingly, machine learning methods have been used to tackle a number of issues related to query optimization [3] in general, and SQL queries specifically. These issues range from statistics estimation [4], selectivity and query size estimation [5,6], execution time prediction [7], query performance prediction [8] and execution plan recycling [9,10].

Authors of [11] provide a comprehensive survey on recommender systems and classify them into three categories: 1–content-based, 2–collaborative and 3–hybrid approaches. The concept of recommendation has also been used is the field of query optimization. The work in [12] presents a number of existing methods in the literature to recommend queries to assist the user in the exploration of databases and data warehouses. While the existing approaches using recommendation for query optimization focus on query recommendation; in our work, we are interested in execution plan recommendation.

Existing approaches in the literature perform query similarity identification based on two main phases: (i) defining a pattern for an appropriate query representation; as well as (ii) developing a similarity function. Queries can be represented at the intentional level by considering the uninterpreted SQL sentence [13] or at the extensional level by using the set of tuples resulting from the query execution [14]. Other query representations range from vectors of features [10] to a set of fragments [15]. Graphs [16] are sometimes used as a pattern for query representation as well. The similarity function varies depending on the nature of the problem. [16] uses a simple equality test of query patterns while the comparison in [17] is based on separate tests of query fragments. Other ways for establishing similarity are based on classical functions applied to the queries representation. For instance, authors of [14] use the inner product, while similarity by [14,18] is identified based on cosine distance and Jaccard similarity, respectively. Our approach for similarity identification is different as we are building two features' datasets, not only one. The first dataset represents the textual features of the queries while the second dataset contains features extracted from their execution plan. Our goal is to find association rules between these two datasets and use them for plan recommendation.

3. Approach to Execution Plan Recommendation

In this paper, we propose a mechanism to build an accurate execution plan recommendation system. Figure 2 illustrates our approach.

For a given query q_n, we want the optimizer to choose a suitable execution plan, already stored in the cache memory of the DataBase Management System (DBMS), and reuse it to execute q_n. For this task, we need to study the similarity of q_n with "*old*" queries that were previously executed by the optimizer. If a similarity is detected between q_n and an "*old*" query q_o , then the execution plan $Plan_o$ used by the optimizer to execute q_o will be reused to process q_n.

Our study relays on two main components: the first is the "Features Extractor", the task of which is to capture features from query statements and execution plans. The second component is the "Similarity Evaluation" module that receives features as input and predicts similarity. The contributions of this paper can be summarized as follows:

- We firstly propose a textual representation of our data and use it to build the *Features Extractor module*.
- Secondly, we present a straightforward *Similarity Evaluation module* that relies only on the comparison of the SQL statements of the queries and studies the pitfalls associated with this method.
- Next, we show how to build an improved *Similarity Evaluation module* enabled by machine learning techniques. The improved version of the module takes into account the features of the queries' execution plans.
- We compare three machine learning algorithms and study which one is more accurate in our context.

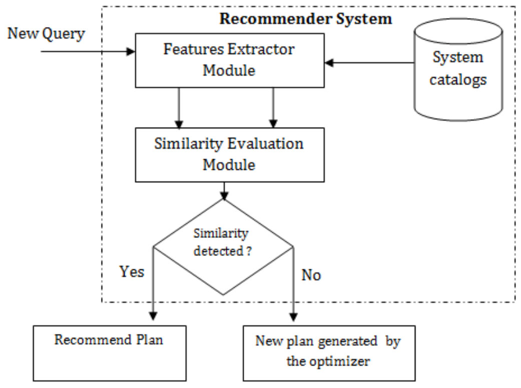
Recommender System
New Query
Features Extractor Module
System catalogs
Similarity Evaluation Module
Similarity detected ?
Yes
No
Recommend Plan
New plan generated by the optimizer

Figure 2. Our approach for plan recommendation.

4. Feature Extraction

In order to build our *Features Extractor module*, we need to identify a representation model for our data. In the straighforward solution, queries representation (Queries Features Matrix) is derived from their SQL statements, while in the improved solution, an additional representation in the execution plan space is used (Execution Plan Features Matrix). Both matrices are presented in this section.

4.1. Query Features Matrix

4.1.1. Features

As a crucial step in processing textual data, text representation converts the content of a given piece of text into a compact format so that it can be recognized and automatically classified. Consequently, the data (e.g., SQL query) is represented as a vector in the feature space, that is, a query Q would be represented as $Q = (F_1, ..., F_k)$, where k is the feature set size. In our context, a features matrix is an $n \times k$ matrix, where n is the number of query vectors.

