



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2015/08.31.17.43-TDI

TEXT MINING APPLIED TO SQL QUERIES: A CASE STUDY FOR SDSS SKYSERVER

Vitor Hirota Makiyama

Master Thesis for the Graduate Program in Applied Computing, advised by Dr. Rafael Duarte Coelho dos Santos, approved in September 21, 2015.

URL of the original document:

<<http://urlib.net/8JMKD3MGP3W34P/3K6JNQ8>>

INPE
São José dos Campos
2015

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE

Gabinete do Diretor (GB)

Serviço de Informação e Documentação (SID)

Caixa Postal 515 - CEP 12.245-970

São José dos Campos - SP - Brasil

Tel.:(012) 3208-6923/6921

Fax: (012) 3208-6919

E-mail: pubtc@sid.inpe.br

**COMMISSION OF BOARD OF PUBLISHING AND PRESERVATION
OF INPE INTELLECTUAL PRODUCTION (DE/DIR-544):****Chairperson:**

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Members:

Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)

Dr. Amauri Silva Montes - Coordenação Engenharia e Tecnologia Espaciais (ETE)

Dr. André de Castro Milone - Coordenação Ciências Espaciais e Atmosféricas
(CEA)

Dr. Joaquim José Barroso de Castro - Centro de Tecnologias Espaciais (CTE)

Dr. Manoel Alonso Gan - Centro de Previsão de Tempo e Estudos Climáticos
(CPT)

Dr^a Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação

Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)

DIGITAL LIBRARY:

Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)

Clayton Martins Pereira - Serviço de Informação e Documentação (SID)

DOCUMENT REVIEW:

Simone Angélica Del Ducca Barbedo - Serviço de Informação e Documentação
(SID)

Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

ELECTRONIC EDITING:

Marcelo de Castro Pazos - Serviço de Informação e Documentação (SID)

André Luis Dias Fernandes - Serviço de Informação e Documentação (SID)



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2015/08.31.17.43-TDI

TEXT MINING APPLIED TO SQL QUERIES: A CASE STUDY FOR SDSS SKYSERVER

Vitor Hirota Makiyama

Master Thesis for the Graduate Program in Applied Computing, advised by Dr. Rafael Duarte Coelho dos Santos, approved in September 21, 2015.

URL of the original document:

<<http://urlib.net/8JMKD3MGP3W34P/3K6JNQ8>>

INPE
São José dos Campos
2015

Cataloging in Publication Data

Makiyama, Vitor Hirota.

M289t Text mining applied to SQL queries: a case study for SDSS
skyserver / Vitor Hirota Makiyama. – São José dos Campos :
INPE, 2015.

xx + 55 p. ; (sid.inpe.br/mtc-m21b/2015/08.31.17.43-TDI)

Dissertation (Master in Applied Computing) – Instituto Na-
cional de Pesquisas Espaciais, São José dos Campos, 2015.

Guiding : Dr. Rafael Duarte Coelho dos Santos.

1. Text mining. 2. SQL. 3. KDD. 4. SDSS. I.Title.

CDU 004.4

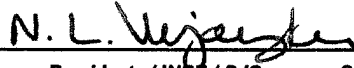


Esta obra foi licenciada sob uma Licença [Creative Commons Atribuição-NãoComercial 3.0 Não Adaptada](https://creativecommons.org/licenses/by-nc/3.0/).

This work is licensed under a [Creative Commons Attribution-NonCommercial 3.0 Unported License](https://creativecommons.org/licenses/by-nc/3.0/).

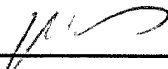
Aprovado (a) pela Banca Examinadora
em cumprimento ao requisito exigido para
obtenção do Título de **Mestre** em
Computação Aplicada

Dr. Nandamudi Lankalapalli Vijaykumar



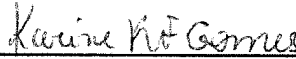
Presidente / INPE / SJC Campos - SP

Dr. Rafael Duarte Coelho dos Santos



Orientador(a) / INPE / SJC Campos - SP

Dra. Karine Reis Ferreira Gomes



Membro da Banca / INPE / São José dos Campos - SP

Dr. Gilberto Ribeiro de Queiroz



Membro da Banca / INPE / São José dos Campos - SP

Dra. Daniela Leal Musa



Convidado(a) / UNIFESP / São José dos Campos - SP

Este trabalho foi aprovado por:

maioria simples

unanimidade

Título: "Text Mining applied to SQL Queries: a case study for SDSS skyserver".

Aluno (a): **Vitor Hirota Makiyama**

São José dos Campos, 21 de setembro de 2015

To my parents, hoping to always make you proud.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Rafael Duarte Coelho dos Santos, for the confidence granted in terms of freedom and flexibility to pursue my own ideas and interests, always available to discuss anything further and help make whatever happen; and for all the amazing opportunities I was able to take advantage of. My deepest gratitude for all the time and effort given.

I can not thank my wife, Kareninne Carvalho, enough, for the ever lasting patience and support, putting up with the sometimes crazy schedule, periods of absence and hours of science, math and space blab to which she would never ask to stop, even though she probably should.

The program would not be the same without the friendly help of fellow colleagues, specially Alessandra Moraes, José Renato, Marcio Azeredo, Marluce Scarabello, and Wanderson Costa, with whom I have had a fair share of moments of despair throughout the program's obligations. I thank you all for the hours of shared hard working and partnership during these years, along with plenty of joy and laughter. A friendship I will hold dear for the rest of my life.

I would also like to thank Fabiana and Cornelis, long lasting friends, for opening up their home and welcoming me during the first periods of study in São José dos Campos, which greatly eased my transition into the program.

Lastly, but not least, my appreciation to INPE, for the study opportunity provided and CAPES, for the financial support.

ABSTRACT

SkyServer, the Internet portal for the Sloan Digital Sky Survey (SDSS) catalog, provides a set of tools that allows data access for astronomers and scientific education. One of the available interfaces allows users to enter *ad-hoc* SQL statements to query the catalog, and has logged over 280 million queries since 2001. To assess and investigate usage behavior, log analyses were performed after the 5th and 10th year of the portal being in production. Such analyses, however, focused on the HTTP access, and just simple information for the database usage. This work aims to apply text mining techniques over the SQL logs to define a methodology to parse, clean and tokenize statements into an intermediate numerical representation for data mining and knowledge discovery, which can provide deeper analysis over SQL usage, and also has a number of foreseen applications in database optimization and improving user experience.

MINERAÇÃO DE TEXTO APLICADO À CONSULTAS SQL: UM ESTUDO DE CASO PARA O SDSS SKYSERVER

RESUMO

SkyServer, o portal de Internet para o catálogo *Sloan Digital Sky Survey* (SDSS), fornece um conjunto de ferramentas que permitem acesso a dados para astrônomos e para educação científica. Uma das interfaces disponíveis permite a inserção de instruções SQL *ad-hoc* para consultar o catálogo, e já recebeu mais de 280 milhões de consultas desde 2001. Para avaliar e investigar o comportamento de uso, análises de log foram realizadas após o 5º e 10º ano de vida do portal. Tais análises, no entanto, focaram no acesso HTTP, e apenas informações básicas de utilização do banco de dados. Este trabalho tem por objetivo aplicar técnicas de mineração de texto sobre os logs SQL com o intuito de definir uma metodologia para analisar, limpar e dividir em símbolos tais declarações em uma representação numérica intermediária para posterior mineração de dados e extração de conhecimento; possibilitando análises mais profundas sobre o uso de SQL, e também aplicações previstas em otimização de banco de dados e para melhora de experiência de usuário.

LIST OF FIGURES

	<u>Page</u>
2.1 An overview of the KDD process steps.	5
2.2 Frequency distribution of the top 5000 SQL terms from the SDSS Sky- Server SQL logs.	8
2.3 Example of a clustering analysis shown as the color labeling of input patterns into three clusters.	10
3.1 The methodology flowchart.	21
3.2 Example of a SQL query and its normalized version. Whitespace is in- cluded for readability.	24
3.3 Feature vector.	24
3.4 Example of a token set and statements that generated it.	25
4.1 FCM training metrics for different values of c	32
4.2 FCM cluster validity measures for different values of c	33
4.3 U-Matrix	34
4.4 Hitmap	35

LIST OF ABBREVIATIONS

BMU	–	Best Matching Unit
FCM	–	Fuzzy C-Means
HTML	–	Hypertext Markup Language
INPE	–	Brazilian National Institute for Space Research
IP	–	Internet Protocol
IR	–	Information retrieval
KDD	–	Knowledge Discovery in Databases
SDSS	–	Sloan Digital Sky Survey
SOM	–	Self-Organizing Maps
SQL	–	Structured Query Language
TF-IDF	–	Term Frequency * Inverse Document Frequency
UCSC	–	University of California, Santa Cruz
XML	–	Extensible Markup Language

LIST OF SYMBOLS

$d(x, y)$	–	Distance measure between points x and y
df_t	–	Document frequency of term t
idf_t	–	Inverse Document Frequency of term t
$\text{sim}(X, Y)$	–	Similarity measure between sets X and Y
$tf_{t,d}$	–	Term frequency of term t in document d

CONTENTS

	<u>Page</u>
1 INTRODUCTION	1
1.1 Context and Motivation	1
1.2 Related Work	2
1.3 Thesis Overview	3
2 TEXT MINING	5
2.1 Introduction	5
2.2 Information Retrieval	7
2.2.1 Vocabulary Construction	7
2.2.2 Term Distribution and Weighting	8
2.3 Clustering	9
2.3.1 Measures of Association	10
2.3.1.1 Euclidean metrics	11
2.3.1.2 Cosine coefficient	11
2.3.1.3 Jaccard coefficient	12
2.3.1.4 Discussion on the different measures	13
2.3.2 Methods and Algorithms	13
2.3.3 K-Means	13
2.3.4 Fuzzy C-Means	14
2.3.5 Cluster validity	16
2.3.6 The Curse of Dimensionality	17
2.3.7 Self-Organizing Maps	17
3 METHODOLOGY	21
3.1 Selection	21
3.2 Preprocessing	22
3.3 Transformation	25
3.4 Data Mining	28
4 EXPERIMENTAL RESULTS	31
4.1 On data and implementation	31
4.2 Analysis of number of clusters with FCM	32
4.3 Visual analysis of the correlation between queries and templates	33

5 CONCLUSIONS	37
REFERENCES	39
APPENDIX A - PARSER	45
APPENDIX B - TEMPLATES	53

1 INTRODUCTION

1.1 Context and Motivation

Long before the big data hype, astronomy projects had to deal with large amounts of data being collected and generated. One such project is the Sloan Digital Sky Survey (SDSS), the most influential astronomy survey to date (MADRID; MACCHETTO, 2009). In operation since April 2000, the program is in its fourth iteration (SDSS-I, 2000-2005; SDSS-II, 2005-2008; SDSS-III, 2008-2014; SDSS-IV, 2014-2020) and has created a detailed three-dimensional map of the Universe, with images of over one third of the sky, and spectra for more than five million astronomical objects (ALAM et al., 2015).

Raw data collected by SDSS is processed for reduction, correction, calibration, and feature extraction; which is then stored in an indexed database and eventually made public (STOUGHTON et al., 2002; SZALAY et al., 2002). The Catalog Archive Server, one of SDSS's data distribution interfaces, was originally designed as an object-oriented database, but during the first public data release faced too many bugs and issues with performance and scalability as data increased. At the time, an alternative easy-to-use, web-based version was also deployed using a relational database that became known as SkyServer. Geared towards casual users with visualization tools and educational resources, it also included an *ad-hoc* SQL query submission page, which for general surprise proved to be far more popular and reliable to get data out of the database, even with professional astronomers. This fact eventually led to the original design being deprecated in favor of the alternative (THAKAR et al., 2003).

