University of Salerno.

Developed for the examination of the semantic web.

A basic methodology in R for querying Pubmed in order to obtain correlations between the search terms and for the organization of resources by topic.

Author:
Francesco Bardozzo

Supervisor:
Prof.ssa Sabrina Senatore

October 2015

Contents

Contents i

List of Figures ii

1 Introduction and PubMed Resources 1
- 1.1 Resources and Query . . . 3
- 1.2 From a PubMed Query to a Corpus . . . 4

2 Text Mining of Queried Abstracts 8
- 2.1 Localisation and Extraction of the Corpus . . . 8
- 2.2 Transformations of the Corpus . . . 9
- 2.3 Evaluation Methods . . . 10
- 2.4 Build an RDF document which describes the correlations . . . 12

3 Probabilistic Topic Modelling for Queried Abstracts 19
- 3.1 Probabilistic Topic Models . . . 20
  - 3.1.1 Preprocessing . . . 20
  - 3.1.2 Fitting the topic models . . . 21
  - 3.1.3 Comparing fitted models . . . 22
    - 3.1.3.1 Evaluation of α for VEM . . . 22
    - 3.1.3.2 Evaluation of the mean entropy . . . 22
  - 3.1.4 Topics and Terms . . . 22
- 3.2 Build an RDF document that describes the Topics . . . 23
- 3.3 Future Developments . . . 27
- 3.4 Conclusions . . . 29

Bibliography 31

List of Figures

1.1 Flowchart of the principal steps and concept that we want to reach with this paper. The flowchart was built with https://creately.com/. Each red shape indicates the number of the chapter where the arguments will be treated. 3

3.1 Diagram of classes of the finalstore generated by the service offered by http://rhizomik.net/html/redefer/rdf2svg-form/ 27
3.2 A piece of the MeSH list retrieved from the MeSH 2015 SPARQL EndPoint 29
3.3 Diagram RDF/RDF Schema, the pink circles are the instances of the classes that are represented in blue. The diagram was generated with the service https://creately.com/ and you can modify the diagram for future developments at this URL: https://creately.com/diagram/hi8vy64d1/FT3998VglOlJUBd0SCOx27UasBg%3D. 30

Chapter 1

Introduction and PubMed Resources

With the increase of the information content coming from different medical databases it is possible to obtain significant correlations between the medical terms that are used to describe the various diseases. The identification of relationships between seemingly unrelated diseases may help to identify new approaches to medical diagnosis and eventually turns out to be possible to organize the information obtained by topic. Thus it is then possible to make a semantic analysis of the information and to this end obtain a more organized and meaningful vision of the information available.

The analysis of correlation between search terms and their positioning by topic can facilitate and speed up the diagnostic work carried out by specialized personnel. In particular, we center our analysis on a case study highly documented in the scientific literature, which describes a relationship between two diseases: diabetes and cancer. In medical terms this type of correlation is defined comorbidity. Furthermore we will spread the space of the terms and their correlation, both between the terms queried and the terms that could be significant when queried and also between the terms queried and those that comes from a topic analysis.

We will perform a query on the medical database PubMed[9] search for the terms cancer and diabetes, PubMed returns in response to the query the list of publications which deal with these topics. The list of papers is not disposed in a particular order. The source of our analysis, the Corpus, consists of the abstracts extracted from the list furnished by PubMed. Basic, after the extraction of abstracts from PubMed, we will seek the association rules between the query terms and some other words most frequent. Subsequently, we will perform a topic analysis trying to identify the same query terms within specific topic and in this way we can organize the complete papers related to the

abstracts. In order to give meaning to the data and made available for searches we will use Semantic Web techniques. Instead we will not go into medical details but we will only discuss the methodological approach and programming for this type of analysis.

For this purpose, we will focus more in technical detail in the use of the R language[13] and Semantic Web techniques applied. Obviously this is not the end point but the starting point for future developments.

The document is organized in 3 chapters, from the extraction of the abstracts to the engineering of an RDF Schema[15][16].

The first chapter provides an introduction to the resources used and available on the web and will describe a clever way to use them in a R framework and using XML technologies. The ultimate goal of the first chapter is precisely to query the PubMed service, get a list of publications related to the query, extract the abstracts of these pubblications that will form the Corpus.

The second chapter provides a description of using the Corpus in order to derive the association rules between terms and also compared to the terms used for queries. From this analysis it will be possible in R to automatically generate an RDF with the corresponding correlations.

The third chapter will show how the same corpus generated by the query performed on PubMed is used to proceed with topic-analysis. From this analysis it will be possible in R to automatically generate a RDF describing the respective topic and abstracts related to it.

The third chapter describes the final RDF that is generated combining the two RDF of the third and second chapter.

The figure 1.1 describes the principal steps and concept that we want to reach with this paper.

To fully understand this document should already have a good skill in using the R programming language and concepts of the Semantic Web (Triple-Store, RDF, SPARQL SPARQL endpoint [17]), XML, Web Services, Text Mining and Probabilistic Topic Models. However we try to make the text straightforward and simple, with particular attention to the details that really matter. So, some sections of this document are R source-driven and will describe: the libraries used, the methods and the results, step by step, in this order.