Features, more often referred to as *terms*, can be at various levels, such as words, phrases, or any other syntactic indexing units used to identify a SQL query. For our application and approach requirements, we propose a representation method that we call the *Fragments Model*. The fragments model, proposed here, has two main advantages over the classical term model. Firstly, it considerably reduces the dimensions of the feature space and therefore reduces resource consumption in the treatment of queries. This advantage is of crucial importance when we have to deal with query datasets of a large size. The second advantage is the capacity of the fragment model to offer an entity recognition capability. That is, while the classical model treats all tokens of a given query as equivalent terms with the same nature, the fragments model offers the possibility to identify entities in the query. Therefore, relations, attributes and predicates can be recognized that are decisive in identifying queries having similar execution plans.

Fragments Model

In this representation, we split the SQL queries into fragments rather than words. Each fragment is identified by a keyword at the beginning and another one to mark its end in the SQL statement. We have identified six types of fragments as shown in Table 1. Combined fragments of all queries represent the features set.

Table 1. Fragments' models.

Fragment	Begining	End
Fragment 1	Select	From
Fragment 2	From	Where, Group By, Order By
Fragment 3	Where	like, =, Group By, Order By
Fragment 4	=	AND, End of the query
Fragment 5	AND	=, End of the query
Fragment 6	like, Group By, Order By	End of the query

Example 1. The fragments' models of queries Q_0 and Q_1 are presented in Table 2.

Q_0: SELECT LINE_LEVEL FROM PRODLEVEL WHERE LINE_LEVEL ='RLI9AYC8YAQ4'

Q_1: SELECT ACTVARS.DOLLARSALES FROM ACTVARS,PRODLEVEL
WHERE ACTVARS.PRODUCT_LEVEL= PRODLEVEL.CODE_LEVEL
AND PRODLEVEL.LINE_LEVEL ='RLI9AYC8YAQ4'

Table 2. Fragments' models of queries Q_0 and Q_1.

Fragments	Q_0 Model	Q_1 Model
Fragment 1	LINE_LEVEL	ACTVARS.DOLLARSALES
Fragment 2	PRODLEVEL	ACTVARS,PRODLEVEL
Fragment 3	LINE_LEVEL	ACTVARS.PRODUCT_LEVEL
Fragment 4	'RLI9AYC8YAQ4'	PRODLEVEL.CODE_LEVEL
Fragment 5	NULL	PRODLEVEL.LINE_LEVEL
Fragment 6	NULL	'RLI9AYC8YAQ4'

4.1.2. Weighting Method

Feature weighting [19] is another important step that assigns appropriate weights to features. The simplest weighting method is the binary representation that assigns 1 to a feature if it occurs in the data and 0 otherwise. In this solution, we prefer to use the 'Term Frequency-Inverse Document Frequency' (TF-IDF) method rather than the simple frequency weighting. The TF-IDF [20] technique provides a composite weight that is more discriminative compared to a simple term frequency weighting method, and therefore captures similarities in a more precise way. While simple frequency only provides information about presence or absence of a fragment in a given query, TF-IDF gives information on the 'importance' of that fragment in the set of queries. That is, the TF-IDF scheme assigns the highest weights to fragments that occur many times within a small number of queries and lowest weights to those occurring in the majority of queries. This property helps in obtaining quality discriminative features, which are not supported by classical term frequency representation.

TF-IDF

TF-IDF is an abbreviation for 'Term Frequency-Inverse Document Frequency'. It is a statistical metric that is widely used in information retrieval field to measure the importance of a word in a given document. This measure is calculated using Equations (1) and (2).

$$tf.idf(f,q,Q) = tf(q,f).idf(f,Q), \tag{1}$$

$$idf(f,Q) = \log \frac{N}{|q \in Q : f \in q|}, \tag{2}$$

where

- N: Total Number of Queries in the Database;
- $tf(q,f)$: Frequency of feature f in query q;
- $|q \in Q : f \in q|$: Number of queries where the feature f appears.

4.2. Execution Plan Features Matrix

In order to build the execution plan features matrix, we need to choose some features from the execution plan of the SQL queries that were previously executed by the optimizer.

