International
Virtual
Observatory
Alliance

Ranking Query Result Sets

Version 0.10

IVOA Note 2006 May 10

Interest/Working Group:
http://www.ivoa.net/twiki/bin/view/IVOA/IvoaDAL

Author(s):
Markus Dolensky, Bruno Rino

Abstract

The purpose of this document is to describe possible probabilistic scoring methods for database query results. The IVOA Simple Spectrum Access Protocol [SSA IF] introduces the concept of a scoring mechanism to rank matching records by relevance. The intention is to provide guidelines for design and implementation without tying it closely to a specific VO protocol.

Status of This Document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

Acknowledgements

Thanks to Tobias Scholl, Technical Univ. Munich, for reviewing an early sketch of this document, to Johan Lindroos, ESO/SAMPO, for his feedback on query operators and to Andreas Wicenec, ESO, for extensive discussion of initial ideas. Arnold Rots, SAO, is acknowledged for inspiring the concept of ranking records via a score heuristic.

Contents

1 Introduction

How to pick the most interesting candidates from a wealth of records satisfying a query?

This documents provides some hints on how to help human users in making the right choice at the first attempt. Obviously there is no right or wrong in preferring one data set over the other. What a machine can do though is offering a statistical probability based on a priori knowledge and based on the analysis of the query parameters.

In the context of result sets returned by VO standard protocols such as [ADQL] or SSA there is quite a bit of known structure to it as compared to a classic Google type search scenario. Adopting an indexation algorithm that treats a [VOTable] like a free text document would defeat the purpose of using XML in first place. The attribute vectors (records) to be ranked have mostly got assigned data types, units, (error) bounds, semantic concepts [UCD] and even string values are mostly part of some controlled vocabulary. Therefore, one can narrow down the problem definition as follows...

2 Problem Definition

Let's consider the simplest problem instance whereby an inventory of real or virtual data products is stored in a database table *D*. *D* has got n tuples $\{t_1,...,t_n\}$ over a set of attributes $A = \{A_1,...,A_m\}$. A query *Q* is defined as an expression of constraints

$$Q := X_1\ Op_1\ x_1\ AND\ ...\ AND\ X_s\ Op_s\ x_s$$

where X_i is an attribute from *A* and x_i is its value. Op_i denotes a query operator such as =, >=, boolean NOT etc.

For the design of a probabilistic algorithm it is important to distinguish the set of of known attributes $X = \{X_1,...,X_s\} \subseteq A$ from the set of unspecified attributes $Y = A - X$.

S denotes the answer set $S \subseteq \{t_1,..., t_n\}$ of *Q*. The problem to solve occurs when the query is resulting in a large *S*. Large means that a human user requires assistance in efficiently choosing a subset that is sufficient to satisfy the specific purpose of the individual archival research. This is of particular interest in usage scenarios where only a single item (best match) or a very limited number of items is desirable and the query is imprecise.

For the sake of simplicity above problem instance neglects the possibility of nested boolean expressions and missing values. This document does not attempt a detailed analytical discussion but proposes some pragmatic approach to satisfy the following requirements ...

3 Requirements

Below a summary of requirements that the authors deem necessary for an implementation of a probabalistic scoring heuristic that can be applied to typical SSA, ADQL, etc. usage scenarios.

R1 Returning score as a number:
The relevance, i.e., score of each record in result set *S* shall be expressed as a single, positive floating point number. The higher the probability that a record is relevant to a query *Q*, the higher its value.

R2 Context sensitivity:

The algorithm must be capable to take at least two independent contributing components into account:

Firstly, the relevance of t must depend on Q. In other words, no $t \in$ D is a priori more relevant than any other one. The algorithm has to provide a measure on how closely tuple t_i is matching the set of constraints X.

Note:

> This implies that the score of a tuple t_i is computed at the time of the query because in the general case it is not feasible to pre-compute and store all possible scores for all possible queries in advance. This does not prevent the computation of statistics in advance to increase efficiency.

Note:

> A feedback loop, for instance, may change the score dynamically. Therefore, the relevance of t for a given Q may change over time.

Secondly, the importance of an attribute with respect to the others in A. The relevance of an attribute shall be defined independent of Q. As a bare minimum it is required to weight user provided constraints X differently from unspecified attributes (defaults) denoted as set Y.

R3 Generality:
The algorithm must be generic, so that optional and service specific constraints can be supported.

In particular, it must not restrict its applicability to a handful of commonly used parameters like celestial position as this would add little or no value to existing search interfaces which often provide positional plotting capabilities or even a method to cross match.

R4 Keep quality out of the game:
The score must be kept separate from any quality measure of a datum. Even data of perfect objective quality may score zero when they are irrelevant to a particular search. Conversely, the seemingly worst dataset may score highest if the goal of the query is to look for the worst.

R5 VO Compliance:
Implementations shall be compliant with current IVOA standards where applicable. For instance, it must be possible to integrate the ranking as part of the common VO protocols such as VOTable result sets with a defined *utype* attribute.