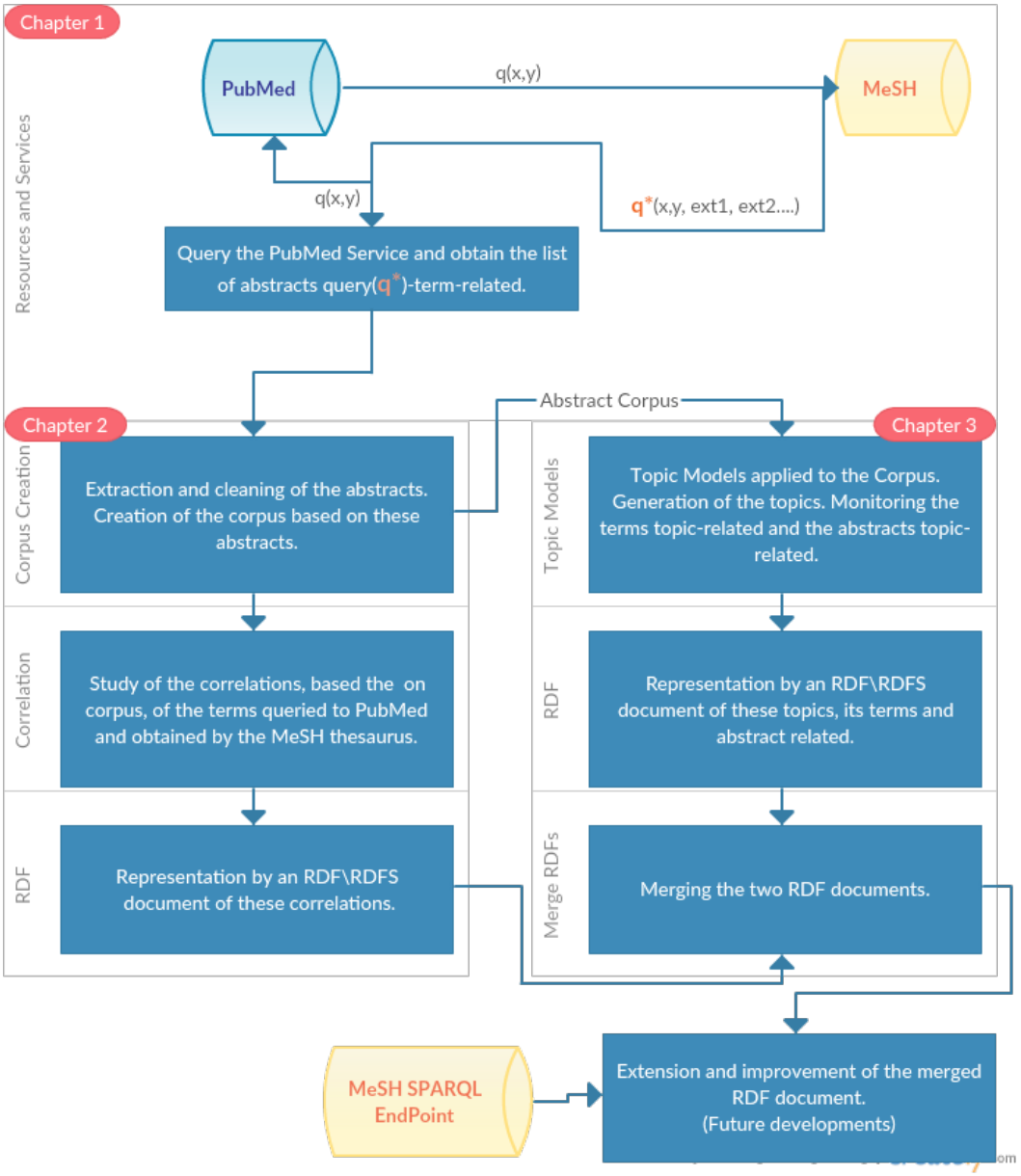

FIGURE 1.1: Flowchart of the principal steps and concept that we want to reach with this paper. The flowchart was built with https://creately.com/. Each red shape indicates the number of the chapter where the arguments will be treated.

1.1 Resources and Query

Our analysis is based on the contents furnished by MEDLINE®, that contains journal citations and abstracts for biomedical literature from around the world. In particular, we use the services provided by PubMed®, this service provides free access to MEDLINE and links to full text articles when possible. PubMed comprises more than 25 million citations for biomedical literature from MEDLINE, life science journals, and online books. Citations may include links to full-text content from PubMed Central and publisher web sites.