The *EXPLAIN PLAN* instruction in *Oracle* [21] stores execution plans in a table called PLAN_TABLE. This table contains the number of columns, among which we identify five parameters, namely: Cost, Plan Hash Value, Cardinality, Operation and Bytes. A brief description of the features is given in Table 3, as we use them to represent the SQL query in the execution plan space.

Table 3. Description of the selected execution plan features.

Feature	Description
PLAN_HASH_VALUE	A metric that identifies execution plans.
COST	Execution cost as estimated by the optimizer. The value of this attribute has no measurement unit.
CARDINALITY	Number of rows returned by a query.
OPERATION	Defines the operation type (Select, Update, ...).
BYTES	Number of needed bytes per query.

5. Straightforward Solution

In this solution, the *Similarity Evaluation module* receives as input a $1 \times k$ features vector V_n of a new query q_n , and an $n \times k$ "old" queries features matrix M. Recall that, n is the number of old queries and k is the number of features. Then, it computes the similarity between V_n and each row of M. This step produces an $n \times 1$ similarity vector S. Each value $s_{0 \leq i \leq n}$ of S represents the similarity of query $q_{0 \leq i \leq n}$ with q_n. The plan of the query with the highest measure s is then recommended.

Several similarity measures are discussed in the literature [22]. In this paper, we use the cosine similarity, which calculates the cosine of the angle betweens two feature vectors as defined in Equation (3), where x_{1k} and x_{2k} are values of feature k in query vectors x_1 and x_2, respectively.

$$cos(\theta) = \frac{\sum_{n}^{k=1} x_{1k}x_{2k}}{\sqrt{\sum_{n}^{k=1} x_{1k}^2 \sum_{n}^{k=1} x_{2k}^2}}. \quad (3)$$

Limitation

As we show later in our experiments, this approach proved accurate to recognize queries that are almost identical, but it presents a significant risk to:

- Judge two queries as not similar, by analyzing their SQL syntax, while their execution plans are actually similar.
- Find that two queries have high similarity measure by comparing their statements, but still their plan are different. This is basically due to the fact that small differences in the SQL statements, that could be ignored in classical text categorization tasks, are very decisive in the process of defining an execution plan.

In the next session, we propose an improved solution enabled by machine learning techniques.

6. Solution Enabled by Machine Learning Techniques

6.1. Evaluating Classification Techniques

Classification is a basic task in data analysis that requires the construction of a classifier, that is, a function that is "trained" to assign a class label to unknown instances described by a set of features.

In Figure 3, we present the solution enabled by classification techniques. This approach begins with the class assigner module, the mission of which is to evaluate execution plan features of "old" queries and assign same class labels to queries that have equivalent execution plan features. This module's output is a $1 \times n$ vector C, where each value $c_{0 \leq i \leq n}$ is the class label of query $q_{0 \leq i \leq n}$. The next step is building the training set, which consists of merging vector C to the query features matrix M. Recall that matrix M contains only features that are derived from the SQL statement. Finally, a classifier will "learn" from this training set how to identify similar queries. In this paper, we evaluate three classifiers, namely, Naive Bayes, Support Vectors Machines (SVM) and Classification based on Associative Rules (CAR).

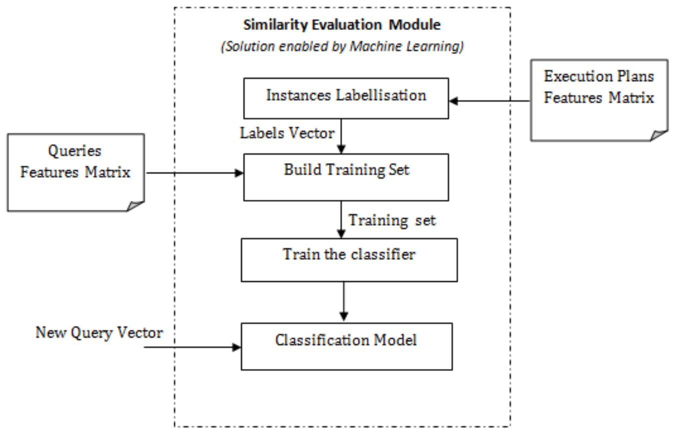

Figure 3. Improved solution enabled by classification techniques.