4 Measuring the Quality of a Match

When breaking down the problem into patterns for certain classes of attributes then one can devise a method for each attribute class that is tailored to measure one aspect of the quality of a match.

An example:
The fact that a higher spectral or spacial resolution should be considered *better* if two observations are otherwise similar is obvious to the domain expert. It is, however, not obvious how to express *better* in the context of a specific encoding in a database. If the spectral resolution is given in nm then the smaller the number, the higher the resolution. If, however the resolution is given as an absolute number like, say R=500 then a bigger number like R=600 is better in the general case.

Such considerations lead to a number of basic methods that apply to a large fraction of potential attributes in Astronomy data collections. For instance, the numeric difference is a reasonable first order in indicator for many measured quantities:

$$\textit{numeric distance}(x_i) := norm(abs(X_i - x_i))$$

abs() returns the absolute value. More interesting is the normalization function *norm()*. It is up to the ranking algorithm to define it. One possible definition is to normalize against the dynamic *range* of attribute X_i in database *D*:

$$norm(xi) = \begin{cases} 0 & for\, abs(Xi - xi) > range \\ 1 & forXi - xi = 0 \\ \left(1 - \dfrac{1}{\dfrac{range}{abs(Xi - xi)}}\right) & for\, abs(Xi - xi) \le range \end{cases}$$

Example: $X_i = 5$, x_i=4.5, dynamic range = (3,8) $\Rightarrow$ 5

norm(abs(5 - 4.5)) = $\underline{0.9}$

A completely different method is required to deal with enumerated values (controlled vocabularies). The following method choice() defines a way to judge the applicability of several data formats that are available for a given data product:

$$choice(xi) = \begin{cases} 0.1 & for\,image/jpeg \\ 0.2 & for\,image/png \\ 0.8 & for\,application/fits \\ 1.0 & for\,application/x-votable+xml \end{cases}$$

Other common patterns are matching substrings. A normalization of the length substring against the length of the given text may yield a measure for the quality of a match.

Again, there is no limit to possible methods. It is important to the domain expert to have a set of methods that is applicable to *A*.

5 Design Considerations

This chapter outlines a number of ideas and design considerations for the implementation.

In the context of an SSA service one can view the scoring method as an optional sub-service that can be switched on and off.

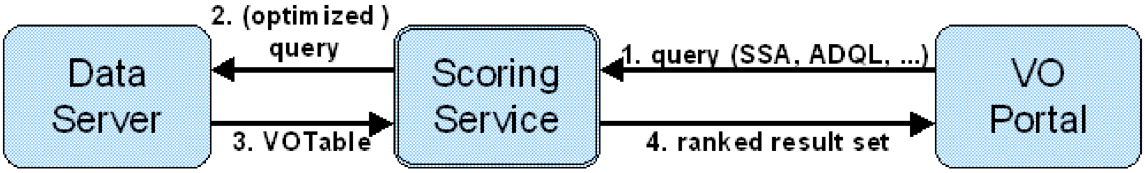

Figure 1: Service architecure.

A ranking mechanism may gather input from a number of sources (Figure 2).

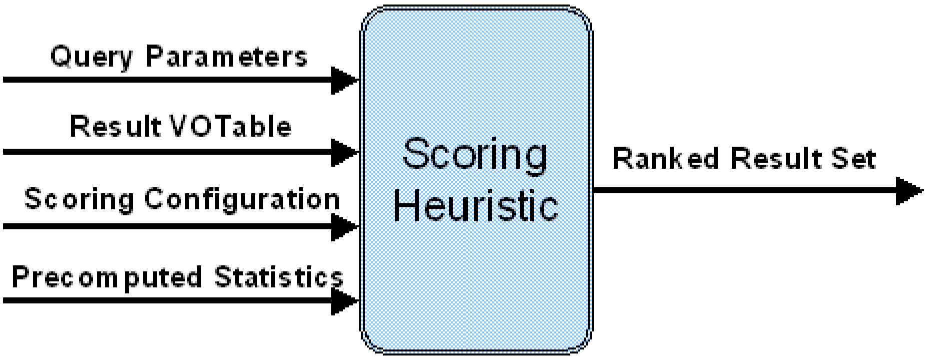

Figure 2: I/O of a scoring heuristic.

Most importantly there is the actual set of query parameters. Then there is the VOTable that carries S or some subset of attributes thereof. Its score column gets populated by the ranking system (Figure 3).

```
<FIELD name="score" datatype="float"
ucd="meta.code.number;stat.likelihood" uytpe="ssa:Query.Score"/>
```

Figure 3: Score column definition in a VOTable V1.1 with UCD 1+.

There is also a priori knowledge which should be exploited as much as possible because a human user tends to provide only the minimum of information on top of application defaults. A priori knowledge may have various origins such as:

- domain expert (archive scientist)
- pre-computed statistics on the data collection
- worklog captured from previous user sessions (feedback loop)

In order to capture knowledge of a domain expert who can accurately characterise a data collection there should be a configuration mechanism. The domain expert may, for instance, apply global weights to each attribute in *A* and specify a method that is suited to measure the quality of a query match.