What is of interest to us, it is focused with a proven method to extract a certain number of abstracts in which are present the keywords that we asking to this service by a query. The research model of the terms of PubMed isn't naive, for example is possible to combine search terms with Boolean operators (AND, OR, NOT), PubMed applies an AND operator between concepts, e.g., "*vitamin c common cold*" is translated as "vitamin c AND common cold". In this way, *AND* retrieves results that include all the search terms, OR retrieves results that include at least one of the search terms, NOT excludes the retrieval of terms from your search. PubMed processes searches in a left-to-right sequence. Use parentheses to "nest" concepts that should be processed as a unit and then incorporated into the overall search. Therefore we can write a query as the following: $commoncoldAND(vitamincORzinc)$.

There are some "hot" terms that have some predetermined subsets. For example the subject term "cancer" it is extended by cancer-related text words. This is possible because the relationships between these concepts is determined by the medical thesaurus MeSH[10]. The Medical Subject Headings (MeSH®) thesaurus is a controlled vocabulary produced by the National Library of Medicine (NLM) and used for indexing, cataloging, and searching for biomedical and health-related information and documents.

Many synonyms, near-synonyms, and closely related concepts are included as entry terms to help users find the most relevant MeSH descriptor for the concept they are seeking. In NLM's online databases, many terms entered by searchers are automatically mapped to MeSH descriptors to facilitate retrieval of relevant information.

In particular when we approach to PubMed queries by R we are considering automatically also the MeSH Entrez databases, which are designed to assist those searching MEDLINE® / PubMed®; and the UMLS Metathesaurus® with links to many other controlled vocabularies.

In the following sections we will show how to query PubMed through R, making full use of the service, plus getting the concepts extended from MeSH. In the last chapter we try to setup R-SPARQL queries on MeSH 2015 EndPoint to further strengthen our results.

1.2 From a PubMed Query to a Corpus

In this case study we are querying PubMed choosing two subject term "diabetes" and "cancer". For our purposes we need a very large amount of data, being in fact a statistical analysis; a reasonable number of documents that we can get is for example 10000.

Mainly we use these two libraries:

The R library "rentrez" [21] provides an R interface to the NCBI's EUtils API[12] allowing users to search databases like GenBank and PubMed, process the results of those searches and pull data into their R sessions.

The R library "XML"[6] provides a set of tools for parsing and generating XML within R and S-Plus.

The source code is as follows:

```
library(rentrez)

querypub <- c("diabetes", "cancer")
ndoc <- 10000

pubmed_search <- entrez_search( db = "pubmed",
                                term = paste(querypub, collapse = " AND "),
                                retmax = ndoc )
```

The files saved in the variable *pubmed_search* is an XML file and for our purpose should convert it to a list.

```
library(XML)

pubmed_file_list<- xmlToList(pubmed_search$file)
```

One of the fields of this variable, as described above, describes the automatic translation of the query and the terms by the service, in this example is:

```
 library(XML)

 pubmed_file_list<- xmlToList(pubmed_search$file)



 > pubmed_file_list$QueryTranslation

 ("diabetes mellitus"[MeSH Terms] OR ("diabetes"[All Fields] AND
  "mellitus"[All Fields]) OR "diabetes mellitus"[All Fields] OR
  "diabetes"[All Fields] OR "diabetes insipidus"[MeSH Terms] OR
  ("diabetes"[All Fields] AND "insipidus"[All Fields]) OR
  "diabetes insipidus"[All Fields]) AND ("neoplasms"[MeSH Terms] OR
  "neoplasms"[All Fields] OR "cancer"[All Fields])"
```

Leave to the reader the simple exercise to isolate the words differ from "cancer" or "diabetes" to have two main sub-lists: one that contains "cancer" and "neoplasms" and

another containing "diabetes", "mellitus" and "insipidus ". As we mentioned earlier the key terms have been extended directly by the service. We have to track them for the next phase of text mining.

Another important information which gives us the variable *pubmed_search* come from the sub-item *pubmed_search$ids* that is a list of unique IDs representing publications related to the subject terms searched. In fact once obtained IDs papers we can use another method of the library *rentrez* to get the full abstract, date, authors etc .. directly and save it to a text file.

All these text files form the Corpus of our analysis. The abstracts and some other important informations are extracted in XML and after parsed in a plain list of texts. This is useful and allows us to use the plain text in the next phase of text mining.

The following code shows how to extract the abstract, of course, the service could not answer, so the system may give an error, you should handle this error for the method "entrez_fetch", as is done in Java with the construct try / catch.

```
for(j in 0:as.numeric(pubmed_search$retmax))
{

    iddoc <- entrez_fetch(db = "pubmed", id=pubmed_search$ids[j], rettype="xml")

    data <- xmlParse(iddoc)

    list_iddoc<- xmlToList(data)

    ll <- list_iddoc$PubmedArticle$MedlineCitation$Article$Abstract$AbstractText

    ll <-paste( list_iddoc$PubmedArticle$MedlineCitation$Article$Abstract,
                sep="",
                collapse=", "
              )

    #Generate text files, The name of the text file is just the PubMed ID.
    sink(paste(pubmed_search$ids[j],".txt", sep=""))
    cat(ll)
    sink()
}
```

For more informations about the techniques of extraction of data in R from the web services you can read the book "XML and Web Technologies for Data Sciences with R" of Deborah Nolan and Duncan Temple Lang [8].

