Text Mining Applied to SQL Queries: A Case Study for the SDSS SkyServer

Vitor Hirota Makiyama
CAP – INPE
São José dos Campos
São Paulo - Brazil
vitor.hirota@gmail.com

M. Jordan Raddick
Physics and Astronomy Dept.
The Johns Hopkins University
Baltimore, Maryland, USA
raddick@jhu.edu

Rafael D. C. Santos
LAC – INPE
São José dos Campos
São Paulo - Brazil
rafael.santos@inpe.br

Abstract

SkyServer, the portal for the Sloan Digital Sky Survey (SDSS) catalog, provides data access tools for astronomers and scientific education. One of the interfaces allows users to enter *ad hoc* SQL statements to query the catalog, and has logged over 280 million queries since 2001. This paper describes text mining techniques and preliminary results on mining the logs of the SQL queries submitted to SkyServer, along with what other applications we foresee for such procedure.

1 Introduction

With the increase in data collection and generation, datasets are growing at an exponential pace, making a real challenge to make available all the data being produced. As a solution, some large scientific datasets have been made available through publicly accessible RDBMSes (Relational Database Management Systems). In which scientists and interested users can query and analyze only the most relevant and up-to-date data for their needs.

The Sloan Digital Survey is one such case. It makes available the largest astronomy survey to date through SkyServer[1], its Internet portal that allows users and astronomers to query the database and even perform data mining tasks using SQL (Standard Query Language), the *de facto* standard to query relational databases. The portal, in operation since 2001, has proven to be extremely popular, with over 1.5 billion page hits and almost 280 million SQL queries submitted.

Since 2003, SkyServer has been logging every query submitted to the portal. It collects access information, such as timestamp, user ip address, the tool used to submit the query, and the target data release (DR1, DR2, etc); and query information, e.g. the SQL statement, query success or failure and error message, number of rows returned, elapsed time. This data can be used to generate summarized access statistics, like queries per month or data release query distribution over time, as presented by Raddick et al. (2014). But for a more in depth usage analysis, data has to be processed and transformed, like Zhang et al. (2012), which color codes SQL queries for visual analysis and also presents a visual sky map of popular searched areas.

To further analyze such queries, this paper aims to apply text mining techniques with the goal to define a procedure to parse, clean and tokenize statements into a weighted numerical representation, which can then be fed into regular machine learning algorithms for data mining.

We proceed with an exploratory analysis, where we project part of the historical queries into a low dimensional representation and correlate the results with sample templates defined in the SkyServer help pages, a list of predefined queries ranging from Basic SQL, showing simple SQL structures; to specific examples on how to find Stars, Galaxies or Quasars.

2 Text Mining and SQL Queries

Text mining, or Knowledge Discovery in Texts (KDT), is an extension to the traditional Knowledge Discovery in Databases (KDD), the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data (Fayyad et al., 1996), but targeting unstructured or semi-structured data instead of regular databases, such as emails, full-text documents and markup files (e.g., HTML and XML). It is a multidisciplinary field involving, among others, information retrieval and extraction, machine learning, natural language processing, database technology and visualization (Tan, 1999).

[1] http://skyserver.sdss3.org

SQL queries in this context can be seen as mini-documents. As a well defined language, we can leverage the structure provided by the language in order to fine-tune and optimize the preprocessing step of queries to suit the specific cases found. For instance, there is no need for stop words removal, and by analyzing the token type (table name, column, variable, expression, constant, etc) we can perform a different normalization or substitution.

3 Methodology

The methodology followed is the traditional KDD process, comprising the following phases: selection, preprocessing, transformation, data mining, and interpretation/evaluation, with each phase briefly discussed below.

3.1 Selection

For this paper, we used a normalized version of the raw data made available by Raddick et al. (2014) which analyzed a 10-year span of log data (12/2002 to 09/2012), amounting to almost 195 million records and 68 million unique queries.