For astronomers to answer queries like 'find gravitational lens candidates' or 'find objects like this one', they would have to download a subset of the binary data and write their own programs to analyze such data, taking hours or days in the process. The SQL-based SkyServer, however, allowed such queries to be quickly processed through a simple SQL statement. The portal was built to serve as a data mining tool, meaning users could simply and quickly query and analyze only the most relevant and up-to-date data for their needs, without the need for any downloads or custom development, representing a real productivity gain in their workflow (SZALAY et al., 2002). In operation since 2001, SkyServer has proven to be extremely popular, with an average of over 19 million page hits and almost 2 million SQL queries submitted every month (SDSS, 2015).

Since 2003, SkyServer has been logging every query submitted to the portal. Other than the statement itself, it also collects other query information, such as timestamp, target data release, origin (IP address and the tool used), query success or failure, elapsed time, among others. 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. \(2014\)](#).

However, for a more in depth usage analysis, more complex approaches are required, such as data processing and transformation. Thus, this work aims to apply text mining techniques with the goal to define a methodology to parse, clean and tokenize statements into a weighted numerical representation, which can then be fed into regular machine learning algorithms for data mining. As proof-of-concept, we proceed with an exploratory analysis over part of the historical logs to uncover natural groupings through clustering techniques.

1.2 Related Work

There are other works which also analyzed the historical SQL logs from SkyServer. [Singh et al. \(2006\)](#) suggests that SQL queries with incorrect syntax can be compared to the logs, so to recommend similar and correct ones back to the user. [Zhang et al. \(2012\)](#) presents a visualization tool for the logs, color coding queries to easily compare different length statements and plotting a sky map of popular searched areas.

This thesis, in turn, specializes the parsing statements from the former and open up analysis and mining opportunities from the latter by allowing the use of regular machine learning algorithms.

SQL is also used in other scientific projects, such as the UCSC Genome Browser ([KENT et al., 2002](#)), which features a web tool to build queries and direct access to its database; and SQLShare ([HOWE et al., 2011](#)), a cloud-based tool that allows scientists to update their data in plain files or spreadsheets and promptly analyze them using SQL.

Hence, we expect lessons learned in this context could also be applied in any other scientific database publicly available through SQL interfaces.

1.3 Thesis Overview

This thesis is organized as follows. In [Chapter 2](#) we review the field of text mining and related disciplines, which brings together the set of techniques used in exploring and analyzing the data. The methodology, explaining the steps taken towards our objective, is presented in [Chapter 3](#), with discussions of experimental results in [Chapter 4](#). Finally, [Chapter 5](#) presents the conclusions and future directions.

2 TEXT MINING

2.1 Introduction

Knowledge Discovery in Databases (KDD) is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data (FAYYAD et al., 1996). Such process, with its underlying activities, is presented in Figure 2.1.

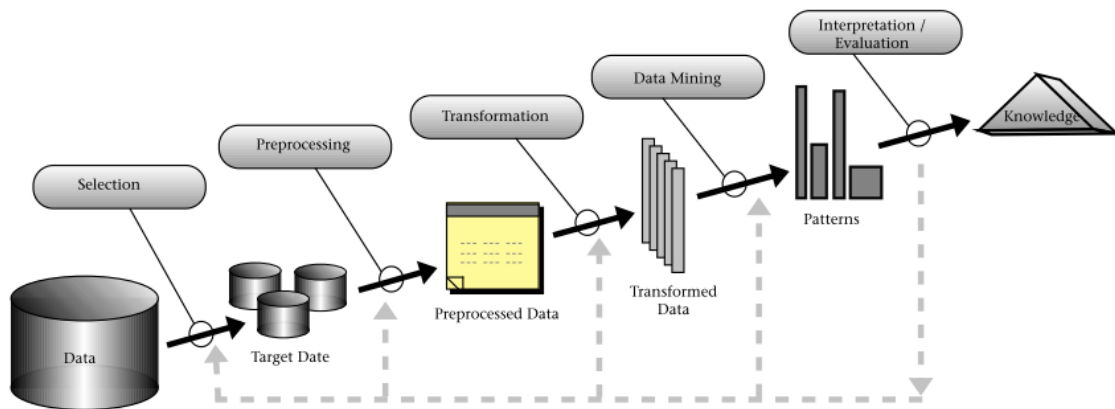


Figure 2.1 - An overview of the KDD process steps.

SOURCE: Fayyad et al. (1996)

Text mining, also known as Text Data Mining or Knowledge Discovery in Texts, can be viewed as an extension to KDD, in which it pursues the same objective and can be applied through the same process, but with specific techniques to deal with the different type of data it targets: unstructured or semi-structured textual data, such as emails, full-text documents and markup files (e.g., HTML and XML) (TAN, 1999; FAN et al., 2006).

KDD is the intersection of a number of research fields, including machine learning, pattern recognition, databases, statistics, artificial intelligence, data visualization, and high-performance computing (FAYYAD et al., 1996). On top of these, text mining also draws on advances from other computer science disciplines concerned with the handling of text and natural language, such as information retrieval, information extraction and natural language processing (TAN, 1999; FELDMAN; SANGER, 2006).

The discovery process, as depicted in [Figure 2.1](#), is interactive and iterative, involving many decisions made by the user, and can have significant iteration, sometimes containing loops between any two steps. After developing an understanding of the application domain and identifying a goal, [Fayyad et al. \(1996\)](#) broadly outline the process to involve *selection*, *preprocessing* and *transformation* of the data to be processed in order to create a target dataset, with noise removed from it, accounted for missing values and properly reduced for the most useful features to represent such data; application of *data mining* algorithms to extract patterns or models; and *evaluating* the results to identify the subset of the enumerated patterns deemed knowledge.

As surveyed by [Fan et al. \(2006\)](#), technologies of text mining include:

Information extraction. Refers to the ability of computers to analyze unstructured text and identify key phrases and relationships within text, by the process of pattern matching. Serves as the basis for many of the various other text mining technologies.

Topic tracking. The inference and prediction of other documents of interest for a given user, based on his access and reading history.

Summarization. To reduce the length and detail of a document to its main points and overall meaning, helping users assess whether a document meets their needs.

Categorization. Refers to the identification of the main themes of a document and assigning a predefined topic.

Clustering. Refers to the grouping of similar documents. The main difference with categorization is that labels are not predefined.

Concept linkage. The ability to connect related documents by identifying their shared concepts, sometimes helping users find information they perhaps would not have found through traditional search.

Information visualization. To provide large textual sources in a visual hierarchy or map. Like concept linkage, it often provides browsing capabilities, in addition to search.

Question answering. Refers to the processing of queries in a natural language form.

We discuss below the supporting techniques in the related fields of information retrieval and machine learning that are of particular interest for this work. By considering SQL statements as short documents, we can use such techniques to perform a number of exploratory analyses over the historical logs of SkyServer, considered here as our document collection.

2.2 Information Retrieval

Information Retrieval (IR) is the field of study interested in finding text documents that satisfy an information need from within large collections. Much of its concepts and technologies govern the basics of how search engines work, such as indexes construction and compression, term vocabulary and spelling correction, boolean and tolerant retrieval, scoring and relevance, among others (MANNING et al., 2009).

In the context of the Text Mining process, as illustrated in Figure 2.1, IR techniques and concepts can be applied throughout the process, specially in the preprocessing, transformation and evaluation steps. Some of which are explained below.

2.2.1 Vocabulary Construction

In Manning et al. (2009), some key definitions are made as follows: *token* is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing; *type* is the class of all tokens containing the same character sequence; *term*, or word, is a type that is included in the vocabulary; and *vocabulary*, also referred to as dictionary or lexicon, is the set of terms.

Vocabulary construction could be as simple as splitting white space in text. This process is known as *tokenization*: the task of chopping a given character sequence, usually throwing away certain characters in the process, such as punctuation. This, however, could lead to duplicate types that just have different letter cases, e.g., “Select” and “select”. Therefore, it is also common to run other preprocessing tasks during vocabulary construction, such as *token normalization*, the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens; dropping common words, known as *stop words*; or *stemming*, the process to reduce inflectional and derivationally related forms of a word to a common base form. (MANNING et al., 2009).

2.2.2 Term Distribution and Weighting

Zipf's Law, a commonly used model of the distribution of terms in a collection of documents, states that the product of the frequency of use of words and the rank order is approximately constant. Let cf_i be the collection frequency of the i th most common term ordered by number of appearances, Zipf's observation was that $cf_i \propto 1/i$ (RIJSBERGEN, 1979; MANNING et al., 2009). It is a power law that, when plotted on a log-log scale, renders a straight line, such as the one depicted in Figure 2.2.

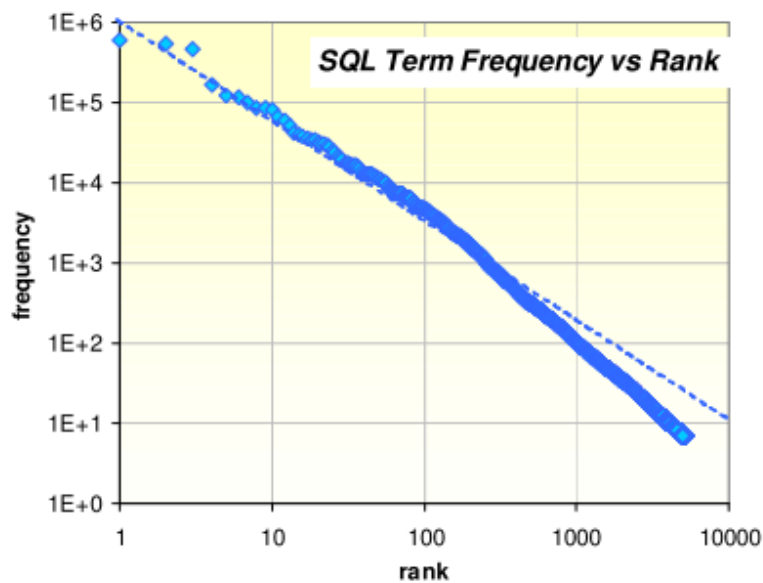


Figure 2.2 - Frequency distribution of the top 5000 SQL terms from the SDSS SkyServer SQL logs. The dashed line shows a -1 slope corresponding to Zipf's Law.

SOURCE: Singh et al. (2006)

Luhn (1958) states that a set of significant words could be established by their rank order based on term frequency, and, thus, this set could be used to discriminate the contents of a document. Use of term frequency is one of the simplest approaches to give a weight to a term, denoted as $tf_{t,d}$, with the subscripts denoting the term and the document in order. This particular representation of a document is known as the *bag of words* model, in which the order of appearance of a given term is irrelevant, but the number of its occurrences are material (MANNING et al., 2009).

However, not all terms have the same significance towards a document’s representation, as also devised by Luhn (1958), where a statistical approach could be used to define “confidence limits” to remove terms that are too common or too rare, leaving only those that have the most resolving power of significance.

An extremely popular approach on this matter was proposed by Jones (1972), and consists in scaling down the weights of terms with high document frequency, df_t , defined to be the number of documents in the collection that contain a term t . With N as the total number of documents in a collection, the scaling factor became known as the inverse document frequency, denoted idf_t :

$$idf_t = \log \frac{N}{df_t}$$

Combining the definitions of term frequency and inverse document frequency gives the *tf-idf* weighting scheme that assigns the largest weight to those terms which 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). Formally, for a term t , a weight in document d is given by:

$$tf-idf_{t,d} = tf_{t,d} \times idf_t$$

In this case, documents are represented as a vector of its terms weights, known as the *vector space* model. In this model, a collection of vectors is denoted as a *term-document matrix*, an $M \times N$ matrix, whose rows represent the M terms of the N documents (MANNING et al., 2009). Note that in the context of IR M is usually large, but also sparse, i.e., there is a large number of terms, but documents do not have all of them.

2.3 Clustering

As introduced before, in a text mining context, clustering refers to the grouping of similar documents, and can be used for example, to improve search performance by narrowing the search space, to organize results by topic similarity and thus helping exploration of relevant groups within the collection, or yet to summarize contents of a given collection (LARSEN; AONE, 1999).

