Schema Profiling of Document Stores*

Enrico Gallinucci, Matteo Golfarelli, and Stefano Rizzi

DISI — CINI, University of Bologna, Italy

Abstract. In document stores, schema is a soft concept and the documents in a collection can have different schemata; this gives designers and implementers augmented flexibility but requires an extra effort to understand the rules that drove the use of alternative schemata when heterogeneous documents are to be analyzed or integrated. In this paper we outline a technique, called schema profiling, to explain the schema variants within a collection in document stores by capturing the hidden rules explaining the use of these variants; we express these rules in the form of a decision tree, called schema profile, whose main feature is the coexistence of value-based and schema-based conditions. Consistently with the requirements we elicited from real users, we aim at creating explicative, precise, and concise schema profiles; to quantitatively assess these qualities we introduce a novel measure of entropy.

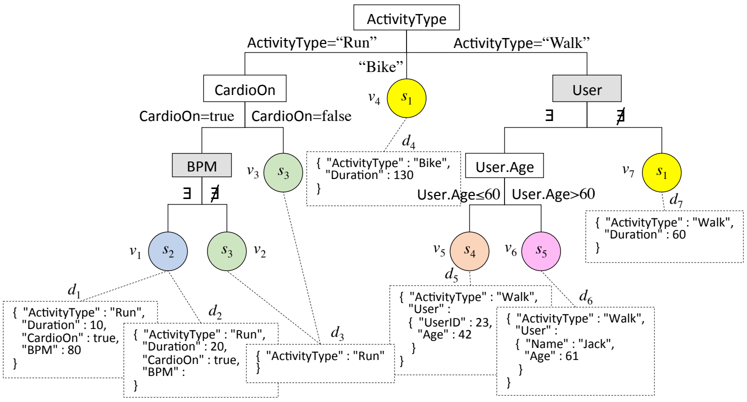
Keywords: NoSQL, Schema Discovery, Decision Trees

1 Motivation and Outline

Recent years have witnessed an erosion of the relational DBMS predominance to the benefit of DBMSs based on alternative representation models (e.g., document-oriented and graph-based) which adopt a *schemaless* representation for data. Schemaless databases are preferred to relational ones for storing heterogeneous data with variable schemata and structural forms; typical schema variants within a collection consist in missing or additional attributes, in different names or types for an attribute, and in different structures for instances. The absence of a unique schema grants flexibility to operational applications but adds complexity to analytical applications, in which a single analysis often involves large sets of data with different schemata. Dealing with this complexity requires a notable effort to understand the rules that drove the use of alternative schemata, plus an integration activity to identify a common schema to be adopted for analysis —which is quite hard when no documentation is available.

In this paper we outline a technique to explain the schema variants within a collection in document stores by capturing the hidden rules explaining the use of these variants. We call this activity *schema profiling*. Schema profiling can be used for instance when trying to decode the behavior of an undocumented application that manages a document-base, or to support analytical applications

* This work was partly supported by the EU-funded project TOREADOR (contract n. H2020-688797).



```
{ "ActivityType" : "Run",
  "Duration" : 10,
  "CardioOn" : true,
  "BPM" : 80
}
```

```
{ "ActivityType" : "Run",
  "Duration" : 20,
  "CardioOn" : true,
  "BPM" :
}
```

```
{ "ActivityType" : "Run"
}
```

```
{ "ActivityType" : "Bike",
  "Duration" : 130
}
```

```
{ "ActivityType" : "Walk",
  "User" :
  { "UserID" : 23,
    "Age" : 42
  }
}
```

```
{ "ActivityType" : "Walk",
  "User" :
  { "Name" : "Jack",
    "Age" : 61
  }
}
```

```
{ "ActivityType" : "Walk",
  "Duration" : 60
}
```

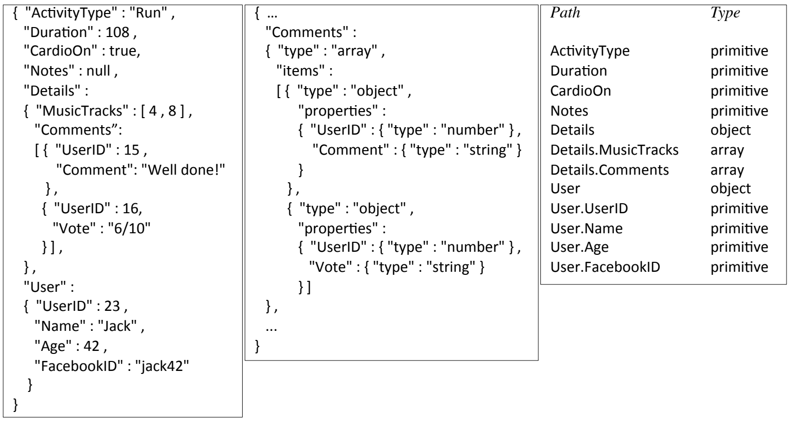
Fig. 1. A schema profile in the physical fitness domain

that query a document store following either a schema-on-write or schema-on-read approach [2].