As a proof-of-concept, we filtered the queries to those coming from the last version of the online SQL search tool (skyserver.sdss3.org), which only allows SELECT statements and has a timeout of 10 minutes. The assumption was to have a dataset with less variance and complexity. This filter also restricted queries with errors and no rows returned, resulting in a final dataset of 1.3 million queries.

3.2 Preprocessing and Transformation

The main objective of the preprocessing phase is to parse the text queries into a *bag-of-words* like representation, but instead of just the set of tokens present in each document, we also keep the count of each token in that statement.

As noted before, we can leverage the fact that SQL is a structured language, by using a proper parser and add a layer of metadata on top of each token. Knowing what kind of token we are processing, we can add specific actions for each token type.

Since SkyServer uses Microsoft SQL Server as its RDBMS, we extended the readily available .NET T-SQL parser library to build a custom one. Other than normalizing case sensitivity, the custom parser also removes constants (strings and numbers), database namespaces, and aliases; substitutes temporary table names, logical and conditional operators for keywords; and qualified each token with the SQL group, e.g. *select, from, where, groupby, orderby*. Substitutions and filters were performed with the intention to remove tokens that are trivial (such as database namespaces) or too specific (such as constants, table aliases, or arithmetic operations), and thus, would be of little contribution in discriminating or grouping each query within the dataset.

An example of the original statement and its normalized version is shown in Figure 1. Figure 2 shows the final feature vector.

```
SELECT p.objid, p.ra, p.dec,
       p.u, p.g, p.r, p.i, p.z,
       platex.plate, s.fiberid,
       s.elodiefeh
FROM   photoobj p,
       dbo.fgetnearbyobjeq(1.62917,
         27.6417, 30) n,
       specobj s, platex
WHERE  p.objid = n.objid
       AND p.objid = s.bestobjid
       AND s.plateid =
         platex.plateid
       AND class = 'star'
       AND p.r >= 14
       AND p.r <= 22.5
       AND p.g >= 15
       AND p.g <= 23
       AND platex.plate = 2803
```

(a) Raw SQL query.

```
select objid ra dec u g r i z
       plate fiberid elodiefeh
from   photoobj fgetnearbyobjeq
       specobj platex
where  objid objid logic objid
       bestobjid logic plateid
       plateid logic class logic
       r logic r logic g logic g
       logic plate
```

(b) Tokenized SQL.

Figure 1: Example of a SQL query and its normalized version. Whitespace is included for readability.

It is important to note that, since the parser is strict, it can only process syntax valid statements.

Lastly, we weight tokens according to its frequency, so the most common or unusual rare tokens are balanced to have more or less con-

select_objid	1
select_ra	1
select_dec	1
select_u	1
select_g	1
select_r	1
select_i	1
select_z	1
select_plate	1
select_fiberid	1
select_elodiefeh	1
from_photoobj	1
from_fgetnearbyobjeq	1
from_specobj	1
from_platex	1
where_objid	3
where_logic	8
where_bestobjid	1
where_plateid	2
where_class	1
where_r	1
where_g	2
where_plate	1

Figure 2: Feature vector.

tribution in its power of discrimination. One of the most popular weighting scheme is the TF*IDF (term frequency times inverse document frequency), which assigns the largest weight to terms that arise with high frequency in individual documents, but are at the same time, relatively rare in the collection as a whole (Salton et al., 1975).

3.3 Data Mining

On a general perspective from data analysis, clustering is the exploratory procedure that organizes a collection of patterns into natural groupings based on a given association measure (Jain et al., 1999). Intuitively, patterns within a cluster are much more alike between each other, while being as different as possible to patterns belonging to a different cluster.

In text mining, clustering can be used to summarize contents of a document collection (Larsen and Aone, 1999). So, with this idea in mind, what kind of summarization could be done over the historic SQL logs and how such summary would compare to the predefined templates? For that, we apply in this paper the Self-Organizing Map (SOM) algorithm.