6.1.1. Naive Bayes

Naive Bayes classifier [23] builds a model by learning, from a training set, the conditional probability of each attribute x_i given the class label y. Next, classification is perfomed by applying Bayes rule to compute the probability of y given the particular instance of $x_1, ..., x_n$, and then predicting the class with the highest posterior probability. The computation of the predictive model is based on a strong independence assumption that considers all the attributes x_i are conditionally independent given the value of the class y. We evaluate the effect of the Naive Bayes classifier on our data, as it is one of the most effective classifiers with a high predictive performance.

6.1.2. Support Vector Machines (SVM)

Support Vector Machines [24,25] is basically a two-class classifier that estimates a decision rule $f : R^n \rightarrow \{\pm 1\}$ using training data $(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n) \in R^n \times \{\pm 1\}$ such that f will correctly classify new examples $(\vec{x}, \vec{y})$. In the case of nonlinearly separable data, SVMs use a kernel function to make the data linearly separable. We could define a kernel $K(\vec{x}, \vec{y})$ as a real-valued function $K : X \times X \rightarrow R$ for which there exists a function $\varphi : X \rightarrow F$ with the property:

$$K(\vec{x}, \vec{y}) = \langle \varphi(\vec{x}), \varphi(s\vec{y}) \rangle. \tag{4}$$

Generally, kernel functions must be positive semi-definite (PSD) to satisfy the theorem of Mercer [26]. Radial Basis Function (RBF) kernel is one of the most efficient and popular kernel functions cited in the litterature, thus we use it in our experimentations.

6.2. Evaluating Classification Based on Association Rules

Association rules which were introduced by [27] represent an automatic approach to discover relationships or possible correlations between objects. They were originally designed for finding multi-correlated items in transactions; however, they can be easily adapted for classification.

Thabtah, F. [28] gives a comprehensive review of associative classification methods. The main objective of CAR is to find a set of rules that satisfy some minimum support and minimum confidence constraints, and form an accurate classifier. Specifically, in this study, we are interested in finding a set of association rules between two SQL query representations and use them to build an efficient classifier. Recent studies have shown that using classifiers based on class association rules resulted in higher accuracy than those obtained by using other classification algorithms such as C4.5 and ILA [29], which strengthens our motivation to evaluate CAR in our improved solution.

7. Experimentations and Results

7.1. Experimental Set-Up

7.1.1. Data

Our experimentations were conducted on the Analytical Processing Benchmark (APB-1) [30], descrived in Table 4, that we used to build 2 different datasets:

- Dataset D_1: Contains 1000 simple queries having different execution plans;
- Dataset D_2: Contains 1000 queries containing subqueries.

The benchmark contains 55 query templates that we used to build D_1 while we formulated 10 templates to build D_2.

Table 4. APB1 benchmark characteristics.

Table Name	Cardinality	Size (MB)
ACTVARS	33,323,400	2085
CHANLEVEL	10	2.4×10^{-4}
CUSTLEVEL	990	2.4×10^{-2}
PRODLEVEL	9900	7.3×10^{-1}
TIMELEVEL	24	3.9×10^{-4}

7.1.2. Models' Validation

We use k-fold cross validation method for model validation. The dataset is divided into k subsets, and the holdout method is repeated k times. Alternatively, each one of the k subsets is used as a test set and the other $k - 1$ subsets are put together to form a training set. Then, the average error across all k trials is computed, and the variance of the resulting estimate is reduced as k increased. We need to consider two aspects when defining the value of parameter k. A large value of k means less bias towards overestimating the true expected error of the model but implies higher variance and higher running time. Hence, we need to choose a value that minimizes the bias without penalizing running time. Generally, the value of k is defined empirically. In our experiments, $k = 10$ offers reasonable bias/variance tradeoff.

7.1.3. Evaluation Metrics

In this paper, we use three metrics in order to measure the prediction accuracy for machine learning techniques.

1–Prediction rate: The rate of correct predictions on the test set. Prediction rate represents the percentage of instances for which predicted class value equals the actual class value.

2–RMSE: Root Mean Square Error (RMSE) is one of the most widely used statistics that measures the differences between values predicted by a model and the actual values.

3–Kappa statistic: Kappa is used as a measure to assess model quality. It is always less than or equal to 1. A value of 1 implies perfect model and values less than 1 imply less than perfect models (0.01–0.2: Slight, 0.21–0.4: Fair, 0.41–0.60: Moderate, 0.61–0.80: Substantial 0.81–0.99: Almost perfect).