A schema profile should describe the rules for assigning a document to a schema based on the document features; hence, modeling a schema profile as a *classifier* is a natural choice, which also allows the existing literature on classification to be reused. More specifically, since users need a comprehensible and compact representation of schema profiles, among the different types of classifiers we focus on decision trees.

Straightly reusing traditional decision trees for schema profiling would mean classifying documents based on the *values* of their attributes only. However, this would often lead to trees where a single rule explains different schemata, which would yield an imprecise information. To address this issues, in our approach documents are also classified using *schema-based conditions* related to the presence or absence of attributes. Consider for example Figure 1, showing a portion of a schema profile built in the domain of physical fitness to describe a collection of documents generated by training machines. Each internal node in the tree is associated with a document attribute a and can express either a value-based condition (white box; each outgoing arc is related to one or more values of a, e.g., $\mathsf{User.Age} < 60$) or a schema-based condition (grey box; the two outgoing arcs represent the presence or absence of a in the document, e.g., $\exists\mathsf{BPM}$). Each path in the tree models a *rule*; it leads to a leaf (represented as a circle) that corresponds to a schema found for the documents that meet all the conditions expressed along that path (document examples are shown in dashed boxes). So, for instance, schema s_1 is used in all the documents for which either $\mathsf{ActivityType} =$ "Bike", or $\mathsf{ActivityType} =$ "Walk" and field User is not present.

Another drawback of traditional decision trees is that they often give several rules for the same class. While this may be correct for some specific collections (e.g., schema s_1 in Figure 1 appears in two leaves, i.e., it is explained by two different rules), in general we wish to keep the number of rules to a minimum



```
{ "ActivityType" : "Run" ,
  "Duration" : 108 ,
  "CardioOn" : true,
  "Notes" : null ,
  "Details" :
  { "MusicTracks" : [ 4 , 8 ] ,
    "Comments”:
    [ { "UserID" : 15 ,
          "Comment": "Well done!"
       } ,
      { "UserID" : 16,
         "Vote" : "6/10"
       } ] ,
  } ,
  "User" :
  { "UserID" : 23 ,
    "Name" : "Jack" ,
    "Age" : 42 ,
    "FacebookID" : "jack42"
  }
}
```

(a)

```
{ ...
  "Comments" :
  { "type" : "array" ,
    "items" :
    [ { "type" : "object" ,
        "properties" :
        { "UserID" : { "type" : "number" } ,
          "Comment" : { "type" : "string" }
        }
      } ,
      { "type" : "object" ,
        "properties" :
        { "UserID" : { "type" : "number" } ,
          "Vote" : { "type" : "string" }
        } ]
  } ,
  ...
}
```

(b)

Path	*Type*
ActivityType	primitive
Duration	primitive
CardioOn	primitive
Notes	primitive
Details	object
Details.MusicTracks	array
Details.Comments	array
User	object
User.UserID	primitive
User.Name	primitive
User.Age	primitive
User.FacebookID	primitive

(c)

Fig. 2. A JSON document representing a training session (a), a portion of its JSON schema (b), and its r-schema (c)

aimed at giving users a more concise picture of schema usage. This is achieved in our approach by introducing a novel measure called *schema entropy* that, together with the classical entropy measure, enables a better assessment of the quality of a schema profile.

2 From Documents to Schema Profiles

The central concept of a document store is the notion of a *document*. Following the most widely adopted format, a document is a JSON object. An *object* is formed by a set of *name/value* pairs called *elements*. A *value* can be either a primitive value (i.e, a number, a string, or a Boolean), an array of values, an object, or null. A collection D is a set of documents. Arrays have no constraint on their size and on the type of their values, i.e., an array can simultaneously contain numbers, strings, other arrays, as well as objects with different structures. Figure 2.a shows a document that represents a training session containing an array of objects with different schemata (element Comments).

The JSON schema initiative provides the specifications to define the schema of a document; however, as shown in Figure 2.b, the resulting schemata provide a complex representation of arrays and are quite verbose. Indeed, the schema of an array is defined as the ordered list of the schemata of its values, so two arrays share the same schema only if they contain the same number of values and these values share the same schemata in the same order. This schema-matching criterion would add unnecessary burden to our approach —also considering that the type variability of array elements is less relevant than that of the other attributes. So we adopt a more concise representation for the schema of a document, called *reduced schema*, which does not enter into the content of arrays, but simply denotes the presence of an array structure.