3.3.1 Self-Organizing Maps

Kohonen's SOM (Kohonen, 2001) is a neural network algorithm that performs unsupervised learning. It implements an orderly mapping of high-dimensional data into a regular low-dimensional grid or matrix, reducing the original data dimension while preserving topological and metric relationships of the data (Kohonen, 1998).

The SOM consist of M units located on a regular grid. The grid is usually one- or two-dimensional, particularly when the objective is to use the SOM for data visualization. Each unit j has a prototype vector $m_j = [m_{j1}, ..., m_{jd}]$ in a location r_j, where d represent the dimension of a data item. The map adjusts to the data by adapting the values of its prototype vectors during the training phase. At each training step t a sample data vector $x_i = [x_{i1}, ..., x_{id}]$ is chosen and the distances between x_i and all the prototype vectors are calculated to obtain the best-matching unit (BMU). Units topologically close to the BMU are then updated, moving their values towards x_i.

Distance calculation between the data vectors and prototypes on the SOM can be calculated using the Euclidean, Cosine or other metrics. The neighborhood considered around the BMU can be circular, square, hexagonal (to determine its shape) and the distance between an unit and the BMU can be weighted by a gaussian or difference-of-gaussians function so units closest to the BMU will be updated with different weights used by units further from it. During training the weights used for updating the units and the size of the neighborhood can change according to several different possible rules.

The algorithm has two interesting characteristics that suggest its use for data visualization: quantization and projection. Quantization refers to the creation of a set of prototype vectors which reproduce the original data set as well as possible, while projection try to find low dimensional coordinates that tries to preserve the distribution from the original high-dimensional data. The SOM algorithm has proved to be especially good at maintain the topology of the original dataset, meaning that if two data samples are close to each other in the grid, they are likely to be close in the original high-dimensional space data (Vesanto, 2002).

These features and the possible variations and parameters of the Self-Organizing Map makes it an interesting tool for exploratory data analysis, particularly for visualization (Morais et al., 2014; Vesanto, 2002). There are three main categories of SOM applications for data visualization: 1) methods that get an idea of the overall data shape and detect possible cluster structures; 2) methods that analyze the prototype vectors (as representatives of the whole dataset) and 3) methods for analysis of new data samples for classification and novelty detection purposes.

In this paper we use visualization methods related to the second and third categories: the U-Matrix and plotting of existing data samples (in our case, query prototypes or templates) over the U-Matrix. The Unified Distance Matrix (U-Matrix) is one of the most used representations of the trained SOM (Gorricha and Lobo, 2012). It is a visual representation of the SOM to reveal cluster structure of the data set. The approach colors a grid according to the distance from each vector prototype and its neighbors: dark colors are chosen to represent large distances while light colors correspond to proximity in the input space and thus represent clusters.

3.4 Data and Implementation

After preprocessing, the initial 1.3 million selected queries were compressed to 8,477 token sets with 2,103 features. As usual in a text mining context, this dataset is extremely sparse, with only 0.008% non-zero values.

Templates were preprocessed in the same manner as the token sets, also using the same idf weights and scaling factors. Since some templates have more than one version, the 45 selected entries expanded to 51, denoted with a suffix letter to indicate when it is a second or third alternative.

Huang (2008) shows that the Euclidean distance performs poorer than other distances in a text clustering context. Hence, for this paper, we chose the Cosine distance as the metric to find BMUs during the SOM training.

For this paper, we used a 30x30 SOM trained for 45 epochs.

3.5 Analysis

We used two plots for an initial visual analysis, the u-matrix, presented in Figure 3, in which numbers indicate the template id over their respective BMU, and a hitmap scatter plot, presented in Figure 4, in which the size of the circles indicates the number of token sets that elected that prototype its BMU.