As we have said we will analyse the corpus containing only abstracts but not we don't calibrate our analysis for example considering the most important words in the title, or date, or the geographic area from which comes the elaborate, or even the publisher, or even considering the network underlying the authors that could influence a writing style than another. But all this information with this simple code could be obtained and used. For now we leave these ideas for future developments.

Chapter 2

Text Mining of Queried Abstracts

The central point of this chapter is focused on the creation of a corpus from a query and then apply text mining monitoring the relationships between query terms. If we want to generate relationships that result from other query terms then the system will create another Corpus. Therefore we consider every single query as generating a small graph of the correlation. The graph of the correlation is extended each time you made a new query. Obviously the nodes used for the merging of the graphs corresponds to the terms that, in this system, are present in more than one query. It is therefore desirable to assign a unique code to each query. In our example, for the query $q_{0001}(diabetes, cancer)$ we associate the code 0001. This type of reasoning about the organization of this informative content could be interpreted and well described by an RDF document.

2.1 Localisation and Extraction of the Corpus

Firstly it is important to select a path for describe the location where there are the text files (corpus) and set the path as a working space directory of R:

```
wscorpus <- "your path /CorpusPubMed/query0001/texts"

setwd(wscorpus)
```

After that we will proceed with the extraction of the corpus, we use the methods of the package *tm*: "Corpus()". Tm is a package built specifically for the operations of text mining [1].

The natural language texts that are present in the Corpus and are written in English, for this reason is reasonable to specify, with the option readerControl, that the texts are written in English("en").

```
library(tm)

corpustmp  <-Corpus(DirSource(wscorpus), readerControl = list(language="en"))
```

2.2 Transformations of the Corpus

A fundamental operation for the tractability of Copora is to clean up the text from potential occurrences of words, numbers or anything that could affect a correct analysis. The library *tm*, with the function *tm_map*, offers a practical interface to apply transformation functions (also denoted as mappings) to Corpus.

Some basic transformations concern the removal of numbers, punctuation, spaces in excess and the transformation of all the words in lower case.

```
corpustmp <- tm_map(corpustmp, removeNumbers)
corpustmp <- tm_map(corpustmp, removePunctuation)
corpustmp <- tm_map(corpustmp, stripWhitespace)
corpustmp <- tm_map(corpustmp, tolower)
```

It turns out to be desirable to remove the words that are recurring in certain contexts most frequently but are not relevant for the analysis to be performed. The set of these words to be removed, such as a filter, are defined stop words. The stop words' set should be built and designed specifically to the problem. In particular, we use the list of stop words recommended by PubMed and available online.

```
corpustmp <- tm_map(corpustmp, removeWords, stopwords(StopWordsFromPubMed))
```

At this point you can use an algorithm for stemming the corpus. As described in the official page, for the English language, the *Porter stemming algorithm (or 'Porter stemmer')* could be done. This algorithm form a process for removing the commoner morphological and linguistic inflections endings from the future vocabulary.

```
corpustmp <- tm_map(corpustmp, stemDocument, language = "english")
```

There are several ways of stemming. Porter stemmer belongs to the class of algorithms of stemming called Truncate Stemmers. These methods are affix removal and therefore are related to removing the suffixes or prefixes of a word. A Truncate (n) stemmer truncate a word at the nth symbol i.e. keep n letters and remove the rest. In this method words shorter than n are kept as it is. The chances of over stemming increases when the word length is small. [5].

For what concerns scientific publications, this algorithm has some limitations. For example if we are interested in numerical quantities, just for a moment we are apart from the fact that we have removed the numerical terms of the text, the stemmer can not identify terms like: "50 ml" of a drug; but this is beyond our purposes.

In order to proceed with our analysis we need to create plain text documents. We use "PlainTextDocument", a simple option of *t*he method tm_map, as follow:

```
corpus_clean <- tm_map(corpustmp, PlainTextDocument)
```

2.3 Evaluation Methods

We can create a Document Term Matrix and apply the evaluation methods for obtain the associations rules. The association rules between the terms represents in some cases the simple correlation between terms, there are different types of associations rules, we use the one given from the library *tm* and provided by the standard library of R for these types of data, that is the Pearson correlation.

```
adtm <-DocumentTermMatrix(corpus_clean,
control = list(bounds = list(local = c(1,Inf))))
```

The function *DocumentTermMatrix* provides some local options which are evaluated for each document and global options which are evaluated once for the constructed matrix.

Available global options are for example *bounds* that is a list with a tag global whose value must be an integer vector of length 2. Terms that appear in less documents than the lower bound *global*[1] or in more documents than the upper bound *global*[2] are discarded. We are searching for the terms that may be present even once in an abstract but of central importance, for this reason we put: list(global = c(1, Inf)) respectively the lower and the upper bound, which means: every term will be considered, also if it occurs only one time.

An important local option is *weighting*, that is capable of handling a TermDocumentMatrix. The default setting of the method DocumentTermMatrix is weightTf and we use

this one. There are different types of weighting, the use of one with respect to another depends to the model and the type of analysis performed and on the transformations of the documents. Worth going into here.