Definition 1 (R-Schema of a Document). *Given document d, the reduced schema (briefly,* r-schema*) of d, denoted $rs(d)$, is a set of* attributes*, each corresponding to one element in d. Attribute $a \in rs(d)$ is identified by a* pathname*, $path(a)$, and by a type, $type(a) \in \{\mathsf{primitive}, \mathsf{object}, \mathsf{array}\}$. While $path(a)$ is a string in dot notation reproducing the path of the element corresponding to a in d, $type(a)$ is the type of that element (type* primitive *generalizes numbers, strings, Booleans, and nulls).*

Note that, although r-schemata are defined as sets of attributes, their pathnames code the document structure (short of the internal structure of arrays).

Example 1. Figure 2 shows a sample document, its JSON schema (as per the specifications of the JSON schema initiative), and its r-schema. Note how, in the r-schema, the complexity and heterogeneity of array Comments is hidden in attribute Details.Comments with generic type array.

To put together the relevant information coded by different r-schemata, for the documents within a collection D we define a global r-schema that includes all the distinct attributes that appear in the r-schemata of the documents in D:

Definition 2 (R-Schema of a Collection). *Given collection D, we denote with $S(D)$ the set of distinct r-schemata of the documents in D (where two attributes in the r-schemata of two documents are considered equal if they have the same pathname and the same type). The* r-schema *of D is defined as $rs(D) = \bigcup_{d \in D} rs(d)$. Given $s \in S(D)$, we denote with $|D|_s$ the number of documents in D with r-schema s.*

Based on the concepts introduced above, we can now define schema profiles.

Definition 3 (Schema Profile). *A* schema profile *for collection D is a directed tree T where each internal node corresponds to some attribute $a \in rs(D)$ and expresses a condition that can be either* value-based *or* schema-based*: (i) a node expressing a schema-based condition has exactly two outgoing arcs, labelled as $\exists$ and $\nexists$ respectively; (ii) a node expressing a value-based condition has two or more outgoing arcs, each labelled with a condition over the domain of a. Value-based conditions can only be expressed on attributes of type* primitive*. Given node v, we denote with $D_v \subseteq D$ the set of documents that meet all the conditions expressed by the nodes in the path from the root to v, with $S(D_v) \subseteq S(D)$ the set of distinct r-schemata of the documents in D_v, and with $|D_v|_s$ the number of documents with r-schema $s \in S(D_v)$ belonging to D_v.*

Intuitively, each path in a schema profile models a *rule* that includes a set of conditions for selecting one or more r-schemata. We also remark that Definition 3 can accommodate both binary and n-ary trees.

Example 2. Figure 1 shows an n-ary schema profile with two schema-based conditions (User and BPM) and three value-based conditions (ActivityType, CardioOn, and User.Age). In this case, it is $D = \{d_1, \ldots, d_7\}$ and $S(D) = \{s_1, \ldots, s_5\}$. The schema profile has leaves $v_1, \ldots, v_7$, with $D_{v_1} = \{d_1, d_2\}$. Note that document d_3 belongs to both v_2 and v_3, since attribute CardioOn is missing.

3 Evaluating Schema Profiles

The first stage of our work has been devoted to elicit user requirements for schema profiling with reference in two application domains: that of a company selling fitness equipment, whose documents contain the registration of workout sessions, and that of a software development company, whose documents contain the log of the errors generated by the deployed applications. The requirements we elicited can be summarized as follows: (i) a schema profile should be *explicative*, i.e., it should give priority to value-based conditions (which explain the difference between two r-schemata in terms of the values taken by an attribute) over schema-based ones (which merely acknowledge this difference); (ii) a schema profile should be *precise*, i.e., it should accurately characterize each single r-schema by avoiding mixing documents with different r-schemata in the same leaf of the tree; (iii) a schema profile should be *concise*, i.e., it should provide a small set of rules (a single rule for each r-schema).

Now we can informally state our final goal as that of finding a schema profile that achieves a trade-off between precision, conciseness, and explicativeness. The problem can be modeled as a classification one where the documents, properly extended with schema information, are the objects to be classified according to the schema labels. Since schema profiles are decision trees, to solve this problem we have extended the well-known C4.5 algorithm [6]. For space reasons we cannot give the details of the extension here, so in the following we just discuss how the above-mentioned requirements can be quantitatively characterized.