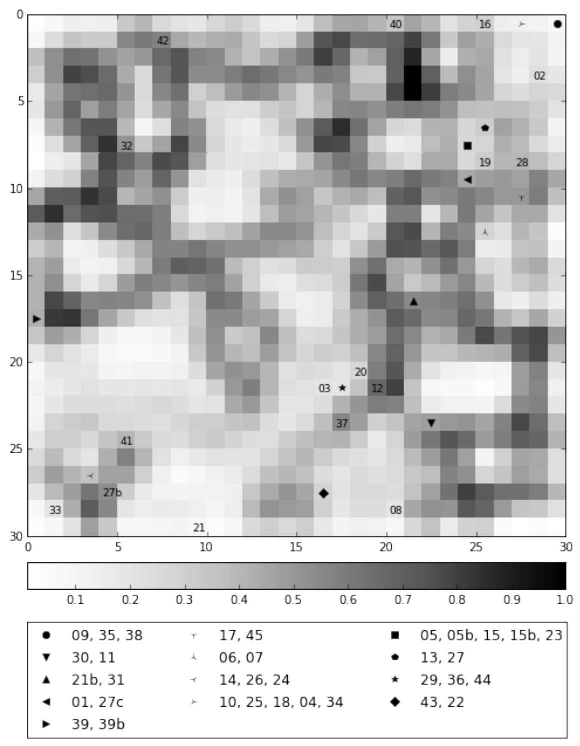

Figure 3: U-Matrix

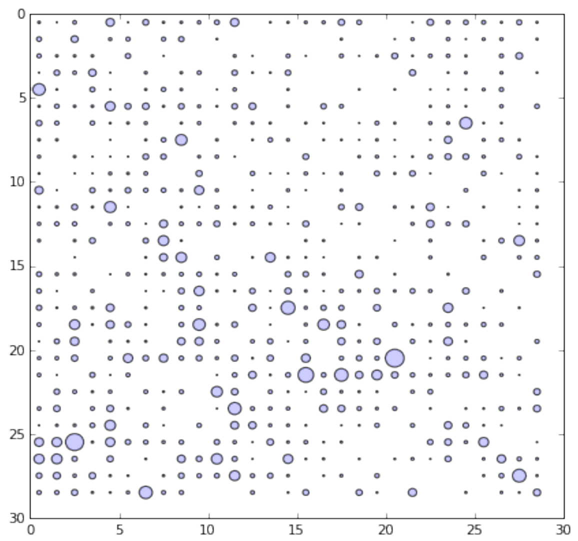

Figure 4: Hitmap

From the figures above, we can see that the trained SOM is able to well distribute the dataset over prototypes and some areas can be visually defined as clusters (regions of light colors circled by dark points).

In some cases, more than one template elected the same prototype as their BMU, as we can check from the legend. So after calculating a distance matrix, we sorted the top 5 closest templates using the Cosine distance, to see how they compare with the trained SOM.

Below, for each pair, we present their Cosine distance using the Term Frequency representation, and the Euclidean distance between their SOM BMUs, along their name.

1. **Pair:** 15 and 15b
   Distances: TF: 0.0 and SOM: 0.0
   15: Splitting 64-bit values into two 32-bit values
   15b: Splitting 64-bit values into two 32-bit values

2. **Pair:** 21b and 31
   Distances: TF: 0.0 and SOM: 0.0
   21b: Finding objects by their spectral lines
   31: Using the sppLines table

3. **Pair:** 22 and 43
   Distances: TF: 0.0205 and SOM: 0.0
   22: Finding spectra by classification (object type)
   43: QSOs by spectroscopy

4. **Pair:** 39 and 39b
   Distances: TF: 0.1610 and SOM: 0.0
   39: Classifications from Galaxy Zoo
   39b: Classifications from Galaxy Zoo

5. **Pair:** 05 and 15
   Distances: TF: 0.1632 and SOM: 0.0
   05: Rectangular position search
   15: Splitting 64-bit values into two 32-bit values

The SQL queries presented that generated the templates listed here are in the Appendix A.