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 asso-

ciation measure. 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 (JAIN et al., 1999). An example is given in Figure 2.3, where it is visually clear the presence of three different clusters based on the density of the groups, i.e., points within a cluster are closer to each other than to any other point in this two-dimensional Euclidean plane.

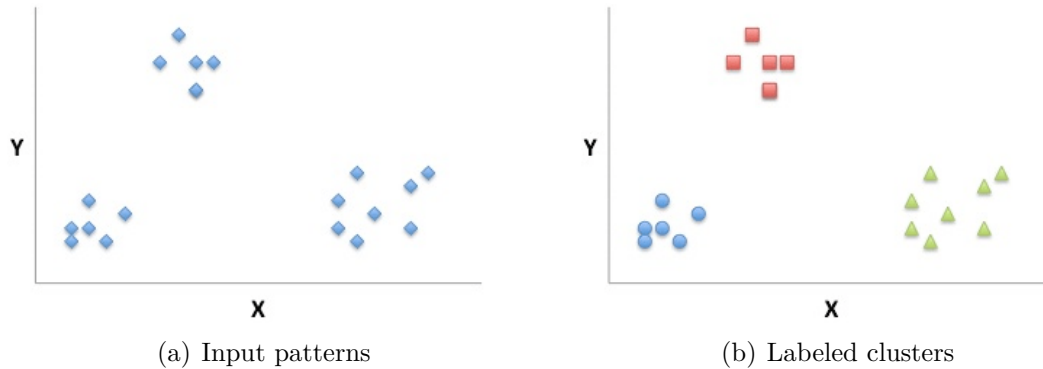


Figure 2.3 - Example of a clustering analysis shown as the color labeling of input patterns into three clusters.

Also referred to as unsupervised classification, clustering fundamentally differs from discriminant analysis, or supervised classification, because there are no prior labels in the data that define what the clusters should be (JAIN et al., 1999).

2.3.1 Measures of Association

Many of the clustering methods are based on a binary relationship between patterns, with association measures quantifying in a numerical measure how similar or dissimilar two patterns are between each other. If one considers patterns as objects, such association could be the number of attributes they share, or considering patterns as points in an Euclidean space, this relation could be described as how close or distant they lie.

Recall from subsection 2.2.2 that documents can be either represented as a bag of words, or vectors. For the first case, if we consider just the set of terms, it is intuitive that two documents with similar bags are similar in content. Formally, given sets X and Y , the similarity measure is a function $\text{SIM}(X, Y)$ that increases as the number of shared terms increases. The simplest measure $|X \cap Y|$, known as the *simple matching coefficient*, is the number of terms that are both in X and Y . For

the case of a vector representation, lets consider the case of a boolean vector, with n components (terms of the vocabulary) with 0s or 1s denoting absence or presence of a term. Given vectors x and y , it is easy to devise that the simple matching coefficient can be written as the sum of components in which both vectors are 1, i.e., their inner product: $\sum_{i=1}^n x_i y_i$ (RIJSBERGEN, 1979; MANNING et al., 2009).

Distances, or dissimilarity measures, can be defined as follows. Given a set of points, called a space, a distance measure is a function $d(x, y)$ that takes two points in the space and produces a real number. It must also satisfy the following axioms, in which case, it is also called a metric (RAJARAMAN; ULLMAN, 2011):

- i. $d(x, y) \geq 0$,
- ii. $d(x, y) = 0$ if and only if $x = y$,
- iii. $d(x, y) = d(y, x)$, and
- iv. $d(x, y) \leq d(x, z) + d(z, y)$, known as the *triangle inequality*.

2.3.1.1 Euclidean metrics

The most familiar distance measure for continuous features is the *Euclidean* distance

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} = \|x - y\|_2$$

Also known as the L_2 -norm, it is just a especial case ($r = 2$) of the L_r -norm, or Minkowsky distance

$$d(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^r \right)^{1/r} = \|x - y\|_r$$

There are two other common cases for the L_r -norm. The L_1 -norm, or *Manhattan distance*, which is just the sum of the absolute differences in each dimension, and the L_∞ -norm, which is the limit as r approaches infinity. Formally, the L_∞ -norm is defined as $\max(|x_i - y_i|)$ over all dimensions i , because as r gets larger, only the dimension with the largest difference matters (RAJARAMAN; ULLMAN, 2011).

2.3.1.2 Cosine coefficient

The cosine coefficient is the angular separation of the vectors that two points make. It is defined by the inner product of these vectors, divided by the product of their

magnitudes (i.e., their L_2 -norms or Euclidean lengths) (RIJSBERGEN, 1979). Given two vectors x and y , the cosine similarity is given by

$$\text{SIM}_C(x, y) = \frac{x \cdot y}{|x||y|} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2 \sum_{i=1}^n y_i^2}}$$

It considers vector directions, and, as such, a vector and its multiples are considered the same. Thus, the cosine coefficient is vector-length invariant, which is specially useful in cases that two documents with similar content, but different lengths can have a significant vector difference considering their Euclidean distance (MANNING et al., 2009). Another interesting property is that it can also be applied to discrete versions of Euclidean spaces, where points are vectors with integer or boolean (0 or 1) components (RAJARAMAN; ULLMAN, 2011).

2.3.1.3 Jaccard coefficient

The Jaccard coefficient is a measure of overlap between sets. Given two sets X and Y , the Jaccard similarity is given by

$$\text{SIM}_J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|},$$

with 0 when there is no overlap and hence total dissimilarity, and 1 when $X = Y$, meaning total similarity. This coefficient also has a heuristic interpretation, in which it measures the probability that an element of at least one of two sets is an element of both (LEVANDOWSKY; WINTER, 1971). Also note that $1 - \text{SIM}_J$, known as the Jaccard distance, is a proper distance metric, abiding to all four axioms defined before (RIJSBERGEN, 1979).

As with the simple matching coefficient, the Jaccard coefficient can be generalized to bit vectors and then further for continuous or discrete non-negative spaces, known as the Extended Jaccard coefficient. Given two vectors x and y , Extended Jaccard similarity is given by

$$\text{SIM}_{EJ}(x, y) = \frac{x \cdot y}{|x|^2 + |y|^2 - x \cdot y}.$$

This version has the morphing property to behave like the Euclidean dis-

tance for smaller vectors, and like the Cosine coefficient for larger vectors (STREHL et al., 2000).

2.3.1.4 Discussion on the different measures

Rijsbergen (1979) states that, although there is a number of different coefficients, the difference in retrieval performance achieved by them are insignificant, provided they are appropriately normalized. As such, Jaccard and Cosine coefficients can be seen as a normalized version of the simple matching coefficient, by considering the sizes of the argument vectors. As expected, their performance is similar, as reviewed in Strehl et al. (2000) and Haveliwala et al. (2002), and also preferred over Euclidean distances for showing better results, as shown by Strehl et al. (2000) and Huang (2008). In regards to Euclidean metrics, Gionis et al. (1999) states that there is no clear difference between using L_1 or L_2 norms.

2.3.2 Methods and Algorithms

There is a large number of different clustering methods and algorithms in the literature, each with different processes and results. Tan et al. (2005) resumes these differences in two categories, types of clusterings and types of clusters.

Clusterings can be *hierarchical or partitional*, in which the former produces a nested structure of clusters, while the latter results in a flat set; *exclusive, overlapping or fuzzy*, in which patterns belong to one, more than one, or to all (with different degrees of membership between 0 and 1) clusters, respectively; and *complete or partial*, which defines if all patterns have been assigned to a cluster or not.

Clusters, among others types, can be *well-separated*, in which patterns are closer to each other in the cluster than to anyone of a different cluster; *prototype-based*, or *centroid-based*, in which each pattern is closer to the prototype that defines the cluster than to any other prototype; or *density-based*, in which a cluster is a dense region of patterns, surrounded by a region of low density.

Below we discuss two methods popularly applied in text mining contexts.

2.3.3 K-Means

K-Means, also denoted as (hard) c-means (CHI et al., 1996), is one of the most popular clustering algorithms. It is a partitional, exclusive and complete approach, based on minimizing the squared error criterion. Let C be the patterns set that are part of

a cluster, K the number of clusters, and V the set of cluster centers (the *centroids*), the squared error function is given as (JAIN et al., 1999; MANNING et al., 2009).

$$J(V) = \sum_{k=1}^K \sum_{x \in C_k} \|x - c_k\|^2 ,$$

where $c_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$ is the centroid of cluster k , calculated as the mean of all the patterns member of that cluster.

Starting with K random initial partitions, it iteratively reassign the patterns to centroids, until convergence, i.e., no reassignments of patterns were made in that iteration or the squared error ceases to decrease significantly (JAIN et al., 1999). Its popularity is due to its implementation simplicity and linear complexity in time ($O(IKMN)$), with I iterations, K clusters, M vector dimensions and N patterns) (MANNING et al., 2009).

The general algorithm goes as follows:

- i. Choose k cluster centers.
- ii. Assign each pattern to the closest cluster center.
- iii. Recompute cluster center using the current cluster memberships.
- iv. If convergence criterion is not met, go to step ii.

Drawbacks, as listed in Berkhin (2006), include, but are not limited to, results strongly depending on the initial guess of centroids, K not easily defined, sensitivity to outliers, not scalable and only applicable for Euclidean spaces. However, given its widespread usage and popularity, a number of extensions and modifications have been proposed, as reviewed by Jain et al. (1999), Berkhin (2006), Manning et al. (2009) and Rajaraman and Ullman (2011), in regards to better centroids initialization or choosing the right value of K , among others.

2.3.4 Fuzzy C-Means

Fuzzy C-Means (FCM), is one such extension of the k-means, and targets cases in which clusters are not completely disjointed, therefore data could be classified as belonging to one cluster almost as well as to another. Here, the difference is that

each pattern belongs to all clusters with varying degree of membership between 0 and 1. The criterion function is updated as following (CHI et al., 1996):

$$J(U, V) = \sum_{k=1}^K \sum_{n=1}^N u_{kn}^m \|x_n - c_k\|^2$$

where:

- x_1, \dots, x_n are data sample vectors;
- $V = c_1, \dots, c_k$ are cluster centroids, calculated as

$$c_k = \frac{1}{\sum_{n=1}^N u_{kn}^m} \sum_{n=1}^N u_{kn}^m x_{kn} ;$$

- $U = [u_{kn}]$ is a $K \times N$ matrix, where u_{kn} is the k^{th} membership value of the n^{th} input sample x_n , calculated as

$$u_{kn} = \frac{\left[\frac{1}{\|x_n - c_k\|^2} \right]^{1/(m-1)}}{\sum_{j=1}^K \left[\frac{1}{\|x_n - c_j\|^2} \right]^{1/(m-1)}} ,$$

and the membership values satisfy the following conditions $0 \leq u_{kn} \leq 1$, $\sum_{k=1}^K u_{kn} = 1$, $0 < \sum_{n=1}^N u_{kn} < n$;

- $m \in [1, \infty)$ is an exponent weight factor;

Note how cluster centroids now consider every pattern, and contributions of samples are weighed by its membership value, which is defined according to its distance to the corresponding centroid. The weight factor m reduces the influence of small membership values. The larger the value of m , the smaller the influence of samples with small membership values (CHI et al., 1996).

The FCM algorithm goes as following:

- Choose the value of m , k cluster centers, and calculate $U^{(0)}$. Set the iteration $\alpha = 1$.
- Compute cluster centers. Given $U^{(\alpha)}$, calculate $V^{(\alpha)}$.

iii. Update membership values. Given $V^{(\alpha)}$, calculate $U^{(\alpha)}$.

iv. Stop the iteration if

$$\max |u_{kn}^{(\alpha)} - u_{kn}^{(\alpha-1)}| \leq \varepsilon$$

else let $\alpha = \alpha + 1$ and go to step ii, where ε is the pre-specified small number representing the smallest acceptable change in U .

