Schema Extraction Model

Ni Lao, 2007.11.25

1.Motivation

As we can see from the Cafarella and et al. (2007) paper, there is a trend of IE to become unsupervised, domain-independent, and scalable. This is mainly achieved by using lightweight NLP algorithm (like POS tagger and chunker for TextRunner (Banko et al. 2007)) on the Web data in an unsupervised fashion.

In order to utilize the extracted information, three models are proposed by Cafarella and et al. These structural access models differ at how much work is done offline. Generally the structured assess to the web text has two stages: the *extraction stage* that produces facts from webpages, and the *integration stage* that produces table from facts. The *Schema Extraction Model* does both stages offline, the *Text Query Model* does both stages offline, and the *Extraction DB Model* falls in the middle.

	Extraction	Integration
Schema Extraction Model	offline	offline
ExDB Model	offline	online
Text Query Model	online	online

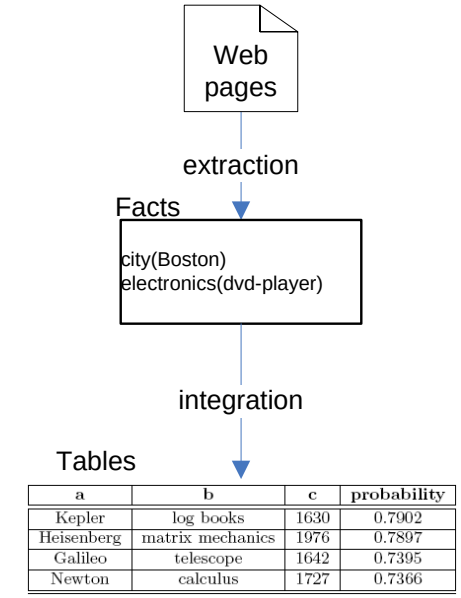

Figure: Two stages of structured web access

While ExDB model is useful for expert information seekers, or as a backend to other applications, the Schema Extraction Model is more appealing to ordinary users by freeing them from generating SQL queries.

Suppose a user is trying to find hotels in Manhattan. Using ExDB model he will need to come up with a SQL query expressing his concern about price and location. This query can produce a table of four columns

```
SELECT h.c1, l.c2, p.c2
FROM hotel as h, location as l, price as p
WHERE h.c1=l.c1 & l.c1=p.c1
        & l.c2= “Manhattan”
ORDER BY p.c2 ASC
```

Figure: SQL query

Hotel	Location	Price	Probability
Motel 6	Manhattan	110	0.3
Holiday Inn	Manhattan	120	0.1
Holiday Inn	Manhattan	220	0.9
Hilton	Manhattan	330	0.8
…			

However, this approach can have several draw backs.

1. The user need to know SQL and spend time to write it
2. Without first showing the user some statistics, they may lack of the domain knowledge about what are the available semantic types and relations, and what factors are important. For example, beside price and location, another important factor for hotels is the quality of service. A novice user may easily ignore this factor and make a decision that he latter regret
3. From the performance point of view, since everyone who investigates hotels would be interested in location, price, and quality of service, it is quite reasonable that the system should pre-compute this table offline, instead of doing the costly join operations online again and again for many users.

Different than ExDB model, the Schema Extraction Model does a further offline integration step to derive a single best schema for the corpus. User’s new work flow become as below.

1. User enters a keyword query “hotel”. The system retrieves tables most relevant to “hotel”.
2. By observing schema of the table and sample records, the user decide that he need constraint of Location= “Manhattan” and Price<200, and service= “excellent”. Some

handy UI should let user interact with the table and specify these constraints

3. Records that satisfy the constraints are shown to the user.

Cafarella and et al. (2007) only gave design to the ExDB model. In this white paper I will propose algorithm to implement Schema Extraction Model, and design experiment for evaluation.

2. Greedy Algorithms

Consider the following structural information. A person can have *relations* (predicate with two arguments, like *born-in, parent* etc.), *semantic type* (predicates with one argument, like *inventor, American*). An *inventor* can have all these, plus his/her *invention*. How are we going to generate schema for the above structural information automatically?

2.1 A Naïve Approach

The most naïve approach for schema extraction is to create a table for each predicate, generating a two-column table for any relation and a one-column table for any semantic type. It is far from ideal, because information about an object is scattered in many tables, and thus hard to understand or use.

predicate <Birthday>

Edison	1870
Einstein	1880
…	

predicate <Invent>

Edison	phonograph
Edison	light bubble
…	

predicate <Inventor>

Edison
Da Vinci
...

