Evaluating Semantic Queries for Dataset Engineering on the Hyperknowledge Platform

Marcio Moreno, Polyana Bezerra, Rodrigo Costa, Vítor Nascimento, Elton Soares, and Marcelo Machado

IBM Research, Brazil, Av Pasteur 146 Rio de Janeiro - RJ, Brazil
`mmoreno@br.ibm.com, {polyana.bezerra, rodrigo.costa, vitor.nascimento, eltons, marcelo.machado}@ibm.com`

Abstract. Machine learning typically requires training and validation of models with large and heterogeneous datasets. The engineering of these datasets is a critical task for enabling high accuracy and generalization, although in many cases it is done following an ad-hoc approach. Hyperknowledge can enable more structured engineering of datasets, by representing the datasets' symbolic and non-symbolic information, within the same framework, and enabling queries for dataset creation, retrieval, resampling, and combination. In this poster paper, we present how the Hyperknowledge Platform evaluates those queries and analyze its performance quantitatively. The preliminary results indicate that our platform can support data scientists' work while adding negligible time overhead.

Keywords: Hyperknowledge; Hybrid Knowledge Representation; Hyperlinked Knowledge Graph; HyQL; Multimodal data.

1 Introduction

As the popularity of Machine Learning (ML) tasks increases, so does the amount of data used to train and test them. The effectiveness of such algorithms is related to the quality and variety of the data applied during the training stage [3]. The continuous growth in size and heterogeneity of datasets used in ML tasks makes what we call *Dataset Engineering* (DE) a key step for effective data exploitation, leading to more effective models. Here we define DE as the process of handling data through structuring and traceability. In this paper, we present how the Hyperknowledge Platform [5] enables the engineering of datasets, by leveraging the Hyperknowledge (HKW) model and the Hyperknowledge Query Language (HyQL). We evaluate the response times of the HyQL queries related to those tasks to check if these are lower, or in the same order of magnitude, as the requirements posed by other data science tasks (such as model training, raw file transfer time, and size). Thus, we designed a testbed containing a dataset that combines a domain ontology and a dataset of images along with its metadata and bounding boxes descriptions; and a set of queries that represents DE-related tasks. We argue our approach can assist data scientists in their tasks saving time and promoting better data exploitation.

2 Hyperknowledge Platform

The central component of the HKW Platform, depicted in Figure 1, is the HKBase that stores and maintains instances of HKW models. It has a flexible internal architecture to which components can be coupled through the implementation of one of its APIs (IObserver, IDataBase, and IReasoner). The functionalities provided by HKBase are exposed to applications through its RESTful API. The current HKBase implementation uses two different data sources, one for storing data structured following the HKW model and another for storing unstructured content, which can be used for storing datasets. It also contains both the query and reasoning engines that will be used for processing HyQL queries. An engine implementation might be optimized depending on which data source is being used for storing the model, without creating a hard dependency for HKBase.

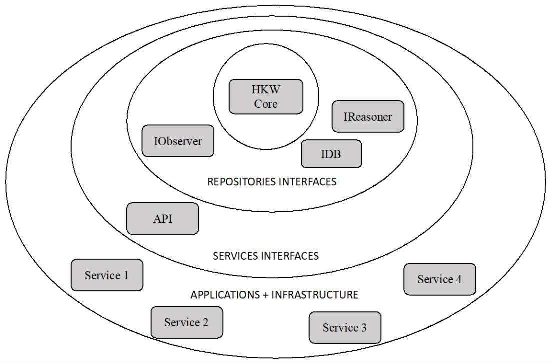

Fig. 1. HKW Platform architecture [5].

The HyQL engine's main components are its grammar [2] and its processing engine. The processing steps followed by the HyQL processing engine are summarized in Algorithms 1 and 2. In both algorithms, the variable $insertClauses$ will be used to store the set of pairs (c, E) such that c is the target context and E is the set of entities to be inserted in c. Meanwhile, the variable $updateClauses$ will store the set of tuples (c, E, C, P) such that c is the target context, E is the set of entities to be inserted in c, C is the set of contexts to be generated by the SPLIT function, and P is the set of proportion values associated with each context in C. Finally, in Algorithm 1, the variable R stores the set of entities/rows to be retrieved by the query.

The $execDEClauses$ function logic is described in Algorithm 2. This function receives the $insertClauses$ and $updateClauses$ variables, passed as parameters to the call performed in line 13 of Algorithm 1, to identify which contexts and reference nodes [3] must be created in HKBase as a result of the query.

[2] Accessible at https://ibm.box.com/v/iswc2021-hyql-grammar. For clarity, we have suppressed the handling of spaces, comments, and lower case keywords.

[3] Reference node is a special type of HKW node that promotes the reuse of nodes across multiple contexts.