7.1.4. Tools

For preprocessing the data, generating and testing the models, we use WEKA [31], a software developped by the Machine Learning Group at the University of Waikato. We performed the SVM tests with the Sequential Minimal Optimization (SMO) algorithm, which is well-known for its simplicity. We use the Java implementation (JCBA) of the Classification Based on Associations algorithm (CBA) to evaluate the performance of CAR. In the straightforward solution, Matlab was used to calculate similarity.

7.2. Experimental Results

7.2.1. Experiment 1: Straightforward Solution

In this experiment, we are interested in evaluating the performance of textual models for detecting similarity between SQL queries. We evaluate this solution at two levels:

- Firstly, we compare our fragments model to a word model (with simple frequency weighting) as we want to identify which one of these two models is more likely to identify queries having similar execution plans;
- Secondly, we test the models with our two datasets: D_1 and D_2.

For each dataset, we choose an arbitrary query to evaluate the solution. Accordingly, for D_1 we consider query Q_{new1} while we take Q_{new2} and Q_{new3} from D_2. Here, our goal is to see what are the queries that will be identified as similar to Q_{new1}, Q_{new2} and Q_{new3} by this method, using the fragment model. We consider queries for which similarity is higher than 0.95 as similar.

Statements of test queries are as follows:

Q_{new1}: SELECT ACTVARS.DOLLARSALES FROM ACTVARS,PRODLEVEL
WHERE ACTVARS.PRODUCT_LEVEL= PRODLEVEL.CODE_LEVEL
AND PRODLEVEL.LINE_LEVEL ='RLI9AYC8YAQ4'

Q_{new2}: SELECT CHANNEL_LEVEL,TIME_LEVEL,DOLLARSALES
FROM ACTVARS,CHANLEVEL WHERE ACTVARS.CHANNEL_LEVEL=
(SELECT BASE_LEVEL FROM CHANLEVEL WHERE BASE_LEVEL ='C7W3GE3KN9GU')

Q_{new3}: SELECT * FROM ACTVARS WHERE TIME_LEVEL IN (SELECT MONTH_LEVEL
FROM TIMELEVEL WHERE MONTH_LEVEL = '199607')

Table 5 presents the top three queries (*i.e.*, with the highest similarity values) associated to each test query as returned by the straightforward method. For the purpose of comparison, we additionally calculate, for each of these result queries, the similarities using word model representation. Table 6 reports the query features in the execution plan space.

Q_{Recom1}: SELECT ACTVARS.DOLLARSALES FROM ACTVARS,PRODLEVEL
WHERE ACTVARS.PRODUCT_LEVEL= PRODLEVEL.CODE_LEVEL
AND PRODLEVEL.LINE_LEVEL = 'ITQS4CE29HQR'

Q_{Recom2}: SELECT CHANNEL_LEVEL,TIME_LEVEL,DOLLARSALES
FROM ACTVARS,CHANLEVEL WHERE ACTVARS.CHANNEL_LEVEL=
(SELECT BASE_LEVEL FROM CHANLEVEL WHERE BASE_LEVEL ='SMYQ2W6NF5X8')

Q_{24}: SELECT CHANNEL_LEVEL,TIME_LEVEL,DOLLARSALES
FROM ACTVARS,CHANLEVEL WHERE CHANLEVEL.BASE_LEVEL=
(SELECT CHANNEL_LEVEL FROM ACTVARS WHERE CHANNEL_LEVEL ='SMYQ2W6NF5X8')

Q_{Recom3}: SELECT * FROM ACTVARS WHERE TIME_LEVEL IN
(SELECT MONTH_LEVEL FROM TIMELEVEL
WHERE MONTH_LEVEL = '199511')

Q_{3Sim}: SELECT * FROM ACTVARS WHERE ACTVARS.TIME_LEVEL IN
(SELECT TIMELEVEL.MONTH_LEVEL FROM TIMELEVEL
WHERE TIMELEVEL.MONTH_LEVEL = '199607')

Table 5. Cosine similarity/query model.

Dataset	New Query	Suggested Similar Query	Fragments Model	Words Model
D_1	Q_{new1}	Q_{Recom1}	0.98	0.93
		Q_{12}	0.90	0.90
		Q_{13}	0.90	0.89
D_2	Q_{new2}	Q_{Recom2}	0.97	0.92
		Q_{22}	0.91	0.90
		Q_{23}	0.87	0.92
		Q_{24}	0.33	1
D_2	Q_{new3}	Q_{Recom3}	0.98	0.93
		Q_{32}	0.85	0.95
		Q_{33}	0.78	0.92

Table 6. Comparing queries in the execution plan space.