2.2 Another Naïve Approach

Another naïve approach is to create a table for each semantic type, and add related relations to the table as extra columns.

1) Determine tables and rows,

- Each semantic type can potentially generate a table (city(), electronics(), etc.).
- The number of records in each table is equal to number of significant instance for that type (with confidence score above a threshold α_c).
- Ignore those semantic types (tables) with number of records less than a threshold N_s.

2) Determine columns

- Given all the instances of a table *t*, all relations related to these instances are potential columns.
- For each instance *i*, find the set of relations *R(i)* that are significant to it (with confidence score above a threshold α_r)
- The relations that are significant to more than β_r percent of the instances are considered significant to table *t*. They become the columns of the table.

semantic type <Inventor>

Inventor	**Born-in**	**Invent**
Edison	1847	Phonograph, light bubble, …
Da Vinci	1452	…

The potential problems with this method are

- **Multiplicity of a relation.** A cell of the table can contain many instances. E.g. an *inventor* may have produced a whole bunch of *inventions* all under the relation of *invent*. Special mechanism than traditional database table is needed to take care of this.
- **Redundancy.** There could be duplicated instance and attribute among tables. E.g. *Edison* can appear in table *American, man*, and *inventor*, and relations like *invent, born-in* can appear in all three tables. This redundancy can be removed if relations like *invent, born-in* only appears in table *inventor*.
- **Semantic type is not integrated in the table.** Consider the table *inventor*, it is desirable if the semantic types like *nationality* and *gender* can also be integrated in the table.

2.3 Consider Multiplicity of A Relation.

The diversity of a relation can be measured by entropy of the second argument given its first argument. Relations bearing divers target are considered *one-to-many* relations (e.g. *invent,*

be_friend). Other are considered *one-to-one* relations (birthday). For a one-to-one relation, each cell of its corresponding column will has the most frequent argument as its value. For a one-to-many relation, each cell of its corresponding column will be a pointer to a table containing all possible arguments (e.g. a table for Edison's inventions).

semantic type <Inventor>

Inventor	**Born-in**	**Invent**
Edison	1847	<Edison Invent>
Da Vinci	1452	<Da Vinci Invent>

List table <Edison Invent>

Edison Invent
phonograph
light bubble

2.4 Consider Semantic Types (Is-A Relation).

For is-a relations like *American, Italian* we do not want then to be separate columns with binary values, which inflate the size of table. These two semantic types can be conbined by leveraging hyponym knowledge. for example, if many instances of person have significant is-a relation with *American* or *Italian*, and both *American*, *Italian* have significant *is-a* relation with *nationality*, then *nationality* can become a column in the table *person* and each cell of this column can take on the value *American* or *Italian* depending on with relation is more significant to the person.

semantic type <Inventor>

Inventor	**Born-in**	**Invent**	**Nationality**
Edison	1847	<Edison Invent>	American
Da Vinci	1452	<Da Vinci Invent>	Italian

List table <Edison Invent>

Edison Invent
phonograph
light bubble

semantic type <nationality >

Nationality
American
Italian

2.5 Consider Redundancy

Until now none of the above rules consider redundancy of information among tables. This issue will be addressed in the next section where schema extraction is formulated as an Optimization Problem.
Should distinguish attribute and super. (the difference between C++ inheritance and that of Java)

3 Schema Extraction as an Optimization Problem

Schema Extraction can be seen as a structural learning problem which tries to derive a single best schema for the whole corpus by optimizing the following contradicting criterions.

- **Completeness:** all extractions from text appear in the output
- **Simplicity:** the output has few tables
- **Fullness:** the output database has few NULLs

It would be a more principled way than those greedy algorithms of section 2, if we can write down an objective function to reflect the above principles, and optimize schema accordingly with some procedure. An optimization process would allow modification of the schema towards a better fitness to the data, and potentially solve the data redundancy problem in section 2.2.

Given a set of relations *Rel* and semantic types *Type* extracted for the corpus, the fitness of a schema *Sch* is defined as

$$Fitness(Sch, rel, Type) = \sum_{r \in rel} p(r)Sch(r) + \alpha \sum_{t \in Type} p(t)Sch(t) - \gamma \#cells(Sch) - \lambda \#table(Sch)$$

Where *p(r), p(t)* are the extraction confidence for relation *r* and semantic type *t* respectively.
Sch(r),Sch(t) indicate whether relation *r* and semantic type *t* are expressed in the schema.
#table(Sch), #cells(Sch) are the total number of tables and cells in the schema
α, β, γ are parameter tunable for balancing among the three criterions.