Algorithm 1: HyQL engine main processing steps.

```
    Input: HyQL Query $Q$
 1  $insertClauses = \emptyset$; $updateClauses = \emptyset$; $R = \emptyset$;
 2  $subQueriesObjects = ParseQuery(Q)$;
 3  foreach subquery object $sQ$ of $subQueriesObjects$ do
 4      $sQData = FetchData(sQ)$;
 5      $sQResultSet = EvaluateConstraints(sQ, sQData)$;
 6      $sQR = getResults(sQResultSet, sQ)$;
 7      $sQI = (sQc, sQE) = getInsertClause(sQ, sQR)$;
 8      if $sQI \neq \emptyset$ then add $sQI$ to $insertClauses$;
 9      $sQU = (sQc, sQC, sQP, sQE) = getUpdateClause(sQ, sQR)$;
10      if $sQU \neq \emptyset$ then add $sQU$ to $updateClauses$;
11      merge $sQR$ with $R$;
12  end
13  $execDEClauses(insertClauses, updateClauses)$;
14  return $R$
```

Algorithm 2: Dataset engineering clauses execution.

```
    Input: $insertClauses$ and $updateClauses$
 1  foreach pair $(c, E)$ of $insertClauses$ do
 2      create context $c$ in HKBase;
 3      foreach entity $e$ of $E$ do create a reference node to $e$ inside $c$;
 4  end
 5  foreach tuple $(c, C, P, E)$ of $updateClauses$ do
 6      if $(C \neq \emptyset)$ and $(P \neq \emptyset)$ then
 7          foreach context $sc$ of $C$ and corresponding proportion $sp$ of $P$ do
 8              create subcontext $sc$ inside $c$;
 9              create reference nodes to $sp$ of $E$ within $sc$;
10          end
11      else
12          foreach entity $e$ of $E$ do create a reference node to $e$ inside $c$;
13  end
```

3 Experimental Evaluation

To evaluate our work quantitatively, we have measured the median execution time the HyQL engine spends executing some of the queries related to DE tasks. The main goal of this evaluation is to answer the following research questions (RQs): (RQ1) Is the time required to answer DE queries within our platform lower or in the same order of magnitude as the time consumed by other data science tasks, such as model training and raw file transfer time? (RQ2) Are there meaningful differences in response times across HKBase configurations?

To answer those RQs, we designed an experimental dataset based on the Pascal VOC2012 [4]. Using the HKW Platform, this dataset was represented as an HKW context containing descriptions of its classes, images, and relationships between the bounding boxes of images and classes using 20 concept nodes, 17.125 content nodes, and 57.263 links between node anchors, respectively.

Based on this dataset we designed 11 experimental queries, presented in Table 1, that include 4 SELECT queries, 4 INSERT queries based on the results of SELECT queries, an UPDATE query that splits data into multiple contexts, a SELECT query that uses spatial operators for filtering results, and an INSERT query based on this SELECT query.

In total, 3 HKBase configurations, presented in Table 2, were evaluated. Evaluating multiple combinations of HKBase data sources, and HyQL engine implementations, helps us understand which configurations perform better for each type of query and allows us to find the optimal setting for a given scenario.

The Generic HyQL engine implementation can work with any data source supported by HKBase as all query processing is performed using the standard data source API (IDB) while the SPARQL-based HyQL engine optimizes the processing of HyQL clauses in Jena by translating them directly into a single SPARQL query with multiple subqueries.

The hardware in which these experiments were executed was composed of two dedicated Dell OptiPlex 7050 machines [4]. Half the repetitions for every configuration were executed on each machine to guarantee a fair comparison. The software setup of both machines was also the same [5]. The scripts used to run the experiments were implemented in *Python* and used *Docker Compose* to deploy and reset the service containers. Both the server and the client script ran on the same host machine. For running the queries, the dataset was loaded once and all the queries were executed in sequence, following their numerical order. All repetitions of each query were also executed in sequence and after each repetition of queries with INSERT and UPDATE clauses, the execution script removed all the entities generated by the previous repetition to preserve the IID assumption [2].

Table 1. HyQL queries designed for experiments.

ID	HyQL Query
Q1	SELECT dataset
Q2	SELECT training_set
Q3	SELECT test_set
Q4	SELECT distinct image WHERE image has cat
Q5	INSERT INTO myDatasets VALUES (SELECT dataset)
Q6	INSERT INTO myTrainingSets VALUES (SELECT training_set)
Q7	INSERT INTO myTestSets VALUES (SELECT test_set)
Q8	INSERT INTO myCatImages VALUES (SELECT distinct image WHERE image has cat)
Q9	UPDATE datasets APPLY (SPLIT(image, [”image_training_set”, ”image_validation_set”, ”image_test_set”], $[0.65, 0.05, 0.3]$) GROUP BY image)
Q10	SELECT image WHERE image has person AND image has cat AND person contains cat
Q11	INSERT INTO myCatAndPersonImages VALUES (SELECT image WHERE image has person AND image has cat AND person contains cat)

Table 2. HKBase configurations evaluated.

