Keyword Search over Data Service Integration for Accurate Results

Vidmantas Zemleris
Vilnius University, Lithuania
vidmantas.zemleris@cern.ch
for the benefit of CMS Collaboration

Valentin Kuznetsov
Cornell University, USA
vkuznet@gmail.com

Summary

Virtual data integration aims at providing a coherent interface for querying heterogeneous data sources (e.g. web services, proprietary systems) with minimum upfront effort in integration. Data is usually accessed through a structured queries, such as SQL, requiring to learn the language and to get acquainted with data organization, which may pose problems even to proficient users.

We present a keyword search system, which proposes a ranked list of structured queries along with their explanations. It operates mainly on the metadata, such as the constraints on inputs accepted by services. It was developed as an integral part of the CMS data discovery service and is currently available as open source.

Context: a system for Virtual Data Integration

"CMS Data Aggregation System" (DAS):

- accepts simple structured queries
- integrates heterogeneous services
  - → parse the query
  - → contact services
  - → eliminate inconsistencies in the responses:
    - ∗ entity naming; data formats (XML, JSON)
  - → combine the responses
- requires only minimal service mappings
  - → no predefined schema
  - → minimal effort in defining services

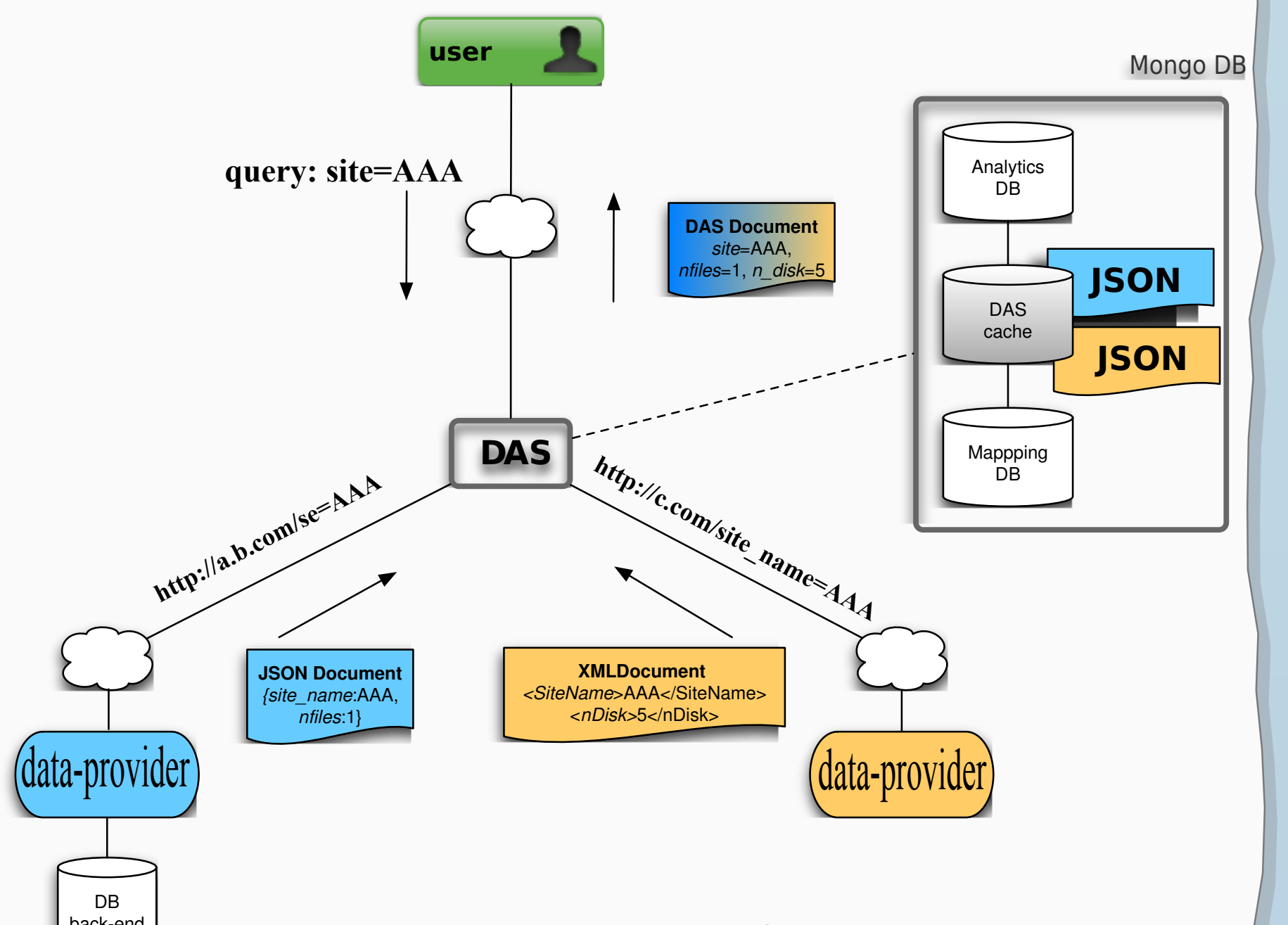


Figure 1: DAS workflow

Queries must specify: entity to be retrieved and filtering criteria. Optionally, the results can be further filtered, sorted or aggregated

$$\underbrace{\textbf{dataset}}_{\text{entity requested from services}}\quad \underbrace{\textbf{dataset=*RelVal*}}_{\text{conditions as service *inputs*}}\ \textbf{|}\ \underbrace{\textbf{grep dataset.nevents >1000}}_{\text{filters and projections on service *outputs*}}\ \textbf{|}\ \underbrace{\textbf{avg(dataset.size), median(dataset.size)}}_{\text{aggregators}}$$

still, it is overwhelming for users to:

- learn the query language
- remember how exactly the data is structured and named

Could keyword queries solve this?

Interpreting Keyword Queries: Problem definition

Input: query, KWQ=$(kw_1, kw_2, .., kw_n)$
ambigous; nearby keywords are often related

Task: translate it into structured query
(made of tag_j - schema and value terms, *unknown*, operators)

Given: metadata only:

- names of entities and their attributes
  - → *service inputs* or their *output* fields
- possible values (only for some inputs)
- *constraints* on data-service *inputs*:
  - → mandatory inputs
  - → regular expressions on values

Example. Consider this query: average size of RelVal datasets with its number of events > 1000