2.3.5 Cluster validity

Since clustering is an unsupervised learning process, there is no information on labels for the data, as opposed to supervised learning, in which results can be compared to the correct label of a given pattern. Clustering results can then be assessed through an expert, or by a particular automated procedure and relates to two issues: i) interpretability, and ii) visualization (BERKHIN, 2006).

Assessment process depends on a number of factors, such as the method of initialization, the choice of the number of classes, and the clustering method. FCM provides more flexibility than its hard counterpart, K-Means. Thus, we shall consider validity for FCM only, and specifically how to choose the number of clusters c , since initialization requires a good estimate of the clusters and is application dependent.

Below we describe four of these validity measures: partition coefficient, partition entropy, Fukuyama-Sugeno, and Xie-Beni (CHI et al., 1996; PAL; BEZDEK, 1995).

The *partition coefficient* v_{PC} and *partition entropy* v_{PE} both measure the “fuziness” of the clustering result. The former by measuring the closeness of all input samples to their corresponding centroids, and the latter by measuring the distance the matrix U is from being crisp. They are given by

$$v_{PC}(U) = \frac{1}{N} \sum_{k=1}^c \sum_{n=1}^N (u_{kn})^2$$

and

$$v_{PE}(U) = -\frac{1}{N} \sum_{k=1}^c \sum_{n=1}^N u_{kn} \log(u_{kn}) .$$

If each sample is closely associated with only one cluster, i.e., for each n , u_{kn} is large for only one k value, then the uncertainty of the data is small, which corresponds to a large $v_{PC}(U)$ value. And if all u_{kn} ’s are close to 0 or 1, $v_{PE}(U)$ is small and indicates a good clustering result.

The *Fukuyama-Sugeno* v_{FS} index consists of the difference of two terms. The first term combines the fuzziness in U with the geometrical compactness of the representation of X via the c prototypes V . The second term combines the fuzziness in each row of U with the distance from the k^{th} prototype to the grand mean of the data. The index is defined as

$$v_{\text{FS}}(U, V, X) = \sum_{k=1}^c \sum_{n=1}^N (u_{kn})^m (||x_n - v_k||^2 - ||v_k - \bar{v}||^2) ,$$

where $1 < m < \infty$.

The *Xie-Beni* index is the ratio of the total variation of the partition and the centroids (U, V) and the separation of the centroids vectors, and is given as

$$v_{\text{XB}}(U, V, X) = \frac{\sum_{k=1}^c \sum_{n=1}^N u_{kn}^m ||x_n - v_k||^2}{N(\min_{k \neq l} \{||v_k - v_l||^2\})}$$

Both Fukuyama-Sugeno and Xie-Beni indexes propose good partitions for their minimum values over the number of c 's.

2.3.6 The Curse of Dimensionality

When working with high-dimensional spaces, such as documents, a problem known as the “curse of dimensionality” arises, in which almost all pairs of points are equally far away from one another and almost any two vectors are almost orthogonal (RAJARAMAN; ULLMAN, 2011). One approach to deal with this problem is to apply dimensionality reduction techniques (TAN et al., 2005).

In the context of IR and Text Mining, two common techniques are *Latent Semantic Indexing*, which approximates the term-document matrix by one of lower rank using Singular Value Decomposition (MANNING et al., 2009); and *Minhashing*, which hash document vectors to the same bucket with equal probability of the similarity between them (RAJARAMAN; ULLMAN, 2011).

2.3.7 Self-Organizing Maps

The Self-Organizing Maps (SOM) 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, extracting a latent structure of the input

space while preserving topological and metric relationships. Thus, SOMs can be applied in dimensionality reduction, data visualization, clustering, and classification, among other applications (KOHONEN, 1998; YIN, 2008).

The SOM consist of M neurons located on a regular, usually two-dimensional, grid. Each neuron j is connected to the input and has a prototype vector $w_j = [w_{j1}, \dots, w_{jd}]$ in a location r_j , with the same number of dimensions d as the input samples. Training is based on a competitive learning model, in which, when presented with a stimulus, neurons compete among themselves for the ownership of this input. The winner, along with its neighbors, then strengthen their relationships with this input, eventually making the map localized, i.e., different local fields will respond to different ranges of inputs.

The learning algorithm consists of first, initializing every prototype w to small numbers randomly, and then repeating the following steps (YIN, 2008):

- i. At each time t , present an input $x(t)$, select the winner,

$$v(t) = \arg \min_{k \in \Omega} \|x(t) - w_k(t)\|.$$

- ii. Update the weights of winner and its neighbors,

$$\Delta w_k(t) = \alpha(t) \eta(v, k, t) [x(t) - w_v(t)].$$

- iii. Repeat until the map converges,

where

- Ω is the set of neuron indexes;
- the coefficients $\{\alpha(t), t \geq 0\}$, termed adaptation gain, or learning rate, are scalar-valued, decrease monotonically, and satisfy (i) $0 < \alpha(t) < 1$; (ii) $\lim_{t \rightarrow \infty} \sum \alpha(t) \rightarrow \infty$; (iii) $\lim_{t \rightarrow \infty} \alpha(t) \rightarrow 0$; and
- $\eta(v, k, t)$ is the neighborhood function, which can be the original stepped type of neighborhood function (is one when the neuron is within the neighborhood or zero otherwise), a Gaussian form is often used in practice, i.e. $\eta(v, k, t) = \exp \left[-\frac{\|v - k\|}{2\sigma(t)^2} \right]$, with σ representing the changing effective range of the neighborhood.

Note that winners, also called the best-matching unit (BMU), can also be calculated using any measure of association, changing accordingly in case it is a similarity measure to be the *arg max* of the similarity function.

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 (VESANTO, 2002).

These features and the possible variations and parameters of the SOM 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.

One of the most traditional representations of the trained SOM is the unified distance matrix, or U-Matrix, for short (GORRICHIA; LOBO, 2012). It is formed by U-heights, calculated over the distance of prototypes and their closest neighbors in the map. Formally, let $U_i = \{n_j | d(n_j, n_i) < u, n_j \neq n_i\}$ for some small positive u , the U-height of a neuron $uh(n_i)$ is given as

$$uh(n_i) = \sum_{n_j \in U_i} d(n_i, n_j).$$

Typical visualizations are coloured contour plots on top of the SOM floor, and delivers a “landscape” of the distance relationships of the input data in the data space, allowing one to visually inspect for possible cluster structures, or even outliers (ULTSCH, 2003).

3 METHODOLOGY

Recall from [Figure 2.1](#), that the KDD process, here also used for the text mining process, has 5 steps: selection, preprocessing, transformation, data mining and interpretation/evaluation. Since our goal is to define a methodology to parse the statements into a numerical representation, this work focus in the preprocessing and transformation steps, with the outcome of a term-document matrix, as depicted in [Figure 3.1](#)

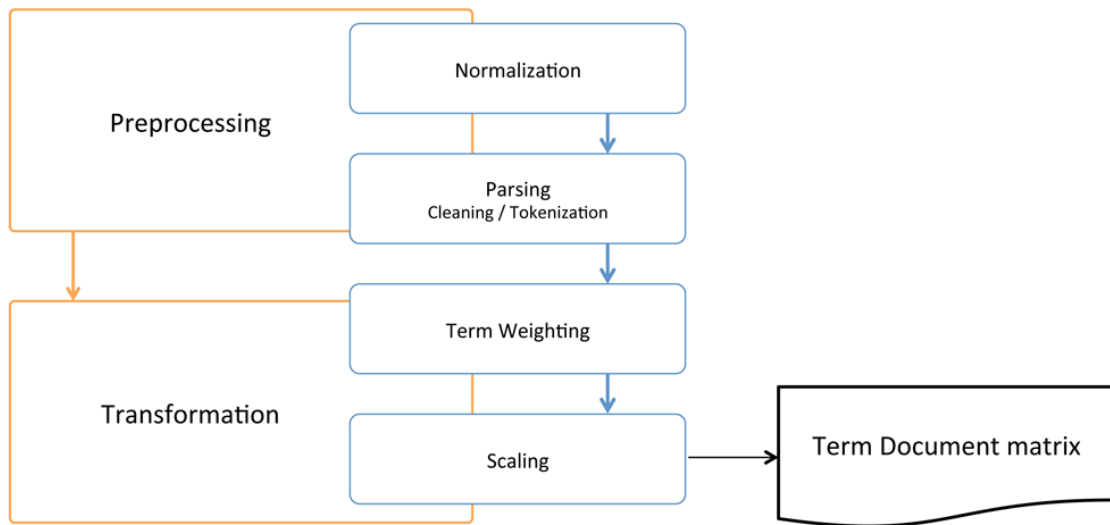


Figure 3.1 - The methodology flowchart.

Thus, this chapter lays out and discuss the actions taken in such steps. We also give an overview of the selection performed to build our target dataset, and the data mining activity for our proof-of-concept experiments, with interpretation of the results being discussed in [Chapter 4](#).

3.1 Selection

Our document collection, as stated before, is the historic logs of SQL queries submitted to SkyServer. In this work, we make use of 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.

SkyServer has a number of different access interfaces, called *requestors* in the logs. The two main form of access, however, are through the *ad-hoc* SQL submission page, also known as the online version; and a batch version, called CasJobs. The online version performs synchronous requests, and thus has a timeout of 10 minutes, limits the total result to a maximum of 500.000 rows and only allows SELECT statements. The batch version, on the other hand, to overcome such limitations, implements an asynchronous request queue having no restrictions on running time or results and also provides a personal database for temporary data storage and full SQL capabilities, like personal stored procedures or function definitions.

With the intent to simplify our target dataset for validation of this methodology, we filtered the queries coming from the last version of the online interface (*skyserver.sdss3.org* requestor), with the assumption that due to the restrictions applied in the search tool, would produce a set of queries with less variance and complexity. This filter also restricted queries with errors and no rows returned.

SkyServer provides extensive documentation on the database and SQL for inexperienced users, which includes a list of sample template queries. These are also part of the target dataset, which we eventually want to correlate with similar queries from the logs.

3.2 Preprocessing

The main objective of the preprocessing phase is to parse the text queries into a vector representation, in which each dimension represents a token and its count of occurrences in that query, or document.

Recall from [subsection 2.2.1](#) that the tokenization process can be as simple as splitting white space in text. SQL however, as a programming language, has a formal structure and syntax and can be more complex than that for tokenization purposes. Consider function calls and parameters, for instance:

```
str(ISNULL(z2.photozerrd1,0),9,7) as photozerrd1
```

This expression is a select argument, made of two nested function calls, one to return 0 in case the column has a null value, the second to convert numeric data into character data based on total length and precision. On a simplistic approach of splitting white spaces, this would render three different tokens (`str(ISNULL(z2.photozerrd1,0),9,7)`, `as`, and `photozerrd1`), with the first one clearly grouping more tokens than it should.

Consider now a second expression:

```
str(ISNULL(z2.photozerrd1, 0), 9, 7) as photozerrd1
```

This expression has the same validity and result as the first one, exactly due to SQL syntax, which makes white space sometimes irrelevant. For this example, any combination of white space before or after commas and parenthesis would have no effect in the output. Still considering a white space only approach, we could have a number of different tokens for the same syntactic expression.

Thus a proper parsing is warranted, that considers such syntax and can properly accounts for cases like this.

Though SQL's structure adds some complexity to the process, by using a parser engine, we can also add a layer of metadata on top of each token according to its semantics (whether it is a *select*, *from* or *where* argument; if its a column or table name, function, expression, or constant), allowing a different processing according to the token type. By knowing there is a formal structure also removes the need for otherwise common steps both in text mining, like stop words removal (present in natural language texts), as in data mining, such as handling missing values (every term not present in a document has just a 0 count in the vector representation).