A schema can be defined by the following series of decisions

- For each semantic type *t*, whether it generates a table
- For each instance *i* of *t*, whether it generates a record
- For each relation *r* related to these instances, whether it becomes a column
- For each semantic type *t'* related to these instances, whether it becomes a column
- whether hyponym of *t'* becomes a column
- For each relation *r*, whether it is one-to-many mapping

These decisions can be visualized as a *schema tree*. As the potential actions to modify the tree can be adding a node to it, or deleting an existing node, a straightforward optimization procedure would be

Pseudo code:
Input: *Sch, Rel, Type*
Output: *Sch*
1 repeat
2 generate potential actions
3 calculate their potential improvement fitness
4 find action *a* that improve fitness the most
5 break if the improvement is negative
6 apply *a* to *Sch*
7 end

In order to properly guide the optimization of schema in this *tree space*, we need to consider the issues of computation complexity and local optima.

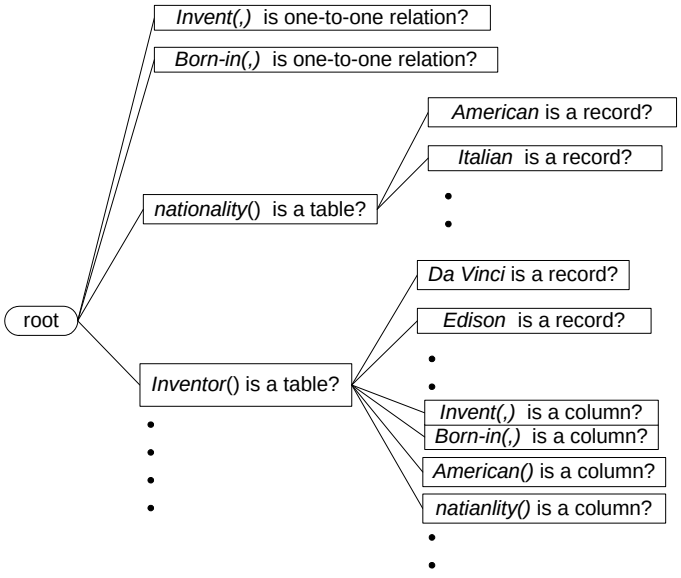

Figure: A Schema Tree

We start optimization from the result of algorithm in section 2 to minimize the likelihood of getting stuck in local optima. But there are still great potential for local optima. For example, without specifying columns for a table, the value of adding this table is not accurate, and the decision of adding/deleting this table can be short sighted. Some experiment on real data is needed to find out how bad this local optima problem can be, and explore potential solutions.

The complexity of optimization procedure can be analyzed as the following. Let *Nr, Nt* be the number of extracted relations and semantic types respectively. Let *Mr, Mt* be the number of unique predicates of these relations and semantic types respectively. Then the number of possible tables is *O(Mt)*. The number of potential columns of all table is *O(Nr+Nt)*. The number of potential record of all table is *O(Nt)*. So at each iteration the number of potential actions is *O(Mt+Nr+2Nt)*. The size of a fully grown schema tree is also *O(Mt+Nr+2Nt)*, which is the expected iterations needed to grow such a tree. Overall the optimization complexity is $O(Mt+Nr+2Nt)^2$.

4. Evaluation and Plan of Work

As structured assess to the web is supposed to be good at finding facts. The proposed system is to be evaluated by fact finding tasks. For example,

- "list the prices of flat screen HDTVs larger than 40 inches with 1080p resolution at shops in the nearest town that are open until 8pm on Tuesday evenings"
- "list all countries that have donated money to the Gujarati earth quake, how much they donated, and when"
- These tasks are typical in QA list questions, and information distillation queries. Therefore, the proposed method can be compared to ad-hoc document retrieval systems, question answering systems. The performance can be measured by how many correct information a user has found versus how much time user has spent interacting with the systems. Corpora of TREC QA, and GALE information distillation tasks can be used for evaluation.

5. Future Works

Given the optimal schema extracted for a corpus, the natural next step is to design a proper querying language and relevance model for users to navigate through the myriads of tables.