Modularize the algorithm so that the total score is a sum over contributing attributes or related groups of attributes (Figure 4).

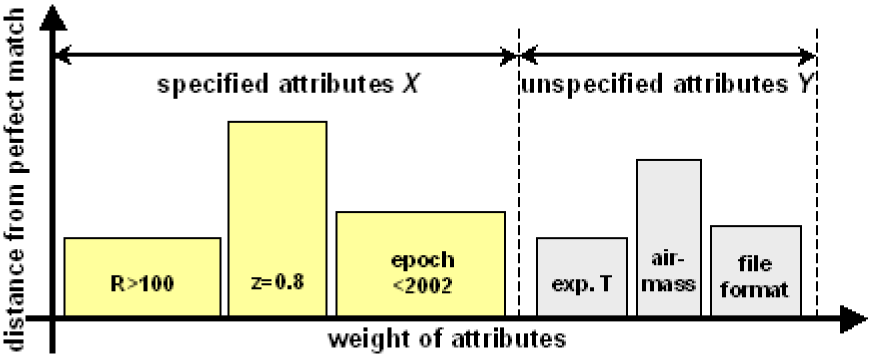

Figure 4: Matching a given tuple with the query constraints. The score is indirectly proportional to the integrated area of the boxes: Each attribute has got a weight (width) whereby wider means more important. User specified parameters *X* are generally more relevant than 'guessed' or unspecified query parameters *Y* which may be ignored completely in a simple implementation. The smaller the distance from a perfect match (height), the better.

Note:

Normalizing the total score is not recommended as this would create the impression to the end user that the score is absolute and could be used to compare the score of different services; in reality the score is strictly relative to a single query from a single service. A client application may always choose to normalize later on. Normalization is a global operation on *S* and prohibits the streaming of results.

As discussed in section 4 the general problem can be broken down into patterns for certain classes of DB attributes.

Scores or partial scores may be cached for efficiency. In general it is important for the design to distinguish between steps that can be pre-computed such as weight maps, statistics based on previous queries and steps that can be taken at the time of the actual query only.

A possible strategy is to bin attribute value ranges and to assign weight factors to each bin of each attribute. For a query instance *Q* it is then a matter of looking up the weights for the given bin of X_i and to put it into the scoring formula. Interpolating weight factors of the two neighboring bins will yield higher precision.

Y (set of unspecified or default params) is usually way larger than *X* (user specified). By keeping track of what users searched for and which data actually were retrieved (e.g. SSA has a distinct query and retrieval mode) one can identify *successful* searches from pointless ones and thereby infer *y* from the actual values of selected (purchased) tuples.

Consider running the scoring algorithm as a service detached from the data: Given that the DB structure is known, for instance, after exploring it via a [SkyNode] interface, plus a given attribute characteristics document which needs to be generated only once by a domain expert it becomes possible to compute scores also for queries against remote SkyNodes.

Some thought needs to be given on how to capture and channel back workload information to the scoring system. This is about how to learn from previous queries and user selections over time. It boils down to the question: “How likely is it that a user who requests value *x* of attribute *X* also is interested in value *y* of attribute *Y*?“ Example from E-shop: People who bought this item also purchased those items. This area is subject to future investigations.

Computing scores for a huge result set is potentially very costly: Hence, it is desirable to refine *Q* before its execution such that it only returns a reduced set *S*. Some recent papers about Top-k scoring and automated query tuning are [RankSQL] and [Selftune].

A further optimization is to define several configuration for specific scenarios. A domain expert would define a set of weight maps and specify different ranking methods depending on the context. This raises the question of how to trigger or select a configuration. Introducing a ranking-specific query parameter would make the ranking system more intrusive on the user side.

Finally, the design should allow for shortcuts when Q yields a trivial result like for instance no matching records, or everything matches or there is one perfect match.

References

[ADQL] Astronomical Data Query Language V1.01
http://www.ivoa.net/Documents/WD/ADQL/ADQL-20050624.pdf

[SkyNode] SkyNode Interface V1.01
http://www.ivoa.net/Documents/WD/SNI/SkyNodeInterface-20050624.pdf

[SSA IF] Simple Spectrum Access Protocol V0.90, interface definition
http://www.ivoa.net/internal/IVOA/InterOpMay2005DAL/ssa-v090.pdf

[UCD] The UCD1+ controlled vocabulary V1.11
http://www.ivoa.net/Documents/REC/UCD/UCDlist-20051231.html

[VOTable] VOTable Format Specification V1.1
http://www.ivoa.net/Documents/REC/VOTable/VOTable-20040811.pdf

[RankSQL] RankSQL: Query Algebra and Optimization for Relational Top-k Queries,
http://eagle.cs.uiuc.edu/pubs/2005/ranksql-sigmod05-lcis-mar05.pdf

[Selftune] Efficient and Self-tuning Incremental Query Expansion for Top-k Query Processing
http://www.mpi-sb.mpg.de/~mtb/pub/sigir05_incrmerge.pdf