4 Conclusions and Future Work

As a work in progress, further analysis is definitely due, but from this very early results with the SOM, further work is justified by noticing that close pair of queries are being correctly mapped close to one another.

The Self-Organizing Map was selected as a visualization tool due to its quantization and projection properties. Other methods such as clustering could be used, but preliminary tests showed that the selection of algorithms and parameters is not trivial, and the results were not as useful for exploratory data analysis as the SOM and its visual representations.

Next steps include the evaluation of which queries were similar (but not equal) to a specific template, in order to identify queries that were derived from a template; the analysis of clusters of queries that do not have an associated template, which could uncover possible good candidates for new templates: popular queries that can be included in the list presented in the SkyServer as samples; and finally, the processing of the whole log of queries to build a more comprehensive dataset of the historical logs.

This structured representation can also be correlated with other features in the logs, as elapsed time or error results, allowing other applications of KDD, such as the running time or failure prediction.

Acknowledgments

Vitor Hirota Makiyama was supported by a grant from *Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior* (CAPES).

The implementation of the SOM algorithm in this paper was based on the work of Vettigli (2015), licensed under the Creative Commons Attribution 3.0 Unported License.

References

Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. 1996. *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press.

Jorge Gorricha and Victor Lobo. 2012. Improvements on the visualization of clusters in geo-referenced data using Self-Organizing Maps. *Computers & Geosciences*, 43:177–186.

Anna Huang. 2008. Similarity Measures for Text Document Clustering. In *New Zealand Computer Science Research Student Conference*, pages 49–56.

Anil K. Jain, M. Narasimha Murty, and P. Joseph Flynn. 1999. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323.

Teuvo Kohonen. 1998. The self-organizing map. *Neurocomputing*, 21(1):1–6.

Teuvo Kohonen. 2001. *Self-organizing maps*, volume 30. Springer.

Bjornar Larsen and Chinatsu Aone. 1999. Fast and Effective Text Mining Using Linear-Time Document Clustering. In *Proceedings of the 5th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 5, pages 16–22. ACM Press.

Alessandra M. M. Morais, Marcos G. Quiles, and Rafael D. C. Santos. 2014. Icon and Geometric Data Visualization with a Self-Organizing Map Grid. In *Computational Science and Its Applications – ICCSA 2014*, volume 8584 of *Lecture Notes in Computer Science*, pages 562–575. Springer International Publishing.

M. Jordan Raddick, Ani R. Thakar, Alexander S. Szalay, and Rafael D. C. Santos. 2014. Ten Years of SkyServer I: Tracking Web and SQL e-Science Usage. *Computing in Science & Engineering*, 16(4):22–31.

G. Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November.

Ah-Hwee Tan. 1999. Text Mining: The state of the art and the challenges. *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, 8:65–70.

Juha Vesanto. 2002. *Data Exploration Process Based on the Self-Organizing Map*. Ph.D. thesis, Helsinki University of Technology.

G. Vettigli. 2015. MiniSom: minimalistic and NumPy based implementation of the Self Organizing Maps.

Jian Zhang, Chaomei Chen, Michael S. Vogeley, Danny Pan, Ani R. Thakar, and M. Jordan Raddick. 2012. SDSS Log Viewer: visual exploratory analysis of large-volume SQL log data. 8294:82940D.

Appendix A. SkyServer SQL Templates

Sample SQL templates available from SkyServer's help pages that are mentioned in this paper. The list below comprises of the identification number used in the exploratory analysis process, name and category, a brief explanation, and the SQL statement.