Configuration	Description
HKB + MongoDB + Generic HyQL	HKBase with MongoDB and Generic HyQL engine.
HKB + Jena + Generic HyQL	HKBase with Jena and Generic HyQL engine.
HKB + Jena + HyQLtoSPARQL	HKBase with Jena and SPARQL-based HyQL engine.

A total of 100 query repetitions were executed on each machine, totaling 200 query repetitions. As highlighted by the 95% confidence intervals (vertical black lines on the upper edge of each bar) in the bar plots of Figure 2, these were sufficient to allow distinguishing which configurations were performed better for each query. These preliminary results indicate that the time overhead required for using our system is minimal when compared to time spent in typical data science tasks, such as: (1) fitting a model to the data (training), which can take multiple days or even months; (2) managing datasets manually, which currently consumes a high proportion of data scientists' work hours [1].

Therefore, we conclude that the answer to the RQ1 is that the requirements posed by our solution can be up to several orders of magnitude lower than the requirements posed by other data science tasks, introducing an overhead of at maximum a couple of minutes required for the retrieval, creation and/or integration of datasets using our system, while removing the need of performing several tasks manually. Also, with regards to RQ2, the results indicate that HKB + Jena + HyQLtoSPARQL configuration presents lower response times.

[4] CPU i7-7700T 2.90 GHz, DDR4 2400 MHz 8GiB, and SSD LITEON 119GiB.

[5] Fedora OS 31, Docker 19, Conda, MongoDB 4.0, and Jena Fuseki 3.12.0 TDB2.

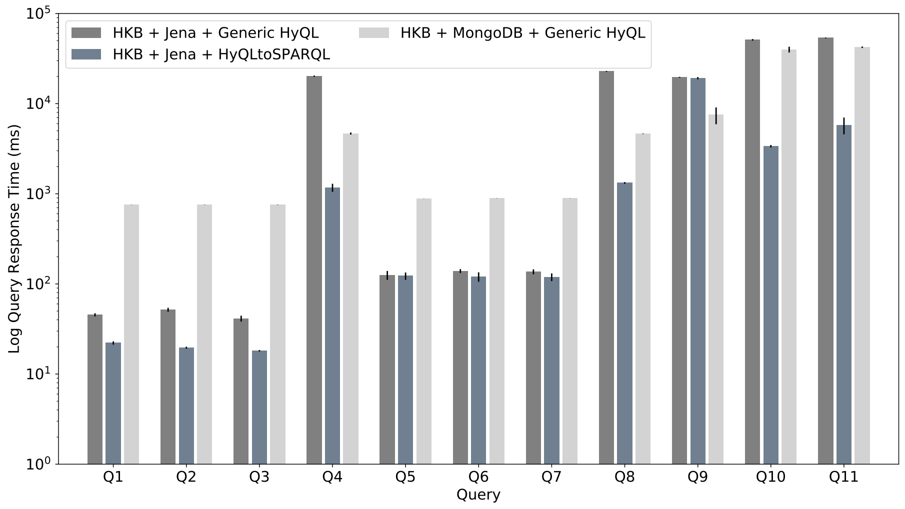

Fig. 2. Median query response time for each HKBase configuration evaluated.

4 Conclusion

In this paper, we presented how the HKW Platform enables semantic queries for dataset engineering. The main contributions of this work are the proposal, implementation, and evaluation of the HyQL engine extensions required for supporting these queries. Data scientists could use this solution to find new datasets, balance, clean, and resample them, and select specific features, which saves time and promotes better data exploitation. The quantitative analysis of the HyQL engine has indicated that the queries discussed in this work can be evaluated in a reasonable amount of time. When we compare the results obtained in our preliminary analysis with the time currently spent for DE and ML tasks, we conclude that the overhead introduced by our approach is negligible.

References

1. Anaconda: 2020 state of data science. Tech. rep., Anaconda, Inc. (2020)
2. Donatiello, L., Nelson, R.: Performance evaluation of computer and communication systems: joint tutorial papers of Performance'93 and Sigmetrics' 93, vol. 729. Springer Science & Business Media (1993)
3. Dong, X.L., Rekatsinas, T.: Data integration and machine learning: A natural synergy. In: Proceedings of the 2018 international conference on management of data. pp. 1645–1650 (2018)
4. Everingham, M., Eslami, S.A., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes challenge: A retrospective. International journal of computer vision 111(1), 98–136 (2015)
5. Moreno, M.F., Santos, R.C., dos Santos, W.H., Cerqueira, R.: Kes: The knowledge explorer system. In: International Semantic Web Conference (P&D/Industry/BlueSky) (2018)