Query	Cardinality	Bytes	Plan Hash Value
Q_{new1}	196,714	9,638,986	1,918,545,856
Q_{Recom1}	196,714	9,638,986	1,918,545,856
Q_{12}	688,500	42,687,000	1,918,545,856
Q_{13}	688,500	42,687,000	1,918,545,856
Q_{new2}	81,000	2,187,000	4,072,885,321
Q_{Recom2}	81,000	2,187,000	4,072,885,321
Q_{22}	81,000	7,695,000	4,072,885,321
Q_{23}	79,442	1,906,608	4,072,885,321
Q_{24}	65,548	1,713,152	4,072,885,321
Q_{new3}	4673	387,859	2,685,087,043
Q_{Recom3}	4490	372,670	2,685,087,043
Q_{3Sim}	4490	372,670	2,685,087,043
Q_{32}	68,850	3,167,100	2,685,087,043
Q_{33}	68,850	3,167,100	2,685,087,043

Queries identified as similar to Q_{new1}, Q_{new2} and Q_{new3}, using the fragments model, are Q_{Recom1}, Q_{Recom2} and Q_{Recom3}, respectively. By comparing features of queries in the execution plan space (Table 6), we observe that Q_{new1} and Q_{Recom1} are identical, in addition to Q_{new2} and Q_{Recom2}. On the contrary, Q_{Recom3} does not seem to be similar to Q_{new3} as its features are different in the space of execution plans. Moreover, we manually identified and recognized Q_{3Sim} as the actual similar query to Q_{new3}.

On the other hand, we can see that the fragment model is more "discriminant" compared to the word model. Actually, similarity values returned by the word model, for recommended queries, are equivalent to those of the fragments model. However, in the case of Q_{new2}, the word model would have recommended query Q_{24} (instead of query Q_{Recom2}) which is a suboptimal recommendation. In contrast, the fragments model is correctly identifying query Q_{24} as a non similar query with a similarity value that equals 0.33. This result is due to the fact that the fragments model is taking into account the position of the terms in the SQL statements, while the word model is only considering the frequency.

Besides the empirical evaluation, we randomly selected a sample of 100 queries from all datasets and counted the number of queries for which similar queries were identified correctly. Similarity was accurately identified only for the 30% of queries when the word model was applied, while we obtained a rate of 40% with the fragments model. According to the observations above, while our proposed model is more accurate compared to the simple word frequency model, it still represents a risk to recommend non similar queries.

7.2.2. Experiment 2: Solution Enabled by Machine Learning Techniques

The second experiment was conducted on D_2 dataset.

Table 7 reports the prediction rates of SVM, Naive Bayes and CAR algorithms. We observe that the highest rate is realized by the JCBA algorithm followed by Naive Bayes and SVM.

Table 7. Comparison of prediction rate.

SVM	Naive Bayes	CAR
76.55%	77.95%	91%

Figure 4 provides a comparison of the Kappa statistic and the RMSE of the classifiers. We can see that the minimal value of RMSE (0.28) is given by JCBA while Naive Bayes and SVM perform almost in the same way with values equal to 0.48 and 0.41, respectively. Furthermore, evaluation of the Kappa statistic reveals that JCBA algorithm is generating an "*Excellent*" model while Kappa values of Naive Bayes and SVM are in the same range and provide a "*Fair*" Model.