In the interest of extracting only the most representative tokens from each query, the SQL parser performs the following:

- normalize all characters to lowercase;
- remove constants (strings and numbers), database namespaces and aliases;
- substitute temporary table names, logical and conditional operators for keywords;
- qualify each token with its SQL group: *select*, *from*, *where*, *group by*, and *order by*;

An example of an original statement and its normalized version is shown in Figure 3.2. Figure 3.3 shows the final feature vector.

```

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

```

(a) Raw SQL query.

```

select objid ra dec u g r i z plate fiberid elodiefeh
from   photoobj fgetnearbyobjeq specobj latex
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 3.2 - Example of a SQL query and its normalized version. Whitespace is included for readability.

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_latex	1
where_objid	3
where_logic	8
where_bestobjid	1
where_plateid	2
where_class	1
where_r	2
where_g	2
where_plate	1

Figure 3.3 - Feature vector.

Note that, in this case, the feature vector has 23 components, but it is only showing its own tokens. After processing the whole collection, the final number of attributes would be the total number of terms in the vocabulary, with terms that are not part of this particular document having a value of 0.

Substitutions and removals are performed with the intention to account for tokens that being trivial, specific, or freely defined, would be of little contribution in discriminating each query due to its unusual frequency (too high or too low) or ambiguous use.

Figure 3.4 shows an example of three queries that only differ in one of their search criteria, but have essentially the same structure, and are eventually compressed to the same token set.

```
select count(*) from galaxy p, specobj s
where p.objid = s.bestobjid and s.z between 0 and 0.1
```

```
select count(*) from galaxy p, specobj s
where p.objid = s.bestobjid and s.z between 1 and 3
```

```
select count(*) from galaxy p, specobj s
where p.objid = s.bestobjid and s.z between .1 and .7
```

(a) Queries that generated the above token set.

```
select: { count }
from: { galaxy, specobj }
where: { objid, bestobjid, logic, z }
```

(b) A sample token set separated by the SQL group.

Figure 3.4 - Example of a token set and statements that generated it.

3.3 Transformation

Following preprocessing, we already have an intermediate structured representation of the SQL queries, and in this phase, we are interested in fine tuning such representation.

The first of which is to properly weight each feature according to its frequency, using the already introduced TF*IDF weighting scheme from [subsection 2.2.2](#).

Consider the queries below:

```

SELECT g.objid, g.ra, g.dec, g.u, g.g, g.r, g.i, g.z, s.z AS redshift,
       zs.elliptical, zs.spiral, zs.uncertain
FROM Galaxy AS G
     JOIN ZooSpec AS zs ON G.objid = zs.objid
     JOIN specobj AS s ON G.objid = s.bestobjid
WHERE s.z BETWEEN -0.1 AND 0.05

```

```

SELECT TOP 100 p.objid, p.ra, p.dec, p.u, p.g, p.r, p.i, p.z,
              s.class, s.z
FROM PhotoObj AS p
     JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE p.u BETWEEN 0 AND 19.6
     AND g BETWEEN 0 AND 20

```

```

SELECT p.objid, p.ra, p.dec, p.u, p.g, p.r, p.i, p.z, p.psfmag_r, s.z
FROM PhotoObj AS p
     JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE s.z<=0.1
     AND p.ra BETWEEN 0.0 AND 5.0
     AND p.dec BETWEEN 10.0 AND 15.0
     AND (CLASS='galaxy')

```