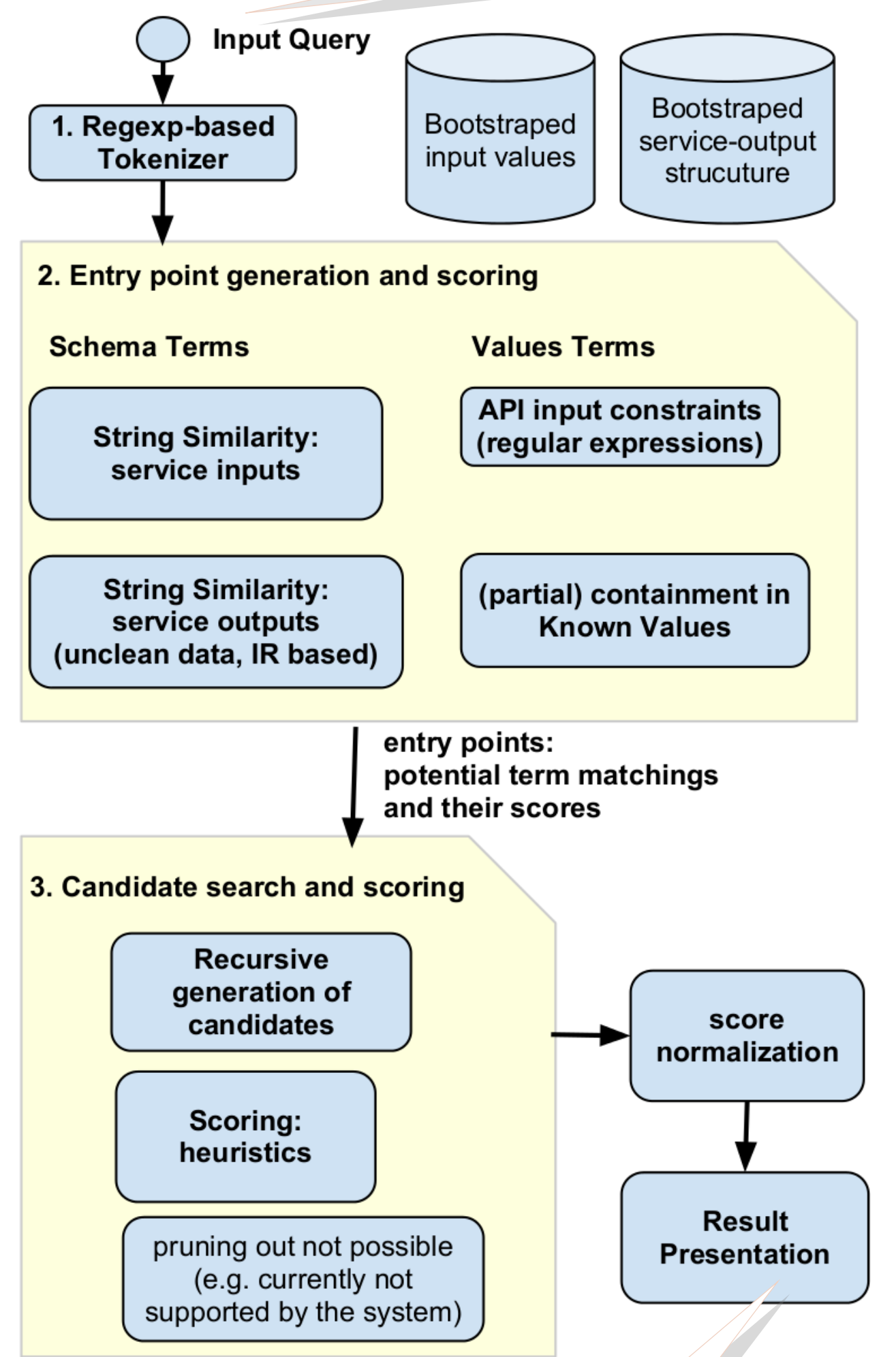
- average RelVal dataset size nevents>1000
- avg(dataset size) RelVal "number of events">1000

For all, the expected result is:

$$\underbrace{\textbf{dataset}}_{\text{entity requested from services}}\quad \underbrace{\textbf{dataset=*RelVal*}}_{\text{conditions as service *inputs*}}\ \textbf{|}\ \underbrace{\textbf{grep dataset.nevents >1000}}_{\text{filters and projections on service *outputs*}}\ \textbf{|}\ \underbrace{\textbf{avg(dataset.size)}}_{\text{aggregators}}$$