05: Rectangular position search (Basic SQL)

Rectangular search using straight coordinate constraints

```
select objid, ra, dec
from photoobj
where (ra between 179.5 and 182.3)
  and (dec between -1.0 and 1.8)
```

15: Splitting 64-bit values into two 32-bit values (SQL Jujitsu)

The flag fields in the SpecObjAll table are 64-bit but some analysis tools only accept 32-bit integers. Here is a way to split them up using bitmasks to extract the higher and lower 32 bits and dividing by a power of 2 to shift bits to the right (since there is no bit shift operator in SQL.)

```
select top 10 objid, ra, dec,
  flags, -- output the whole bigint
      as a check
  flags & 0x00000000ffffffff as
      flags_lo, -- get the lower 32
      bits with a mask shift the
      bigint to the right 32 bits,
      then use the same mask to sget
       upper 32 bits
  (flags/power(cast(2 as bigint),
      32)) & 0x00000000ffffffff as
      flags_hi
from photoobj
```

15B: Splitting 64-bit values into two 32-bit values (SQL Jujitsu)

The hexadecimal version of above query which can be used for debugging

```
select top 10 objid, ra, dec,
  cast(flags as binary(8)) as flags,
  cast(flags & 0x00000000ffffffff as
      binary(8)) as flags_lo,
  cast((flags/power(cast(2 as bigint
      ), 32)) & 0x00000000ffffffff
      as binary(8)) as flags_hi
from photoobj
```

21B: Finding objects by their spectral lines (General Astronomy)

This query selects red stars (spectral type K) with large CaII triplet eq widths with low errors on the CaII triplet equivalent widths.

```
select sl.plate, sl.mjd, sl.fiber,
  sl.caiikside, sl.caiikerr,
  sl.caiikmask, sp.fehadop,
  sp.fehadopunc, sp.fehadopn,
  sp.loggadopn, sp.loggadopunc,
  sp.loggadopn
from spplines as sl
  join sppparams as sp
    on sl.specobjid = sp.specobjid
where fehadop < -3.5
  and fehadopunc between 0.01 and
      0.5
  and fehadopn > 3
```

22: Finding spectra by classification (object type) (General Astronomy)

This sample query find all objects with spectra classified as stars.

```
select top 100 specobjid
from specobj
where class = 'star'
  and zwarning = 0
```

31: Using the sppLines table (Stars)

This sample query selects low metallicity stars ([Fe/H] < -3.5) where more than three different measures of feh are ok and are averaged.

```
select sl.plate, sl.mjd, sl.fiber,
  sl.caiikside, sl.caiikerr,
  sl.caiikmask, sp.fehadop,
  sp.fehadopunc, sp. fehadopn,
  sp.loggadopn, sp.loggadopunc,
  sp.loggadopn
from spplines as sl
  join sppparams as sp
    on sl.specobjid = sp.specobjid
where fehadop < -3.5
  and fehadopunc between 0.01 and
      0.5
  and fehadopn > 3
```

39: Classifications from Galaxy Zoo (Galaxies)

Find the weighted probability that a given galaxy has each of the six morphological classifications.

```
select objid, nvote,
  p_el as elliptical,
  p_cw as spiralclock,
  p_acw as spiralanticlock,
  p_edge as edgeon,
  p_dk as dontknow,
  p_mg as merger
from zoonospec
where objid = 1237656495650570395
```

39B: Classifications from Galaxy Zoo (Galaxies)

Find 100 galaxies that have clean photometry at least 10 Galaxy Zoo volunteer votes and at least an 80% probability of being clockwise spirals.

```
select top 100 g.objid, zns.nvote,
  zns.p_el as elliptical,
  zns.p_cw as spiralclock,
  zns.p_acw as spiralanticlock,
  zns.p_edge as edgeon,
  zns.p_dk as dontknow,
  zns.p_mg as merger
from galaxy as g
  join zoonospec as zns
    on g.objid = zns.objid
where g.clean=1
  and zns.nvote >= 10
  and zns.p_cw > 0.8
```

43: QSOs by spectroscopy (Quasars)

The easiest way to find quasars is by finding objects whose spectra have been classified as quasars. This sample query searches the SpecObj table for the IDs and redshifts of objects with the class column equal to 'QSO'

```
select top 100 specobjid, z
from specobj
where class = 'qso'
  and zwarning = 0
```