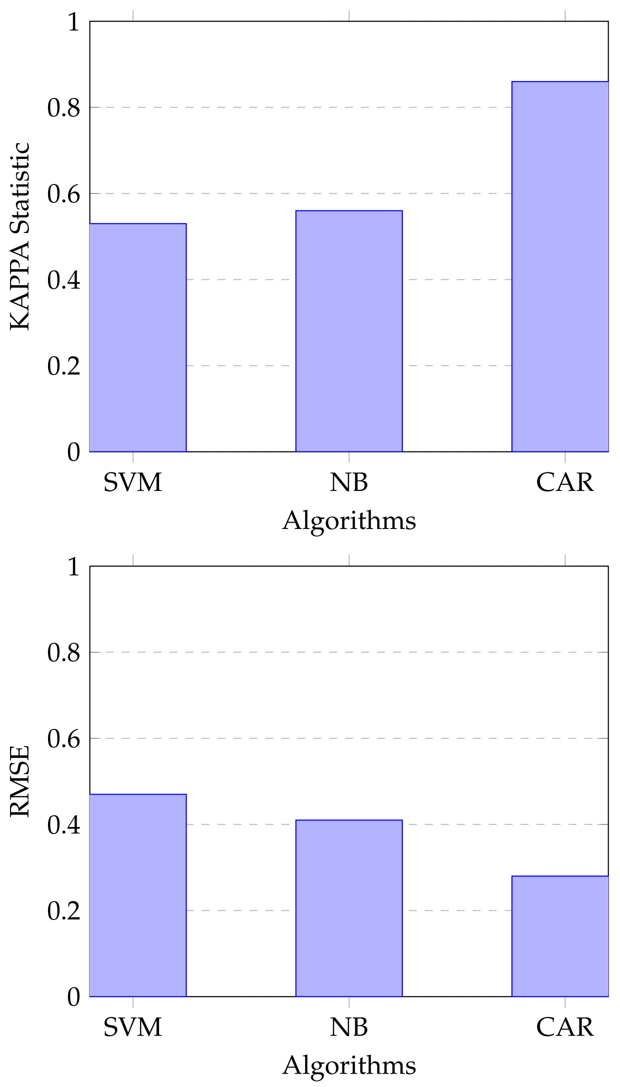

Figure 4. Comparison of Kappa and RMSE.

While the straighforward solution was not able to provide an accurate results for dataset D_2, the improved solution using machine learning techniques performed much better with a prediction rate of 76.55%. Consequently, better recommendation results are obtained, particularly when using Classification based on Association Rules.

8. Conclusions

In this paper, we presented an approach for execution plan recommendation based on similarity identification. Our goal was to build a model that recognizes similarity by using only the SQL statement of new queries. We used machine learning techniques to improve the similarity detection. Classification based on Association Rules has better potential to provide accurate prediction compared to SVM and Naive Bayes. Moreover, the proposed model was able to identify and associate similar queries having similar execution plans. Therefore, our approach is able to recommend relevant execution plans to new queries, based on similarity identification with historical queries that were previously executed by the optimizer. While execution time of models could be a constraint for using CAR, working with a parallelized version of the JCBA implementation, in future studies, can help resolve this issue. Another perspective is to use adaptive recommendations using the optimizers' feedback to enhance the prediction accuracy.

Author Contributions: Jihad Zahir and Abderrahim El Qadi conceived and designed the experiments; Jihad Zahir performed the experiments; Jihad Zahir and Abderrahim El Qadi analyzed the data and Jihad Zahir wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wu, W.; Chi, Y.; Hacígümüş, H.; Naughton, J.F. Towards predicting query execution time for concurrent and dynamic database workloads. *Proc. VLDB Endow.* **2013**, *6*, 925–936.
2. Selinger, P.G.; Astrahan, M.M.; Chamberlin, D.D.; Lorie, R.A.; Price, T.G. Access path selection in a relational database management system. In Proceedings of the 1979 ACM SIGMOD International Conference on the Management of Data, Boston, MA, USA, 30 May–1 June 1979.
3. Hasan, R.; Gandon, F. A machine learning approach to SPARQL query performance prediction. In Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, 11–14 August 2014; IEEE: Piscataway, NJ, USA, 2014; Volume 1, pp. 266–273.
4. Gao, L.; Wang, M.; Wang, X.S.; Padmanabhan, S. A learning-based approach to estimate statistics of operators in continuous queries: A case study. In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, CA, USA, 13 June 2003; ACM: New York, NY, USA, 2003; pp. 66–72.
5. Gryz, J.; Liang, D. Query selectivity estimation via data mining. In *Intelligent Information Processing and Web Mining*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 29–38.
6. Chaudhuri, S.; Ganti, V.; Gravano, L. Selectivity estimation for string predicates: Overcoming the underestimation problem. In Proceedings of the 20th International Conference on Data Engineering, Boston, MA, USA, 30 March–2 April 2004; IEEE: Piscataway, NJ, USA, 2004; pp. 227–238.
7. Ganapathi, A.; Kuno, H.; Dayal, U.; Wiener, J.L.; Fox, A.; Jordan, M.; Patterson, D. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In Proceedings of the IEEE 25th International Conference on Data Engineering, ICDE'09, Shanghai, China, 29 March–2 April 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 592–603.
8. Akdere, M.; Çetintemel, U.; Riondato, M.; Upfal, E.; Zdonik, S.B. Learning-based query performance modeling and prediction. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE), Washington, DC, USA, 1–5 April 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 390–401.
9. Sarda, P.; Haritsa, J.R. Green query optimization: Taming query optimization overheads through plan recycling. In Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, ON, Canada, 31 August–3 September 2004.
10. Ghosh, A.; Parikh, J.; Sengar, V.S.; Haritsa, J.R. Plan selection based on query clustering. In Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 20–23 August 2002.
11. Adomavicius, G.; Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 734–749.
12. Marcel, P.; Negre, E. *A Survey of Query Recommendation Techniques for Data Warehouse Exploration*; Entrepôts de Données et l'Analyse en ligne—EDA: Clermont-Ferrand, France, 2011; pp. 119–134.
13. Yao, Q.; An, A.; Huang, X. Finding and analyzing database user sessions. In *Database Systems for Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 851–862.
14. Stefanidis, K.; Drosou, M.; Pitoura, E. You May Also Like results in relational databases. In Proceedings of the International Workshop on Personalized Access, Profile Management and Context Awareness: Databases, Lyon, France, 24–28 August 2009; Volume 9.
15. Aligon, J.; Golfarelli, M.; Marcel, P.; Rizzi, S.; Turricchia, E. Mining preferences from OLAP query logs for proactive personalization. In *Advances in Databases and Information Systems*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 84–79.