```

SELECT ra, dec, objID, modelMag_u, modelMag_g, modelMag_r,
       modelMag_i, modelMag_z
FROM Galaxy
WHERE ra BETWEEN 140.9 AND 141.1
     AND dec BETWEEN 20 AND 21
     AND modelMag_g >=18
     AND modelMag_u - modelMag_g > 2.2

```

Taking these three samples as our dataset, after parsing, we would have a vector representation for each statement, which we could already turn into a term-document matrix of term-frequencies.

To calculate the weights, we first would need to define, for each token, its document frequency (df_t), i.e., the number of documents in which that token appears, and the inverse document frequency (idf_t), i.e., the log of the ratio between the total number of documents in the collection (in this case, 4) and its document frequency. After that, the TF*IDF scheme is applied by multiplying each term frequency (tf) by its

idf. Table 3.1 presents all these values: term-frequencies for each statement in the first columns, the document frequency and inverse document frequency, and in the last columns, the final term-document matrix weighted by the TF*IDF scheme.

Table 3.1 - Term-document matrix with term-frequencies in the first columns, the df and idf indexes, and the weighted term-frequencies using the TF*IDF scheme.

	TF						TF*IDF			
	1	2	3	4	df	idf	1	2	3	4
select_class	0	1	0	0	1	1.386	0	1.386	0	0
select_dec	1	1	1	1	4	0	0	0	0	0
select_elliptical	1	0	0	0	1	1.386	1.386	0	0	0
select_g	1	1	1	0	3	0.288	0.288	0.288	0.288	0
select_i	1	1	1	0	3	0.288	0.288	0.288	0.288	0
select_modelmag_g	0	0	0	1	1	1.386	0	0	0	1.386
select_modelmag_i	0	0	0	1	1	1.386	0	0	0	1.386
select_modelmag_r	0	0	0	1	1	1.386	0	0	0	1.386
select_modelmag_u	0	0	0	1	1	1.386	0	0	0	1.386
select_modelmag_z	0	0	0	1	1	1.386	0	0	0	1.386
select_objid	1	1	1	1	4	0	0	0	0	0
select_psfmag_r	0	0	1	0	1	1.386	0	0	1.386	0
select_r	1	1	1	0	3	0.288	0.288	0.288	0.288	0
select_ra	1	1	1	1	4	0	0	0	0	0
select_spiral	1	0	0	0	1	1.386	1.386	0	0	0
select_u	1	1	1	0	3	0.288	0.288	0.288	0.288	0
select_uncertain	1	0	0	0	1	1.386	1.386	0	0	0
select_z	2	2	2	0	3	0.288	0.575	0.575	0.575	0
from_bestobjid	1	1	1	0	3	0.288	0.288	0.288	0.288	0
from_galaxy	1	0	0	1	2	0.693	0.693	0	0	0.693
from_inner	2	1	1	0	3	0.288	0.575	0.288	0.288	0
from_join	2	1	1	0	3	0.288	0.575	0.288	0.288	0
from_objid	3	1	1	0	3	0.288	0.863	0.288	0.288	0
from_on	2	1	1	0	3	0.288	0.575	0.288	0.288	0
from_photoobj	0	1	1	0	2	0.693	0	0.693	0.693	0
from_specobj	1	1	1	0	3	0.288	0.288	0.288	0.288	0
from_zoospec	1	0	0	0	1	1.386	1.386	0	0	0
where_class	0	0	1	0	1	1.386	0	0	1.386	0
where_dec	0	0	1	1	2	0.693	0	0	0.693	0.693
where_g	0	1	0	0	1	1.386	0	1.386	0	0
where_logic	0	1	3	3	3	0.288	0	0.288	0.863	0.863
where_modelmag_g	0	0	0	2	1	1.386	0	0	0	2.773
where_modelmag_u	0	0	0	1	1	1.386	0	0	0	1.386
where_ra	0	0	1	1	2	0.693	0	0	0.693	0.693
where_u	0	1	0	0	1	1.386	0	1.386	0	0
where_z	1	0	1	0	2	0.693	0.693	0	0.693	0

Note that some rows become zero valued after weighting. These cases happen if a given term occurs in every document, and thus have an idf of 0. Such terms might be elected for removal, since they do not have any discriminant power between each document.

The second step in the transformation phase is scaling all features to lie in the 0..1 interval through the simple formula (WITTEN et al., 2011)

$$x_i = \frac{v_i - \min v_i}{\max v_i - \min v_i} ,$$

where v_i is the actual value of attribute i , and the maximum and minimum are taken over all instances in the training set.

Using our test scenario with the four statements presented, the final term-document matrix weighted and scaled would have the values as the one presented in [Table 3.2](#).

3.4 Data Mining

At this stage, we have already processed the document collection into a term-document matrix, where each row represents a SQL statement and columns represent the weighted and scaled frequency of each term in the vocabulary for that statement. Considering this matrix as the dataset, it is ready to be fed into regular machine learning algorithms.

In this work, we are interested in clustering techniques, the exploratory analysis to find natural groupings in the data. As such, we perform two experiments, one with the FCM algorithm and its cluster validity indexes to assess an optimal number of clusters in the dataset, the other with the SOM algorithm to make use of its dimensionality reduction and visualization capabilities.

Table 3.2 - Term-document matrix, transformed to be appropriately weighted and scaled.

	1	2	3	4
select_class	0.208	0.208	0.208	0
select_dec	0.500	0	0	0.250
select_elliptical	0.415	0.208	0.208	0
select_g	0.415	0.208	0.208	0
select_i	0.623	0.208	0.208	0
select_modelmag_g	0.415	0.208	0.208	0
select_modelmag_i	0	0.500	0.500	0
select_modelmag_r	0.208	0.208	0.208	0
select_modelmag_u	1.000	0	0	0
select_modelmag_z	0	1.000	0	0
select_psfmag_r	1.000	0	0	0
select_r	0.208	0.208	0.208	0
select_ra	0.208	0.208	0.208	0
select_spiral	0	0	0	0.500
select_u	0	0	0	0.500
select_uncertain	0	0	0	0.500
select_z	0	0	0	0.500
from_bestobjid	0	0	0	0.500
from_inner	0	0	1.000	0
from_join	0.208	0.208	0.208	0
from_on	1.000	0	0	0
from_photoobj	0.208	0.208	0.208	0
from_specobj	1.000	0	0	0
from_zoospec	0.415	0.415	0.415	0
where_class	0	0	1.000	0
where_dec	0	0	0.500	0.250
where_g	0	1.000	0	0
where_logic	0	0.208	0.623	0.311
where_modelmag_g	0	0	0	1.000
where_modelmag_u	0	0	0	0.500
where_ra	0	0	0.500	0.250
where_u	0	1.000	0	0
where_z	0.500	0	0.500	0

4 EXPERIMENTAL RESULTS

4.1 On data and implementation

The initial dataset (the normalized version by [Raddick et al. \(2014\)](#)) was originally composed of almost 195 million records and 68 million unique queries. After filtering as described in [section 3.1](#), the final dataset was reduced to 1.3 million queries, plus 49 sample templates from SkyServer’s help pages.

Data was downloaded in a CSV format and imported into a MongoDB instance, a document oriented, non-relational database. The choice was based on the schemaless paradigm of NoSQL databases, which provided great flexibility while building the target dataset. Querying is made programatically through a number of bindings provided, or directly through a JavaScript interactive shell.

A number of open-source SQL parsers were investigated, but since SkyServer uses the Microsoft SQL Server as its RDMBS, it accepts queries in the Transact-SQL dialect, or T-SQL, which is Microsoft’s proprietary extension to SQL implementing a number of features like stored procedures, local variables, data processing, etc. Thus, standard SQL parsers would not be able to process T-SQL intricacies, and eventually we decided to use a readily available parser library from .NET, the software framework also developed by Microsoft, which served as base for a custom parser, tailored for our needs. Note that the parser is strict, ergo, it can only process syntax valid statements. The code for the custom parser built is presented in the [Appendix A](#).

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 queries, 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.

Python was the main programming language used, and a number of scripts were written to perform the various tasks needed, from implementing the custom parser, to the SOM algorithm (which was based on the work of [Vettigli \(2015\)](#)). For FCM specifically, R was chosen because of its `e1071` package ([MEYER et al., 2015](#)). Finally, most of the computing was performed on a Intel Xeon 3.4 GHz machine with

32 cores and 66 GB of RAM, running a 64-bit implementation of Linux.

4.2 Analysis of number of clusters with FCM

This experiment consisted of clustering the dataset and then calculate the four different validity measures presented in subsection 2.3.5: partition coefficient, partition entropy, Fukuyama-Sugeno and Xie-Beni. Literature usually recommends the range of c to be from 2 to $N - 1$, where N is the number of samples in the dataset. Since it is usually infeasible in regards to time, we limited c to be in the $2 \dots 100$ interval.

As expected, as c increases, training time increases, and the squared error criterion decreases (FCM's objective function), but in this case, the number of iterations needed is rather stable, with an average of 11 iterations needed, as seen in Figure 4.1.

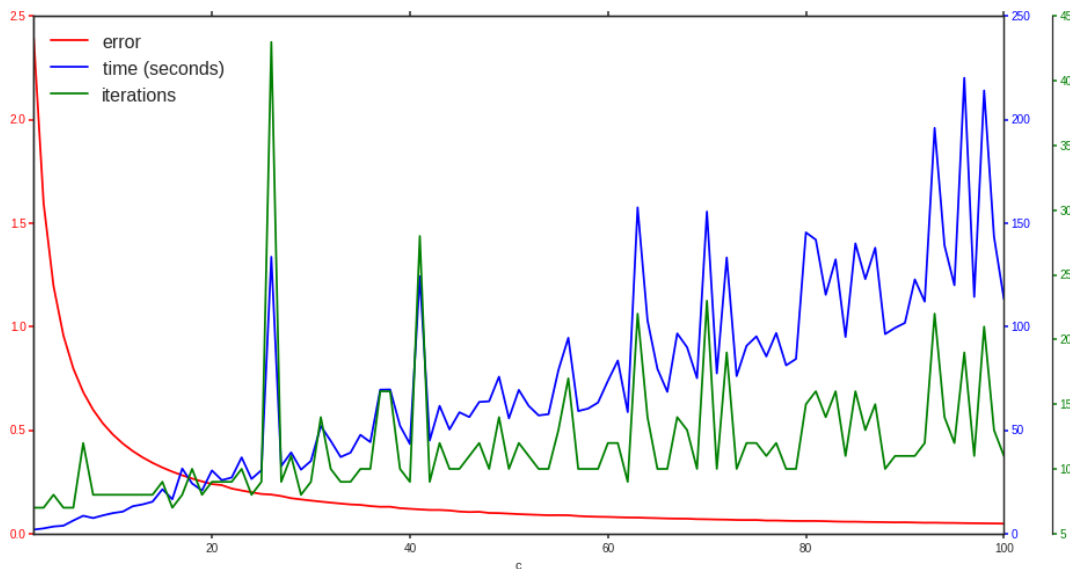


Figure 4.1 - FCM training metrics for different values of c .

The cluster validity metrics are presented in Figure 4.2.

Recall that we seek the maximum for the partition coefficient and the minimum for the other three indexes. Visually inspecting the Figure 4.2, however, we can see that there is no value of c that would have more than one index agreeing with each other. Thus, one might consider that these metrics suggest this dataset does not present a natural grouping.

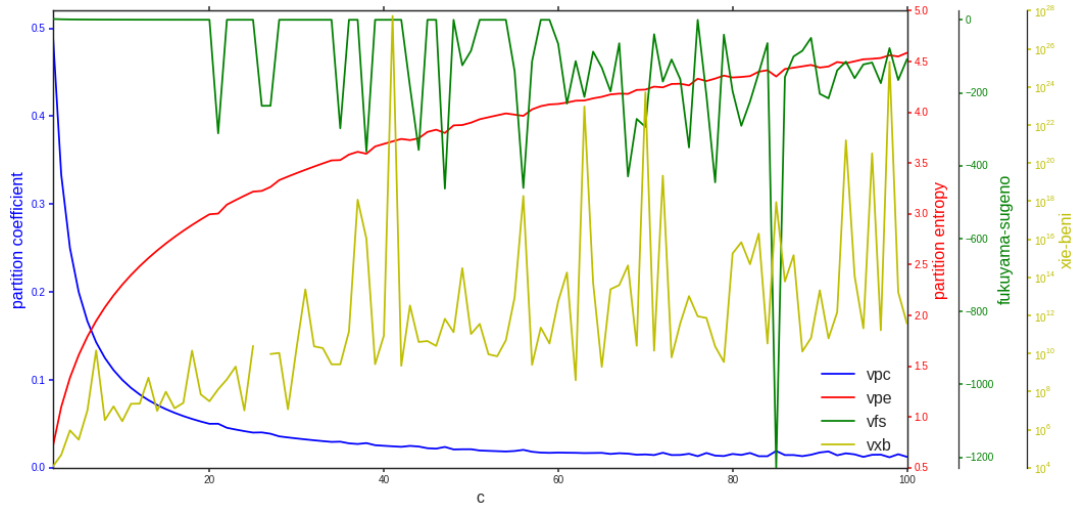


Figure 4.2 - FCM cluster validity measures for different values of c .

4.3 Visual analysis of the correlation between queries and templates

For this experiment, we used a 30x30 SOM trained for 45 epochs, using the cosine distance to determine the BMU during training phase.

We used two plots for an initial visual analysis, the u-matrix, presented in Figure 4.3, in which numbers indicate the template id over their respective BMU, and a hitmap scatter plot, presented in Figure 4.4, in which the size of the circles indicates the number of token sets that elected that prototype its BMU.

From Figure 4.3 and Figure 4.4, 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.

- a) **Pair:** 15 and 15b
Distances: TF: 0.0 and SOM: 0.0

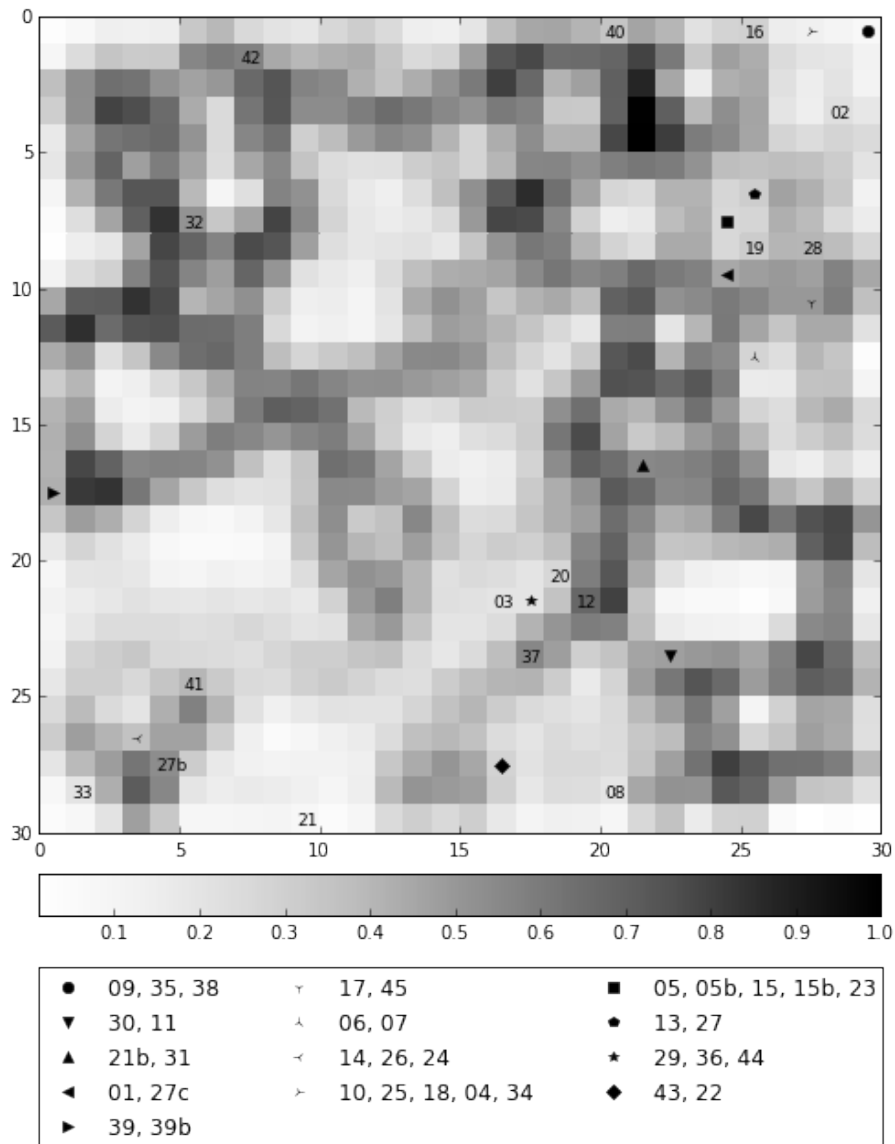


Figure 4.3 - U-Matrix

15: Splitting 64-bit values into two 32-bit values

15b: Splitting 64-bit values into two 32-bit values

b) **Pair**: 21b and 31

Distances: TF: 0.0 and SOM: 0.0

21b: Finding objects by their spectral lines

31: Using the sppLines table

c) **Pair**: 22 and 43

Distances: TF: 0.0205 and SOM: 0.0

22: Finding spectra by classification (object type)

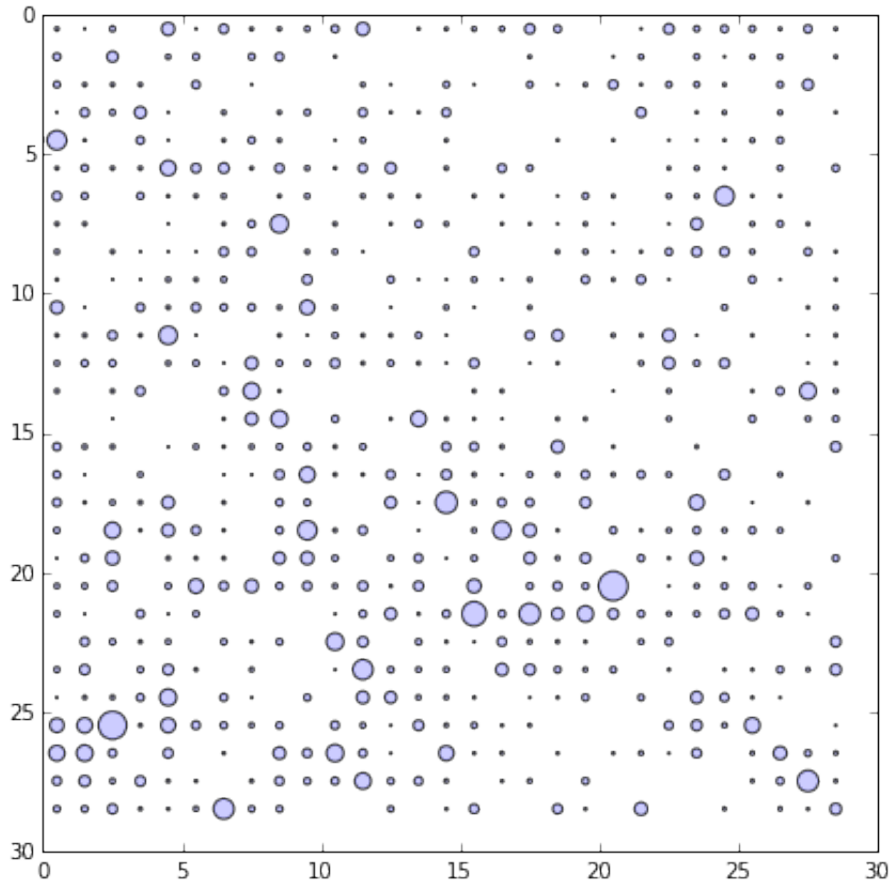


Figure 4.4 - Hitmap

43: QSOs by spectroscopy

d) **Pair:** 39 and 39b

Distances: TF: 0.1610 and SOM: 0.0

39: Classifications from Galaxy Zoo

39b: Classifications from Galaxy Zoo

e) **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](#).

5 CONCLUSIONS

The main goal of this thesis was to investigate text mining techniques for the processing and analysis of the historic logs of SQL queries from SDSS SkyServer. As such, we defined a methodology to properly parse, clean and tokenize such statements into a proper intermediate numerical representation, allowing then, the use of regular data mining algorithms for knowledge discovery, with preliminar experiments showcasing an example of how such methodology can be used.

Also note that the preprocessing and transformation involved in this work are not definitive, and can accommodate changes according to the data mining objective. The parser, for instance, can be quickly adapted to extract or engineer new features as seem fit. If one was to build a similar map of popular searched areas as devised by [Zhang et al. \(2012\)](#), the methodology could be tuned to select the queries with the functions and column names related to this criteria from the already parsed queries and then update the parser to extract the numeral parameters of interest in the selected queries.