Available weighting functions shipped with the tm package are weightTf, weightTfIdf, weightBin, and weightSMART. In weightTf the weighting of the matrix is applied using weights that corresponds to the frequency of the terms.

$\text{weightTf(t)} = \frac{N_t}{Tot_{t's}}$

Where N_t is Number of times term t appears in a document and $Tot_{t's}$ is the total number of terms in the document.

Otherwise is possible to utilize weightTfIdf, which measures how important a term is. While computing weightTf, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$\text{weightTfIdf(t)} = \frac{N_{tot}}{N_{t_{doc}}}$

Where N_{tot} is the total number of documents and N_{doc} is the number of documents with the term t in it.

We don't use this option because we suppose that all the terms that are not "important" are removed from our corpus in the precedent transformations.

The other options are weightBin that binary weight a term-document matrix and weightSMART that weight a term-document matrix according to a combination of weights specified in SMART notation, for more information about this method we suggest to read Manning et all. [7]

At this point it is possible to remove the sparse terms, deciding the sparse value.

```
adtm <- removeSparseTerms(adtm, 0.01)
```

The function *removeSparseTerms* returns a term-document matrix where those terms which have at least a sparse percentage of empty (i.e., terms occurring 0 times in a document) elements are removed. In this example the resulting matrix contains only terms with a sparse factor of less than 0.01.

Finally, we can get the association rules, we do not take all the possible terms of the vocabulary, but only those query-related; example those included in $q_{001}(cancer, diabetes)$ and those extended by the service PubMed, ie "neoplasms", "mellitus", "insipidus".

Therefore the queries will be: $q_{001ext_1}(cancer, neoplasms)$ and $q_{001ext_2}(diabetes, insipidus, mellitus)$, could be considered separately in the searching of correlations for the simple purpose of facilitating the management of data.

```
ass1<-findAssocs(adtm, q0001ext_1, 0.2)
ass2<-findAssocs(adtm, q0001ext_2, 0.2)
```

The option *corlimit* of 0.2 allows to include the terms correlated at least of a correlation value not lower of 0.2.

```
> ass1["cancer"]

breast       prostate     cancers  colorectal   diabetes
  0.33           0.28        0.25        0.24       0.24

> ass1["neoplasms"]
  kobe          metachronous acromegalyassociated       extrapituitary
  0.34                  0.32                  0.29                 0.26
  cteaching        edmonton                   fnh                 fnhs
  0.24                  0.24                  0.24                 0.24
  hcas        hypervascular             peliosis          synchronous
  0.24                0.24                  0.24                 0.24
  tumourlike
  0.24
```

The correlations between "cancer" and the other words in the corpus it is, for example, 0.33 for the keyword "breast" and 0.24 for the word "diabetes".

2.4 Build an RDF document which describes the correlations

In order to organize and to make conceptually interpretable the information obtained from the correlations computed, we need to write a particular document. Basically we are interested to describe, for example, if the correlation obtained is an integer or a float, but, moreover, we are interested in the meaning of this correlation in a context where is related to a specific query. Moreover the terms are more generic that the correlation obtained, therefore we can use these terms also in other queries, and we need to underline this concept.

For these purposes we can write an RDF document, that is a metadata data model with particular specifications furnished by the W3C. In the last years it has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. It is also used in knowledge management applications.

To do that, we use a bridge between R and Apache Jena. We will not go into detail of Apache Jena and please refer to the online documentation.[4] What we need to know is that through this bridge we can build a proper RDF. To install the necessary libraries should take a few more steps. We need to use the libraries: rJava[14], devtools[19], rrdf and rrdflibs[20].

```
#R and Web Semantic - Basic Libraries
    install.packages("rJava")
    if (Sys.getenv("JAVA_HOME")!="")
        Sys.setenv(JAVA_HOME="")
    library(rJava)

    install.packages("devtools")
    require(devtools)
    install_github("rrdf", "egonw", subdir="rrdflibs")
    install_github("rrdf", "egonw", subdir="rrdf", build_vignettes = FALSE)

    library(rrdf)
    library(rrdflibs)
```

We interpret the terms identified as nodes, which are connected by a specific number of links and each link has its own weight that corresponds precisely to the correlation of terms. It is not difficult to explain these concepts for this reason, firstly, we can use TURTLE[18] to give an idea. After we show the R source to construct an RDF automatically for each query that we perform.

In this case we are showing the 6-th correlation of the query 0001 between the two nodes, the terms "diabetes" and "cancer".

```
@prefix q:     <http://query.org/> .
@prefix qq:    <http://query.org/query> .
@prefix qe:    <http://query.org/query/0001/edge/> .
@prefix qn:    <http://query.org/node/> .

<http://query.org/query/0001/edge/edge#23>
 a       q:edge ;
 qn:node <http://query.org/node/node#diabetes>,<http://query.org/node/node#cancer>;
 qe:correlation "0.24"^^<http://www.w3.org/2001/XMLSchema#float> .

<http://query.org/node/node#cancer>
 a       q:node .

<http://query.org/node/node#diabtes>
 a       q:node .
```

In another query where the terms appear to be still present, i.e. "diabetes" and "cancer", could change the value of the correlation but do not change the words "diabetes" and "cancer". So even in a graph the nodes, though there is or not an edge, do not change. To this end the prefixes associated to the nodes are more generic than the one used to

describe a specific correlation in its "edge-context". Obviously for this purpose we have created some classes which if instantiated are cohesive and works with minimal coupling.