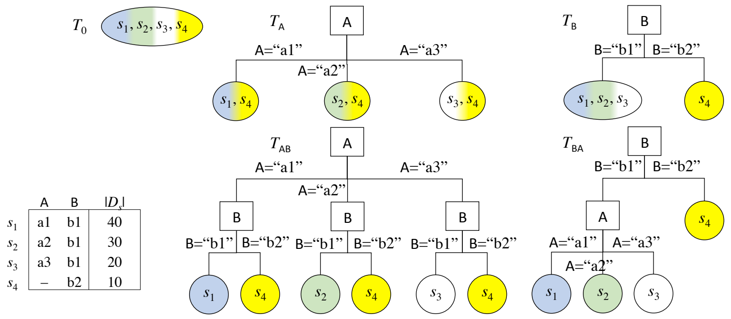
3.1 Explicativeness

We consider a schema profile to be explicative if it prefers value-based conditions over schema-based ones, because the latter acknowledge that there is a difference between two r-schemata but do not really explain its reason. Indeed, a value-based condition always relates two different attributes (for instance, with reference to Figure 1, attributes User.Age and UserID); this is because a condition on the values of an attribute a does not partition the documents based on the presence/absence of a. Conversely, a schema-based condition on a always partitions the documents based on the presence/absence of a so, in a sense, it merely explains itself.

So, to evaluate explicativeness we use the number of schema-based conditions in the schema profile T: the lower this number, the more explicative T.

3.2 Precision

A distinguishing feature of the classification approaches based on decision trees is the function they adopt to quantify the "purity" of the leaves where observations are classified, where a leaf is said to be *pure* if all its observations share the same class. The most common function used to this end is *entropy* [8].



	A	B	$\lvert D_s \rvert$
s_1	a1	b1	40
s_2	a2	b1	30
s_3	a3	b1	20
s_4	–	b2	10

Fig. 3. A collection (on the left) and five possible schema profiles (see Example 3)

Definition 4 (Entropy). *Let $S(D)$ be the set of distinct r-schemata of the documents in collection D, and T be a schema profile for D with leaves $v_1, \ldots, v_m$. The entropy of T is*

$$entropy(T) = \sum_{j=1}^{m} \frac{|D_{v_j}|}{|D|} \cdot entropy(v_j)$$

where $\frac{|D_{v_j}|}{|D|}$ is the probability of leaf v_j, $entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|}$ is the entropy *of leaf v_j, and $\frac{|D_{v_j}|_s}{|D_{v_j}|}$ is the probability of r-schema s within leaf v_j. Leaf v_j is said to be* pure *if $entropy(v_j) = 0$.*

Entropy is strictly related to precision: within a schema profile with null entropy, all the documents included in each leaf share the same r-schema, thus the schema profile has maximum precision.

Example 3. Let D be a collection with 100 documents, 4 r-schemata $s_1, \ldots, s_4$, and two attributes, A and B. The values of the attributes for the different r-schemata are listed in Figure 3 (symbol “–” means “any value”), together with five possible schema profiles. A degenerate schema profile T_0 made of a single root node v_0 has entropy $entropy(T_0) = entropy(v_0) = 1.85$ (because $D_{v_0} \equiv D$). Schema profiles T_{A} and T_{B} have $entropy(T_{\mathsf{A}}) = 0.46$ and $entropy(T_{\mathsf{B}}) = 1.38$, respectively. Both schema profiles T_{AB} and T_{BA} have null entropy because no leaf includes multiple r-schemata.

3.3 Conciseness

Entropy is focused on node purity, hence its minimization often leads to split observations of the same class among several leaves. While this is a secondary problem in generic classification problems, where the precision of the resulting model is more important than its readability, it becomes critical in schema profiling since it conflicts with the conciseness requirement. Indeed, in our context, each r-schema might end up for being explained by a wide set of rules, thus precluding users from getting a concise picture of schema usage. For instance, with reference to Example 3, though both schema profiles T_{AB} and T_{BA} have null entropy, the latter is clearly the one that best represents the collection, with each r-schema being reached by a single path in the tree.

To evaluate the conciseness of schema profiles, we propose a measure called *schema entropy*. Our requirement is to reduce the number of rules provided by maximizing the cohesion of the documents that share the same r-schema —a maximally concise schema profile is one where there is a single rule for each r-schema. So we invert the original definition of entropy to relate it to the purity of the r-schemata instead of the purity of the leaves: in terms of entropy, a leaf is pure if it contains only documents with the same r-schema; in terms of schema entropy, an r-schema is pure if all its documents are in the same leaf. The schema entropy of a degenerate schema profile where all the documents are included into a single node (the root) is 0; when documents are split into different leaves, the schema entropy can never decrease.