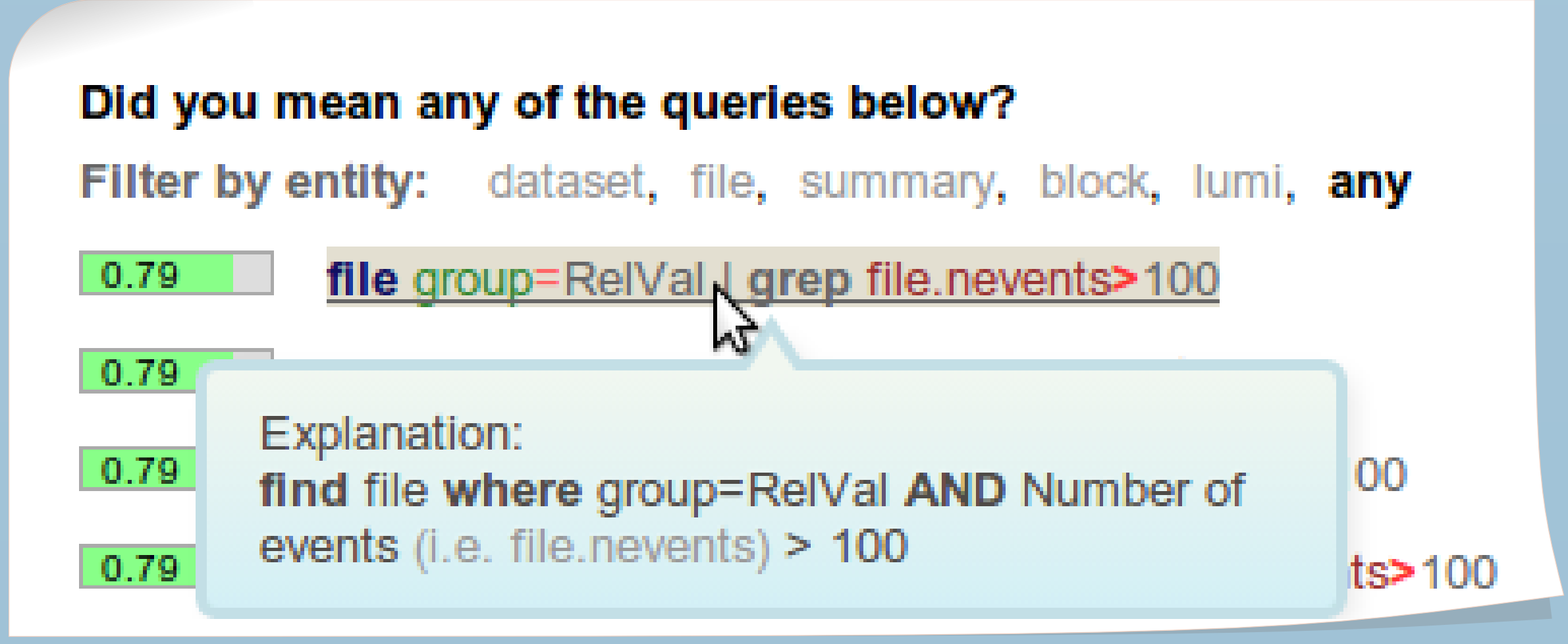
Keyword search overview

- tokenizer:
  - → clean up the query
  - → identify patterns
- identify and score "*entry points*" with
  - → string matching [for entity names]
  - → IR (IDF-based) [output fieldnames]
  - → list of known values
  - → regular expressions on allowed values
- combine *entry points*
  - → consider various *entry point* permutations (keyword labelings)
  - → promote ones respecting keyword dependencies or other heuristics
  - → interpret as structured queries



Input Query
1. Regexp-based Tokenizer
Bootstraped input values
Bootstraped service-output strucuture
2. Entry point generation and scoring
Schema Terms
Values Terms
String Similarity: service inputs
API input constraints (regular expressions)
String Similarity: service outputs (unclean data, IR based)
(partial) containment in Known Values
entry points: potential term matchings and their scores
3. Candidate search and scoring
Recursive generation of candidates
Scoring: heuristics
pruning out not possible (e.g. currently not supported by the system)
score normalization
Result Presentation

More info:



Did you mean any of the queries below?

Filter by entity: dataset, file, summary, block, lumi, **any**

0.79 file group=RelVal | grep file.nevents>100

0.79

0.79 00

0.79 ts>100

Explanation:
find file **where** group=RelVal **AND** Number of events (i.e. file.nevents) > 100

Challenges

- keyword queries are ambiguous → return ranked list of structured query suggestions
- no direct access to the data - querying services is "expensive"
  - → rely on metadata
    - ∗ bootstrap list of allowed values (available only for some fields)
    - ∗ rely on *regexps* with lower confidence (can result in false positives)
- no predefined schema
  - → bootstrap list of fields in service results through queries
  - → some field names are unclean → use IDF (as they come directly from JSON/XML responses)

The ranker

For simplicity it is based on exhaustive search:

- allows finding optimal solutions
- easy to filter out many "invalid" candidates that are not yet supported by services
- our schema is quite small
  - → *cython*-based implementation is already quite fast (bound by MongoDB/IR engine to retrieve entry points)

Scoring function

$$final\ score = \sum_{i=1}^{|KWQ|}\left(\log\left(score_{tag_i|kw_i}\right) + \sum_{h_j\in H} h_j(tag_i|kw_i; tag_{i-1,..,1})\right)$$

$score_{tag_i|kw_i}$ - likelihood of kw_i to be tag_i (from entry points step)[a]
$h_j(tag_i|kw_i; tag_{i-1,..,1})$ - the score boost returned by heuristic h_j given the tag(s) nearby.