With the methods provided by *rrdf* we can add prefixes and triple to a triple store, describing concept and relationship. Obviously at the beginning we should create a triple store in this way:

```
store = new.rdf()
```

At this point it is appropriate to add the prefixes, with the method *add.prefix()* to the store, pointing out that each query creates a context different from each other, it is appropriate, as mentioned earlier, to create a unique identifier of the query, ie qcode = 0001.

We can create a method like this one that returns a triple store with only the prefixes:

```
qcode = 0001
addRDFPrefix <- function(qcode, store)
{
    add.prefix(store, "qe", paste("http://query.org/query/",qcode ,"/edge/",sep=""))
    add.prefix(store, "q", "http://query.org/")
    add.prefix(store, "qq", "http://query.org/query/")
    add.prefix(store, "qn", "http://query.org/node/")
    return(store)
}
```

Some other prefixes that comes from the ontology, RDF and RDF Schema are added automatically. Apache Jena recognize in automatic the prefixes, if are already present the triples for these ones, even if are not specified by a particular name, and assigns a standard prefix like "j_1", "j_2" etc... For this reason, even if at the design stage it is not clear which prefixes use, you may decide at a later time.

Then, for each correlation obtained, we create an edge and two nodes, as well we must add the concepts for have not a loss of generality and for avoid confusion in identifying what is a node and what is an edge. If it is clear for us in this document that an edge is a correlation query-related and a node is a term in correlation with another one, this is not understandable by a machine or by someone who does not read this paper.

In particular we use two methods that add triples to the store, *add.triple* and *add.data.triple*. The first one method adds binary-relation of resources to a triple store, the latter adds a triple that associate a particular type of data to a resource.

The way of using these methods is easy in R, the way of using these methods smartly is not immediate for the people who will approach for the first time to the Semantic Web.

At the beginning we proceed with the addition of three classes, a class that describes the Nodes, a class that describes the concept of query and a class of the edges, in the following manner:

```
qcode = 0001
  #Query Class
  add.triple(store,
             subject = paste("http://query.org/query",sep=""),
             predicate ="http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
             obj="http://www.w3.org/2000/01/rdf-schema#Class")

  #Node Class
  add.triple(store,
             subject = paste("http://query.org/node" ,sep=""),
             predicate ="http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
             obj="http://www.w3.org/2000/01/rdf-schema#Class")
  #Edge Class
  add.triple(store,
             subject = paste("http://query.org/edge" ,sep=""),
             predicate ="http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
             obj="http://www.w3.org/2000/01/rdf-schema#Class")
```

Obviously these classes are related together in a maximal cohesive way. The first important instance is derived from the class Query. Its instance identify the executed query, with a the code 0001 as described previously.

We can give an Query instance to the triple store through the construct *rdf:type* as follows:

```
#Query 0001 Instance
qcode = 0001
add.triple(store,
           subject = paste("http://query.org/query/", qcode, "/", sep=""),
         predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
               obj = "http://query.org/query")
```

There aren't in this engineering rdfs:subClassOf, so there are not particular needs of coupling some classes but there is something that doesn't work, maybe, now, a machine can understand what we're talking about, but the unsuspecting user might still be very confusing.

For these data models is a good idea to comment the classes to have a more human feeling, also in these case we combine the RDF and of RDF Schema syntax.

Note that RDF allows us to define a hierarchy of properties just as in the case of classes, therefore we could have a "Property" and "subPropertyOf" (as classes obviously transitive) and these instances could be commented.

For our purposes we could use only the property of the comment associated as follows:

```
qcode = 0001

  add.data.triple(store,
              subject = "http://query.org/query",
              predicate ="http://www.w3.org/1999/02/22-rdf-syntax-ns#comment",
              data="The class Query, its instances are the executed queries.
                    For these reasons this class is cohesive with the classes
                    Edge and Node. ")

  add.data.triple(store,
              subject = "http://query.org/node",
              predicate ="http://www.w3.org/1999/02/22-rdf-syntax-ns#comment",
              data="The class Node. Its instances are the terms used by the
                    instances of the class Query, for the study of the Topics
                    both for the study of the terms correlations (class Edge).")

  add.data.triple(store,
              subject = "http://query.org/edge",
              predicate ="http://www.w3.org/1999/02/22-rdf-syntax-ns#comment",
              data="The class Edge, its istances represents the correlations
                    between two Nodes. This class is cohesive with the RDF Bag
                    container. ")
```