16. Yang, X.; Procopiuc, C.M.; Srivastava, D. Recommending join queries via query log analysis. In Proceedings of the IEEE 25th International Conference on Data Engineering, ICDE'09, Shanghai, China, 29 March–2 April 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 964–975.
17. Gupta, A.; Mumick, I.S. *Materialized Views: Techniques, Implementations, and Applications*; MIT Press: Cambridge, MA, USA, 1999.
18. Chatzopoulou, G.; Eirinaki, M.; Koshy, S.; Mittal, S.; Polyzotis, N.; Varman, J.S.V. The QueRIE system for Personalized Query Recommendations. *IEEE Data Eng. Bull.* **2011**, *34*, 55–60.
19. Lan, M.; Tan, C.L.; Su, J.; Lu, Y. Supervised and traditional term weighting methods for automatic text categorization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2009**, *31*, 721–735.
20. Aizawa, A. An information-theoretic perspective of tf-idf measures. *Inf. Process. Manag.* **2003**, *39*, 45–65.
21. Using Explain Plan. Available online: https://docs.oracle.com/cd/B10501_01/server.920/a96533/ex_plan.htm (accessed on 10 May 2016).
22. Gomaa, W.H.; Fahmy, A.A. A survey of text similarity approaches. *Int. J. Comput. Appl.* **2013**, *68*, 13–18.
23. Friedman, N.; Geiger, D.; Goldszmidt, M. Bayesian network classifiers. *Mach. Learn.* **1997**, *29*, 131–163.
24. Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning*; Springer: Berlin/Heidelberg, Germany, 2009.
25. Wu, X.; Kumar, V.; Quinlan, J.R.; Ghosh, J.; Yang, Q.; Motoda, H.; McLachlan, G.J.; Ng, A.; Liu, B.; Philip, S.Y.; *et al.* Top 10 algorithms in data mining. *Knowl. Inf. Syst.* **2008**, *14*, 1–37.
26. Minh, H.Q.; Niyogi, P.; Yao, Y. Mercers theorem, feature maps, and smoothing. In *Learning Theory*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 154–168.
27. Agrawal, R.; Srikant, R. Fast algorithms for mining association rules. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, San Francisco, CA, USA, 12 September–15 September 1994; Volume 1215, pp. 487–499.
28. Thabtah, F. A review of associative classification mining. *Knowl. Eng. Rev.* **2007**, *22*, 37–65.
29. Nguyen, L.T.; Nguyen, N.T. Updating mined class association rules for record insertion. *Appl. Intell.* **2015**, *42*, 707–721.
30. OLAP Council APB-1 OLAP Benchmark Release II. Available online: http://www.olapcouncil.org/research/bmarkly.htm (accessed on 10 May 2016).
31. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA data mining software: An update. *ACM SIGKDD Explor. Newsl.* **2009**, *11*, 10–18.