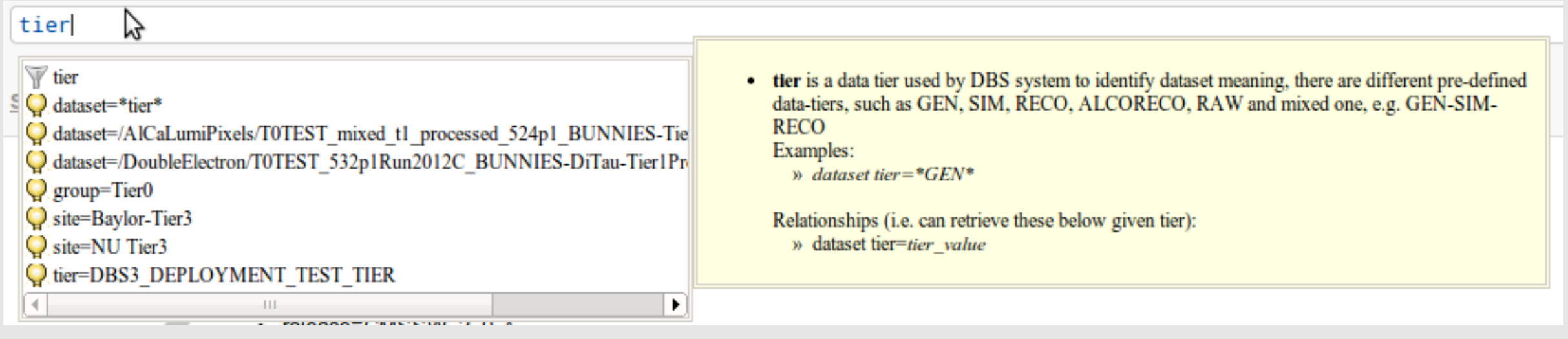
Our finding: summing log-likelihoods is better than plain scores (cf. Keymantic)

[a] $score(tag_i = unknown|kw_i)$ currently uses predefined $P(unknown)$.

Related works

- Keymantic (the closest work)
  1. score keyword mappings individually (entry points)
  2. solve *"weighted bipartite assignment"* ($kw_i \rightarrow tag_j$) *with contextualizations:*
     - → maximize total sum of weights
     - → uses heuristics to account for keyword interdependencies (contextualization)
       - ∗ e.g. <table_name> <attribute>; <attribute> <its value>;
       - ∗ solves it *approximately* with Munkres algorithm modified to consider *contextualizations:*
         - · contextualize - modify weights of $kw_i \rightarrow tag_j$, if tag_j is "related" to earlier sub-assignments
         - · to get multiple results, repeat recursively forcing/preventing certain sub-assignments
  3. interpret generated mappings as SQL queries
- KEYRY - uses HMM (Hidden Markov Model) to label keywords as schema terms
  - → HMM's initial parameters can be estimated from similar heuristics as above
  - → later machine learning can be used (if logs available)

Autocompletion to ease typing queries (prototype)



tier

- tier
- dataset=*tier*
- dataset=/AlCaLumiPixels/T0TEST_mixed_t1_processed_524p1_BUNNIES-Tie
- dataset=/DoubleElectron/T0TEST_532p1Run2012C_BUNNIES-DiTau-Tier1Pr
- group=Tier0
- site=Baylor-Tier3
- site=NU Tier3
- tier=DBS3_DEPLOYMENT_TEST_TIER

- **tier** is a data tier used by DBS system to identify dataset meaning, there are different pre-defined data-tiers, such as GEN, SIM, RECO, ALCORECO, RAW and mixed one, e.g. GEN-SIM-RECO
  Examples:
  » *dataset tier=*GEN**

  Relationships (i.e. can retrieve these below given tier):
  » dataset tier=*tier_value*

Future work & Interesting problems

- improve autocompletion prototype
- improve the ranker
- generic ways to improve services' performance, e.g. *materialized views with incremental refresh*

Open problems & ideas

Top-k (semi-)optimal solutions with Contextualization?

- can we adapt Murty's/Munkres's algs. for top-k list of assignments with contextualizations?
  - → this would at least guarantee optimal top-k for with **some** contextualization
  - → out of scope, ask for handouts/chat

Problematics of the HMM approach:

- what is modelled is not necessarily same as seen by user
  - → models $kw_i \rightarrow tag_j$, while user sees structured queries
  - → therefore, hard to automatically collect training data