There are some classes that is recognized to be auto-descriptive, because there is a clear definition in the RDF syntax, this is the case of a Bag instance, that is a specific container. A better study on the engineering of the RDF could lead us to use an instance from a RDF Class, that represents a list of resources, a sort of container where the instances of the class Edge are listed together. In this case the container of edges for each n-th query is instantiated as follows:

```
qcode = 0001
# This container maintains a list of instances of class Edges, with a correlation
# value and describing the two nodes that are connected from the i-th Edge.
add.triple(store,
         subject = paste("http://query.org/query/",qcode,"/edge/bag", sep =""),
       predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
             obj = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag")
```

Each instance of the bag is associated to the bag by some special properties. The form of these is as follows: "_n", where n is a progressive natural number from 0 to the max cardinality of the set of correlations counted by the variable *count*. These properties are attached to the container as described in the following source and generally provide us an easy way to organize the relationships between terms and their respective resources nested.

Note that there are two *for* one for iterate the list of the association rules (variable: ass) of the term queried and another one for iterate the terms related to the term queried.

There are some comments that help in the reading of the code.

```
count = 0
for(i in 1:length(ass))
{
 for (j in 1:length(ass[[i]]))
 {
  if(length(ass[[i]])!=0)
  {
   count += 1



   #BAG-EDGE Section ########################################################

   #Instance of the class Edge
   add.triple(store,
              subject = paste("http://query.org/query/0001/edge/edge",
                              count, sep="#"),
            predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
                  obj = "http://query.org/edge")

   #Add an instance of the class Edge to the container Bag.
   add.triple(store,
              subject = "http://query.org/query/0001/edge/bag",
            predicate = paste("http://www.w3.org/1999/02/22-rdf-syntax-ns#",
                              count, sep="_"),
                  obj = paste("http://query.org/query/0001/edge/edge", count,
                         sep="#"))



   #Creation of the First and Second Node ###################################

   #Instance of the class Node for the i-th term. First Node.
   add.triple(store,
              subject = paste("http://query.org/node/node",
                              gsub(" ", "_", names(ass[[i]][j])), sep = "#"),
            predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
                  obj = "http://query.org/node")

    #Add the First Node to the i-th instance of the class Edge.
    add.triple(store,
               subject = paste("http://query.org/query/0001/edge/edge",
                           count, sep="#"),
             predicate = "http://query.org/node/node",
                   obj = paste("http://query.org/node/node",
                               gsub(" ", "_", names(ass[[i]][j])),
                               sep = "#"))

    #Instance of the class Node for the i-th term. Second Node.
    add.triple(store,
               subject = paste("http://query.org/node/node",
                               gsub(" ", "_", names(ass)[i]), sep = "#"),
             predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
                   obj = "http://query.org/node")
```

```
  #Add the Second Node to the i-th instance of the class Edge.
  add.triple(store,
                subject = paste("http://query.org/query/0001/edge/edge",
                             count, sep="#"),
              predicate = "http://query.org/node/node",
                    obj = paste("http://query.org/node/node",
                                gsub(" ", "_", names(ass)[i]), sep = "#"))




# Correlation between Nodes ##############################################

# Add the data type that describes the value of correlation between
# the two nodes and the weight of the i-th instance of the class edge.

add.data.triple(store,
                subject = paste("http://query.org/query/0001/edge/edge",
                                 count, sep="#"),
              predicate = "http://query.org/query/0001/edge/correlation",
                   data =  gsub(" ", "_", ass[[i]][[j]]),
                   type =  "float")



      }
   } }
```

In the figure 3.3 is represented both the RDF Schema and the RDF.

Chapter 3

Probabilistic Topic Modelling for Queried Abstracts

In the sections 1.2 and 2.1 we have explained the way for the extraction and setup of a Corpus of abstracts from PubMed queried terms and as we have explained in the introduction we are searching a way of organizing the abstracts query-related in their hidden thematic structure. Probabilistic Topic Models if used with a correct setting can help us in this purpose and in a semi-automatic way. By these models we have a disposition of each abstract related to their relevant terms in a particular topic. Obviously we are interested in the terms that are present both in the i-th topic both in one of the terms queried from PubMed. It will be possible to generate another RDF document that is relatable to RDF document present in the section 2.4. One important step in this analysis is to identify the abstracts with a unique identifier, this is possible in an easy way because we have named the documents of the corpus with their PubMed id, but this is not enough. When we have generated the Document Term Matrix, the documents have not assigned a specific identifier. It should be done by this command:

```
adtm$dimnames$Docs <- pubmed_search$ids
```

In particular we can see the url of the service and the ID of each abstract as a web resource well described by an URL. For this reason, each term, which will be interpreted as a node, will be associated to the resources of the abstracts that represent them more according to the topic analysis. For example an abstract of our corpus has the name: *XXXYYYYY.txt* and is related to the term "cancer", then the resource associated to the node "cancer" is the URL:

http://www.ncbi.nlm.nih.gov/pubmed/XXXYYYY

This type of resource is related to the term "cancer" always in the context of the query performed. The URI it is: http://www.ncbi.nlm.nih.gov/pubmed/. Also in this case, if we have done another query and are extracted other documents, if these documents are linked to the same term queried before then these will be added as resources of another topic analysis but related to same resource. As described in the chapter 2 this is possible by a correct engineering of the RDF document.