Foreseen applications for this methodology include, but are not limited to, generation of detailed usage statistics, with specific information on tables and columns most popularly queried, which can lead to better database indexes and views management, improving performance according to user needs; improving user experience with queries recommendation tools or assistive technologies to offer users suggestions while writing queries, improving user exploration; and finally, by correlating token sets with other features logged, such as query success or running time, one could devise classification models to predict errors in running time or regression models to predict query running time.

As part of the work done in this thesis, we also had accepted a poster presentation for the IASC-ABE Satellite Conference for the 60th ISI WSC, 2015; and a short article for the 2nd Annual International Symposium on Information Management and Big Data SIMBig, 2015.

REFERENCES

- ALAM, S. et al. The eleventh and twelfth data releases of the Sloan Digital Sky Survey: final data from SDSS-III. **The Astrophysical Journal Supplement Series**, v. 219, n. 1, p. 12, jul. 2015. ISSN 1538-4365. Available from: <<http://arxiv.org/abs/1501.00963v3>>. 1
- BERKHIN, P. A survey of clustering data mining techniques. **Grouping multidimensional data**, p. 25–71, 2006. Available from: <http://link.springer.com/chapter/10.1007/3-540-28349-8_2>. 14, 16
- CHI, Z.; YAN, H.; PHAM, T. **Fuzzy algorithms: with applications to image processing and pattern recognition**. World Scientific, 1996. 232 p. (Advances in Fuzzy Systems - Applications and Theory, v. 10). ISBN 978-981-02-2697-8. Available from: <<http://www.worldscientific.com/worldscibooks/10.1142/3132>>. 13, 15, 16
- FAN, W.; WALLACE, L.; RICH, S.; ZHANG, Z. Tapping the power of text mining. **Communications of the ACM**, v. 49, n. 9, p. 76–82, 2006. ISSN 00010782. Available from: <<http://portal.acm.org/citation.cfm?doid=1151030.1151032>>. 5, 6
- FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. **AI magazine**, p. 37–54, 1996. ISSN 0738-4602. Available from: <<http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1230>>. 5, 6
- FELDMAN, R.; SANGER, J. **The text mining handbook: advanced approaches in analyzing unstructured data**. Cambridge: Cambridge University Press, 2006. 423 p. ISBN 9780511546914. Available from: <<http://ebooks.cambridge.org/ref/id/CB09780511546914>>. 5
- GIONIS, A.; INDYK, P.; MOTWANI, R. Similarity search in high dimensions via hashing. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB'99), 25., 1999, Edinburgh, Scotland. **Proceedings...** Edinburgh, Scotland: Morgan Kaufmann, 1999. p. 518–529. ISBN 1-55860-615-7. Available from: <<http://www.vldb.org/conf/1999/P49.pdf>>. 13

GORRICHA, J.; LOBO, V. Improvements on the visualization of clusters in geo-referenced data using self-organizing maps. **Computers & Geosciences**, Elsevier, v. 43, p. 177–186, 2012. 19

HAVELIWALA, T. H.; GIONIS, A.; KLEIN, D.; INDYK, P. Evaluating strategies for similarity search on the web. In: INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 11., 2002, Honolulu, HW. **Proceedings...** Honolulu, HW: ACM, 2002. v. 29, n. 8, p. 432. ISBN 1581134495. Available from: <<http://doi.acm.org/10.1145/511446.511502>>. 13

HOWE, B. et al. Database-as-a-service for long-tail science. In: INTERNATIONAL CONFERENCE SCIENTIFIC AND STATISTICAL DATABASE MANAGEMENT (SSDBM 2011), 23., 2011, Portland, OR. **Proceedings...** Portland, OR: Springer, 2011. p. 480–489. ISBN 978-3-642-22350-1. Available from: <http://dx.doi.org/10.1007/978-3-642-22351-8_31>. 2

HUANG, A. Similarity measures for text document clustering. In: NEW ZEALAND COMPUTER SCIENCE RESEARCH STUDENT CONFERENCE, 2008. **Proceedings...** 2008. p. 49–56. Available from: <http://nzcsrsc08.canterbury.ac.nz/site/proceedings/Individual_Papers/pg049_Similarity_Measures_for_Text_Document_Clustering.pdf>. 13

JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. **ACM computing surveys (CSUR)**, v. 31, n. 3, p. 264–323, 1999. 10, 14

JONES, K. S. S. A statistical interpretation of term specificity and its application in retrieval. **Journal of documentation**, MCB UP Ltd, v. 28, n. 1, p. 11–21, 1972. ISSN 0022-0418. Available from: <<http://www.emeraldinsight.com/10.1108/00220410410560573>>. 9

KENT, W. J. et al. The Human Genome Browser at UCSC. **Genome Research**, v. 12, n. 6, p. 996–1006, may 2002. ISSN 1088-9051. Available from: <<http://www.genome.org/cgi/doi/10.1101/gr.229102>>. 2

KOHONEN, T. The self-organizing map. **Neurocomputing**, Elsevier, v. 21, n. 1-3, p. 1–6, nov. 1998. ISSN 09252312. Available from: <<http://linkinghub.elsevier.com/retrieve/pii/S0925231298000307>>. 18

LARSEN, B.; AONE, C. Fast and effective text mining using linear-time document clustering. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE

DISCOVERY AND DATA MINING (SIGKDD 1999), 5., 1999, San Diego, CA. **Proceedings...** San Diego, CA: ACM, 1999. v. 5, n. 5, p. 16–22. ISBN 1581131437. Available from: <<http://doi.acm.org/10.1145/312129.312186>>. 9

LEVANDOWSKY, M.; WINTER, D. Distance between sets. **Nature**, v. 234, n. 5323, p. 34–35, nov. 1971. ISSN 0028-0836. Available from: <<http://www.nature.com/doifinder/10.1038/234034a0>>. 12

LUHN, H. P. The automatic creation of literature abstracts. **IBM Journal of Research and Development**, v. 2, n. 2, p. 159–165, 1958. ISSN 0018-8646. Available from: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5392672>>. 8, 9

MADRID, J. P.; MACCHETTO, D. High-impact astronomical observatories. p. 2006–2007, jan. 2009. ISSN 1095-9203. Available from: <<http://arxiv.org/abs/0901.4552>>. 1

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to information retrieval**. Cambridge University Press, 2009. 544 p. ISBN 0521865719. Available from: <<http://nlp.stanford.edu/IR-book/>>. 7, 8, 9, 11, 12, 14, 17

MEYER, D. et al. **e1071: misc functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien**. 2015. Available from: <<https://cran.r-project.org/web/packages/e1071/index.html>>. Access in: 2015-08-25. 31

MORAIS, A. M. M.; QUILES, M. G.; SANTOS, R. D. C. Icon and geometric data visualization with a self-organizing map grid. In: **Computational Science and Its Applications â ICCSA 2014**. Springer International Publishing, 2014, (Lecture Notes in Computer Science, v. 8584). p. 562–575. ISBN 978-3-319-09152-5. Available from: <http://dx.doi.org/10.1007/978-3-319-09153-2_42>. 19

PAL, N. R.; BEZDEK, J. C. On cluster validity for the fuzzy c-means model. **IEEE Transactions on Fuzzy Systems**, v. 3, n. 3, p. 370–379, 1995. ISSN 10636706. 16

RADDICK, M. J.; THAKAR, A. R.; SZALAY, A. S.; SANTOS, R. D. C. Ten years of SkyServer I: tracking web and SQL e-Science usage. **Computing in Science & Engineering**, v. 16, n. 4, p. 22–31, 2014. 2, 21, 31

- RAJARAMAN, A.; ULLMAN, J. **Mining of massive datasets**. 2en. ed. Cambridge University Press, 2011. 511 p. ISBN 1107015359. Available from: <<http://infolab.stanford.edu/~ullman/mmds.html>>. 11, 12, 14, 17
- RIJSBERGEN, C. J. van. **Information retrieval**. 2nd. ed. Butterworths, 1979. 208 p. ISBN 0408709294. Available from: <<http://www.dcs.gla.ac.uk/Keith/Preface.html>>. 8, 11, 12, 13
- SALTON, G.; WONG, A.; YANG, C. S. A vector space model for automatic indexing. **Communications of the ACM**, v. 18, n. 11, p. 613–620, nov. 1975. ISSN 00010782. Available from: <<http://portal.acm.org/citation.cfm?doid=361219.361220>>. 9
- SDSS. **Skyserver**. 2015. Available from: <<http://skyserver.sdss3.org>>. Access in: 2015-08-25. 1
- SINGH, V. et al. SkyServer traffic report - the first five years. **Microsoft Technical Report**, jan. 2006. Available from: <<http://arxiv.org/abs/cs/0701173>>. 2, 8
- STOUGHTON, C. et al. Sloan Digital Sky Survey: early data release. **The Astronomical Journal**, v. 123, n. 1, p. 485–548, jan. 2002. ISSN 00046256. Available from: <<http://stacks.iop.org/1538-3881/123/i=1/a=485>>. 1
- STREHL, A.; GHOSH, J.; MOONEY, R. Impact of similarity measures on web-page clustering. In: WORKSHOP ON ARTIFICIAL INTELLIGENCE FOR WEB SEARCH (AAAI 2000), 2000. **Proceedings...** [S.l.], 2000. p. 58–64. 13
- SZALAY, A. S. et al. The SDSS SkyServer: public access to the Sloan Digital Sky Server data. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA (SIGMOD 2002), 2002, Madison, WI. **Proceedings...** ACM, 2002. p. 570—581. Available from: <<http://doi.acm.org/10.1145/564691.564758>>. 1
- TAN, A.-H. Text mining: the state of the art and the challenges. In: WORKSHOP ON KNOWLEDGE DISCOVERY FROM ADVANCED DATABASES (PAKDD 1999), 1999. **Proceedings...** [S.l.], 1999. (KDAD'99), p. 71–76. 5
- TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to data mining**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. 769 p. ISBN 0321321367. 13, 17

THAKAR, A.; SZALAY, A.; KUNSZT, P.; GRAY, J. Migrating a multiterabyte archive from object to relational databases. **Computing in Science & Engineering**, v. 5, n. 5, p. 16–29, sep. 2003. ISSN 1521-9615. Available from: <<http://scitation.aip.org/content/aip/journal/cise/5/5/10.1109/MCISE.2003.1225857>>. 1

ULTSCH, A. Maps for the visualization of high-dimensional data spaces. In: WORKSHOP ON SELF-ORGANIZING MAPS, 2003. **Proceedings...** 2003. p. 225–230. ISBN 086332424X. Available from: <<http://www.informatik.uni-marburg.de/~databionics/papers/ultsch03maps.pdf>>. 19

VESANTO, J. **Data exploration process based on the self-organizing map**. PhD Thesis (PhD) — Helsinki University of Technology, 2002. 19

VETTIGLI, G. **MiniSom: minimalistic and numpy based implementation of the self organizing maps**. 2015. Available from: <<http://github.com/JustGlowing/minisom>>. Access in: 2015-04-17. 31

WITTEN, I. H.; FRANK, E.; HALL, M. A. **Data mining: practical machine learning tools and techniques**. Third. [S.l.]: Morgan Kaufmann, 2011. 629 p. ISBN 9780387312347. 28

YIN, H. Learning nonlinear principal manifolds by self-organising maps. In: GORBAN, A. N.; KéGL, B.; WUNSCH, D. C.; ZINOVYEV, A. Y. (Ed.). **Principal Manifolds for Data Visualization and Dimension Reduction**. Springer Berlin Heidelberg, 2008. chapter 3, p. 68–95. ISBN 9783540737490. Available from: <http://link.springer.com/10.1007/978-3-540-73750-6_3>. 18

ZHANG, J. et al. SDSS Log Viewer: visual exploratory analysis of large-volume SQL log data. **Visualization and Data Analysis**, v. 8294, p. 82940D, 2012. Available from: <<http://dx.doi.org/10.1117/12.907097>>. 2, 37

APPENDIX A - PARSER

Below is the reproduction of two Python code files. The first one presents a sample script showing how to use the parser classes to tokenize statements. While the second one presents the code for the custom parser built on top of the .NET ScriptDom library.

Note that this code was written to be run over IronPython, a Python implementation for .NET, and will not work under other implementations.

tokenizer.py

This script reads statements separated by a new line from a text file, and prints the tokenized version of each statement after parsing.