Definition 5 (Schema Entropy). *The schema entropy of T is*

$$sEntropy(T) = -\sum_{s \in S(D)} \frac{|D|_s}{|D|} \cdot sEntropy(s)$$

where $sEntropy(s) = \sum_{j=1}^{m} \frac{|D_{v_j}|_s}{|D|_s} \log \frac{|D_{v_j}|_s}{|D|_s}$ *is the* schema entropy *of r-schema* $s \in S(D)$*.*

Example 4. With reference to Example 3, it is $sEntropy(T_0) = sEntropy(T_{\mathsf{B}}) = sEntropy(T_{\mathsf{BA}}) = 0$ and $sEntropy(T_{\mathsf{A}}) = sEntropy(T_{\mathsf{AB}}) = 0.16$. Therefore, if both entropy and schema entropy are considered, T_{BA} is the best schema profile.

4 Related Work and Conclusions

Early work on schema discovery focused on describing semi-structured data retrieved from web pages based on the Object Exchange Model (e.g., [5]). As XML became a standard for data exchange on the web, work such as [1] emerged to help software tools in processing XML documents or in integrating data. With the replacement of XML with JSON, similar issues are now being addressed aimed at capturing and representing the intrinsic variety of schemata within a collection of JSON objects. In [3] a unique schema is built for the JSON objects returned by a single API service, while the union of all the attributes in a collection of JSON objects is modeled in [4] to measure the heterogeneity of a collection. In [7], a versioned schema model for a collection of JSON documents is built by creating a new version of the same attribute for every intensional variation found in the documents. Finally, in [9] the documents of a collection are clustered to identify groups of similar schemata; then, the schemata of each group are summarized into a *skeleton*, i.e., a tree containing the smallest set of core attributes according to a frequency-based formula.

In this paper we have outlined an approach to schema profiling for document stores. The idea is to capture the rules that explain the use of different schemata within a collection through a decision tree whose nodes express either value-based or schema-based conditions. Though our approach has some points in common with the above mentioned ones, its goal is completely different. The ultimate goal of the previous work is to provide a way to *describe* the intensional aspects of the documents, while ours is to *explain* the schema variants. Besides, no other approach considers the extensional point of view to describe schema variants. Finally, while all previous approaches produce in output some form of skeleton schema, we create a schema profile that classifies schema variants.

We made some experimental tests using both synthetic and real-world collections, with numbers of documents ranging from about 10000 to more than 5 millions, and it turned out that our approach achieves a good trade-off among precision, conciseness, and expressiveness. Specifically, to assess its accuracy we compared the schema profiles it delivers with a baseline (for synthetic collections the baseline is the set of rules used to generate the collection itself, while for real-world collections it has been provided by the users). For comparisons we adopt the weighted *tree edit-distance* (i.e., the length of the cheapest sequence of node-edit operations that transforms one tree into the other, weighted on the tree depth), which ranges between 0 and 1.17 for synthetic collections and between 0 and 1.33 for real-world collections. Finally, in terms of efficiency, we found that the total time to build a schema profile ranges from a few seconds to about 3 minutes, depending on the number of documents and on their number of attributes (on a 64-bits Intel Core i7 quad-core 3.4GHz).

References

1. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning deterministic regular expressions for the inference of schemas from XML data. ACM TWEB 4(4), 14 (2010)
2. Dong, X.L., Srivastava, D.: Big data integration. In: Proc. ICDE. pp. 1245–1248 (2013)
3. Izquierdo, J.L.C., Cabot, J.: Discovering implicit schemas in JSON data. In: Proc. ICWE. pp. 68–83 (2013)
4. Klettke, M., Störl, U., Scherzinger, S., Regensburg, O.: Schema extraction and structural outlier detection for JSON-based NoSQL data stores. In: Proc. BTW. vol. 2105, pp. 425–444 (2015)
5. Nestorov, S., Ullman, J., Wiener, J., Chawathe, S.: Representative objects: Concise representations of semistructured, hierarchical data. In: Proc. ICDE. pp. 79–90 (1997)
6. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
7. Ruiz, D.S., Morales, S.F., Molina, J.G.: Inferring versioned schemas from NoSQL databases and its applications. In: Proc. ER. pp. 467–480 (2015)
8. Shannon, C.E.: A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review 5(1), 3–55 (2001)
9. Wang, L., Zhang, S., Shi, J., Jiao, L., Hassanzadeh, O., Zou, J., Wangz, C.: Schema management for document stores. Proc. VLDB Endowment 8(9), 922–933 (2015)