```
#!/ mono ipy
import sys
import os
import clr
import System
clr.AddReference('Microsoft.SqlServer.TransactSql.ScriptDom.dll')
import Microsoft.SqlServer.TransactSql.ScriptDom as sd

import classes

def getString(node):
    return ''.join([t.Text for t in list(node.ScriptTokenStream)[node.
        FirstTokenIndex:node.LastTokenIndex+1]])

def __clause(node):
    try:
        return getString(node)
    except:
        return None

parser = sd.TSql100Parser(1)
filename = 'query.txt'

with open(filename) as f:
    for line in f:
        stream = System.IO.StringReader(line.lower())
        fragment, parse_errors = parser.Parse(stream)
        stream.Close()
        errors = ''
        if parse_errors.Count:
```

```

errors = ('The following errors were caught:\n',)
for err in parse_errors:
    errors += ('—', err.Message, '\n')
try:
    for stmt in fragment.Batches[0].Statements:
        sv = classes.Visitor()
        stmt.AcceptChildren(sv)
        qe = stmt.QueryExpression
        query = {
            'modifiers': [_clause(qe.TopRowFilter), qe.UniqueRowFilter],
            'select': ', '.join(map(getString, qe.SelectElements)),
            'from': _clause(qe.FromClause),
            'where': _clause(qe.WhereClause),
            'orderby': _clause(qe.OrderByClause),
            'groupby': _clause(qe.GroupByClause)
        }
        print '—'
        print 'Query:', line
        for key in ['select', 'modifiers', 'from', 'where', 'orderby',
            'groupby']:
            print '-', key
            print 'query:', query[key]
            print 'keywords:', sv.keywords.get(key)
except:
    print sys.exc_info()
finally:
    print ''.join(errors)

```

parser.py

```

import sys
import logging
from collections import defaultdict

import System
import clr
clr.AddReference('Microsoft.SqlServer.TransactSql.ScriptDom.dll')
import Microsoft.SqlServer.TransactSql.ScriptDom as sd

logging.basicConfig(format='%(asctime)s %(levelname)6s [pid %(process)5s] %(message)s')
logger = logging.getLogger()

# for debugging purposes
class plist(list):

```

```

def append(self, value):
    logger.log(1, 'appending %s', value)
    super(plist, self).append(value)

def getString(node):
    return ''.join([t.Text for t in list(node.ScriptTokenStream)[node.
        FirstTokenIndex:node.LastTokenIndex+1]])

def __skip__children(fn):
    def wrapped(self, node):
        fn(self, node)
        node.Accept(self.skipVisitor)
    return wrapped

class BaseVisitor(sd.TSqlFragmentVisitor):
    def __init__(self):
        self.nodes = set()
        self.keywords = defaultdict(plist)
        self.skipVisitor = SkipVisitor(self)
    def __getEnumValue(self, enum):
        return enum.ToString().lower()
    def __callMethodByType(self, __type, node):
        logger.log(1, "[%15s] %s: %s", self.__class__.__name__, __type,
            getString(node))
        return getattr(self, __type)(node)
    def __visit(self, node):
        try:
            __type = node.GetType().Name
            self.__callMethodByType(__type, node)
        except AttributeError:
            logger.log(1, "[%15s] Method %s not found", self.__class__.
                __name__, __type)
        except Exception as e:
            logger.log(1, "[%15s] Exception", self.__class__.__name__,
                exc_info=e)
    def Visit(self, node):
        super(BaseVisitor, self).Visit(node)
        if node not in self.nodes:
            self.nodes.add(node)
            self.__visit(node)

class PrintVisitor(BaseVisitor):
    def __init__(self, parent=None):
        if parent:
            self.nodes = parent.nodes

```

```

        self.keywords = parent.keywords
    else:
        super(PrintVisitor, self).__init__()
def _visit(self, node):
    print '%-30s: %s' % (node.GetType().Name, getString(node))

class SkipVisitor(BaseVisitor):
def __init__(self, parent):
    self.nodes = parent.nodes
def _visit(self, node):
    pass

class Visitor(BaseVisitor):
def QuerySpecification(self, node):
    # modifiers
    urf = node.UniqueRowFilter
    if urf == urf.Distinct:
        self.keywords['select'].append('distinct')
        # self.keywords['modifiers'].append(self._getEnumValue(urf.
            Distinct))
    try:
        trf = node.TopRowFilter
        _keywords = ['top', 'percent', 'withties']
        _filter = [1, trf.Percent, trf.WithTies]
        self.keywords['modifiers'].extend([i for (i, v) in zip(_keywords,
            _filter) if v])
    except:
        pass
    sv = SelectVisitor(self)
    for elm in node.SelectElements:
        elm.Accept(sv)
def FromClause(self, node):
    node.AcceptChildren(FromVisitor(self))
def WhereClause(self, node):
    node.AcceptChildren(WhereVisitor(parent=self))
def OrderByClause(self, node):
    node.AcceptChildren(OrderByVisitor(self))
def GroupByClause(self, node):
    node.AcceptChildren(GroupByVisitor(self))
def HavingClause(self, node):
    node.AcceptChildren(HavingVisitor(self))

class ChildVisitor(BaseVisitor):
    key = None
    fn_blacklist = ['cast', 'format', 'str']

```

```

def __init__(self, parent):
    self.parent = parent
    self.nodes = parent.nodes
    self.skipVisitor = parent.skipVisitor
    try:
        self.keywords = parent.keywords[self.key]
    except:
        self.keywords = parent.keywords

def _visit(self, node):
    _type = node.GetType()
    try:
        self._callMethodByType(_type.Name, node)
    except AttributeError:
        while True:
            # try parents type
            try:
                _type1, _type = _type, _type.BaseType
                if _type1.Name == 'TSqlFragment':
                    break
                logger.log(1, "[%15s] Method %s not found, retrying with %s",
                           self.__class__.__name__, _type1.Name, _type.Name)
                self._callMethodByType(_type1.Name, node)
                break
            except AttributeError:
                continue
        except Exception as e:
            logger.log(1, "[%15s] Exception", self.__class__.__name__,
                       exc_info=e)

def _visitchildren(self, node):
    self.AcceptChildren(self)

# general
def VariableReference(self, node):
    self.keywords.append('variable')

def Literal(self, node):
    # dismissing every literal, othwerwise, uncomment the following
    # lines
    pass
    # literal = self._getEnumValue(node.LiteralType)
    # literal = literal in ['numeric', 'integer', 'real'] and 'number'
    # or literal
    # self.keywords.append(literal)

```

```

# expressions
def _getExprToken(self, node):
    _expr = {
        'BinaryExpression': 'operand',
        'BooleanBinaryExpression': 'logic',
        'BooleanComparisonExpression': 'compare',
        'BooleanNotExpression': 'not',
        'CoalesceExpression': 'coalesce',
        'ExistsPredicate': 'exists',
        'FullTextPredicate': 'contains',
        'LikePredicate': 'like',
        'NullIfExpression': 'nullif',
    }
    try:
        return _expr[node.GetType().Name]
    except KeyError:
        try:
            return self._getEnumValue(node.TernaryExpressionType)
        except:
            return None

def _expression(self, node):
    try:
        node.Expression.Accept(self)
    except:
        # accepts everything
        node.AcceptChildren(self)
def _prepend_expression(self, node):
    self.keywords.append(self._getExprToken(node))
    self._expression(node)

def BinaryExpression(self, node):
    node.FirstExpression.Accept(self)
    node.SecondExpression.Accept(self)

BooleanComparisonExpression = BinaryExpression
BooleanBinaryExpression = BinaryExpression

def BooleanTernaryExpression(self, node):
    self.BinaryExpression(node)
    node.ThirdExpression.Accept(self)

ScalarExpression = _expression
BooleanParenthesisExpression = _expression

```



```

PrimaryExpression = _expression
ParenthesisExpression = _expression

BooleanNotExpression = __prepend__expression
CoalesceExpression = __prepend__expression
ExistsPredicate = __prepend__expression
FullTextPredicate = __prepend__expression
NullIfExpression = __prepend__expression

def CaseExpression(self, node):
    self.keywords.append('case')
    for t in node.WhenClauses:
        self.keywords.append('when')
        t.WhenExpression.Accept(self)
        t.ThenExpression.Accept(self)
    if node.ElseExpression:
        self.keywords.append('else')
        node.ElseExpression.Accept(self)
@_skip_children
def CastCall(self, node):
    node.Parameter.Accept(self)
@_skip_children
def ColumnReferenceExpression(self, node):
    ids = node.MultiPartIdentifier.Identifiers
    identifier = ids[ids.Count-1]
    # for simplicity, consider every doublequoted identifier as
    # constant
    if self._getEnumValue(identifier.QuoteType) == 'doublequote':
        return
    self.keywords.append(identifier.Value)
@_skip_children
def FunctionCall(self, node):
    if node.FunctionName.Value not in self.fn_blacklist:
        self.keywords.append(node.FunctionName.Value)
    for p in node.Parameters:
        p.Accept(self)

# select
@_skip_children
def SelectScalarExpression(self, node):
    node.Expression.Accept(self)
def SelectStarExpression(self, node):
    self.keywords.append('*')

# from

```

```

@_skip_children
def __getSchemaObjectBase(self, node):
    obj = node.SchemaObject.BaseIdentifier.Value
    self.keywords.append('#' in obj and 'temp' or obj)
NamedTableReference = __getSchemaObjectBase
SchemaObjectFunctionTableReference = __getSchemaObjectBase
def QualifiedJoin(self, node):
    node.FirstTableReference.Accept(self)
    self.keywords.extend([self.__getEnumValue(node.QualifiedJoinType),
        'join'])
    node.SecondTableReference.Accept(self)
    self.keywords.append('on')
    node.SearchCondition.Accept(self)
def UnqualifiedJoin(self, node):
    node.FirstTableReference.Accept(self)
    self.keywords.append(self.__getEnumValue(node.UnqualifiedJoinType)
        )
    node.SecondTableReference.Accept(self)

# sub-queries
def QuerySpecification(self, node):
    self.nodes.remove(node)
    node.Accept(self.parent)

class SelectVisitor(ChildVisitor):
    key = 'select'

class FromVisitor(ChildVisitor):
    key = 'from'

class WhereVisitor(ChildVisitor):
    key = 'where'

class OrderByVisitor(ChildVisitor):
    key = 'orderby'

class GroupByVisitor(ChildVisitor):
    key = 'groupby'

class HavingVisitor(ChildVisitor):
    key = 'having'

```

APPENDIX B - TEMPLATES

Sample SQL templates available from SkyServer's help pages¹ 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)

¹<http://skyserver.sdss.org/dr12/en/help/docs/realquery.aspx>

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 splines as sl
  join sparams 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 ($[\text{Fe}/\text{H}] < -3.5$) 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 splines as sl
  join sparams 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
```


PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.