3.1 Probabilistic Topic Models

To perform this kind of analysis we use the R library *topicmodels*[3]. The central problem of these models is to assume or not that the topics are correlated or uncorrelated each other. The latent Dirichlet allocation (LDA) model is a Bayesian mixture model for discrete data where topics are assumed to be uncorrelated. The correlated topics model (CTM) is an extension of the LDA model where correlations between topics are allowed. [2] We will not describe the details of these models for which you can refer to the online documentation but we will show an example of use applied to our purposes.

3.1.1 Preprocessing

As explained before we can use the DocumentTermMatrix method for the extraction of terms, also bounding the terms presence in each document, in this case we apply the bound of 4 as minimum presence in a document of a specific term.

```
adtm <-DocumentTermMatrix(corpus_clean,
                          control = list(bounds = list(local = c(4,Inf))))

adtm$dimnames$Docs<-pubmed_search$ids
```

In this way we are reducing the vocabulary to only 7622 terms on 10000 abstracts. Other types of corpus preprocessing are already applied as described in the section 2.2 and saved in the variable "corpus_clean". To the end of a correct vocabulary selection is not enough to apply the transformations shown so far but we should choose if we are considering a term frequency document matrix or some other types of matrices as described in the Evaluation Methods, section 2.3. Here we can consider the min term frequency-inverse document frequency (tf-idf) to select the vocabulary as follows:

```
require(slam)
summary(col_sums(adtm))

term_tfidf <- tapply(adtm$v/row_sums(adtm)[adtm$i], adtm$j, mean) *
              log2(nDocs(adtm)/col_sums(adtm > 0))
```

If we get the summary of term_tfidf we have:

```
summary(term_tfidf)
> Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
> 0.2076  1.4410  2.0980  2.7940  3.3220 13.2900
```

The min term f-idf is 0.2076 and we can select the vocabulary for fit the topic models:

```
adtm <- adtm[,term_tfidf >= 0.2076]
adtm <- adtm[row_sums(adtm) > 0,]
```

3.1.2 Fitting the topic models

As mentioned before we don't describe the models used, but is important to know that we apply a maximum likelihood (ML) estimation of the LDA model. The log-likelihood of the data, i.e., the sum over the log-likelihoods of all documents, is maximized with respect to the canonic model parameters α and β. The method used for fitting the LDA models is the variational expectation-maximization (VEM) algorithm.

As described in the paper topicmodels we fit an LDA model with an a-priori selected number of topics using: VEM with α estimated, VEM with α fixed and Gibbs sampling with a burn-in of 1000 iterations and recording every 100th iterations for 1000 iterations. The initial α is set to the default value. In addition a correlated topic model (CTM) is fitted using VEM estimation.

We ran a k-fold cross-validation to decide the number of topics to be used. We selected k = 10 because the query terms, with k = 10, appears to be well separated and correctly topic-related also with the other terms that are present in the vocabulary.

```
k <- 10
SEED <- 10000
PubMedTM <-
           list(VEM =       LDA(adtm, k = k,
                                control = list(seed = SEED)),
                VEM_fixed = LDA(adtm, k = k,
                                control = list(estimate.alpha = FALSE,
                                seed = SEED)),
                Gibbs =     LDA(adtm, k = k, method = "Gibbs",
                                control = list(seed = SEED,
                                               burnin = 2000,
                                               thin = 100,
                                               iter = 2000)),
                 CTM =       CTM(adtm, k = k,
                                 control = list(seed = SEED,
                                 var = list(tol = 10^-4),
                                 em = list(tol = 10^-3)))
                 )
```

3.1.3 Comparing fitted models

In a certain sense we would like to understand the goodness of the fitted method studying the percentage of documents assigned with a high probability to one single topic.

The most likely topic for each document, that is obtained by LDA model (VEM) with α estimated under the assumption that the topics, are uncorrelated.

Below we show two methods for evaluating the fitness of the models.

3.1.3.1 Evaluation of α for VEM

A lower α indicate that is higher the percentage of documents which are assigned to one single topic with an high probability.

```
sapply(PubMedTM[1:2], slot, "alpha")
> VEM          VEM_fixed
  0.05481134   5.00000000
```

3.1.3.2 Evaluation of the mean entropy

Is possible to obtain the mean entropy for each fitted model over abstracts, higher values indicate that the topic distributions are more evenly spread over the topics.

```
sapply(PubMedTM, function(x)
                  mean(apply(posterior(x)$topics, 1,
                  function(z) - sum(z * log(z))))
      )

VEM         VEM_fixed  Gibbs      CTM
0.7797416   2.1537742  2.1424352  0.7583069
```

3.1.4 Topics and Terms

We have chosen VEM, in the previous section we explain why and now we show a way for get the VEM data about terms topic-related and topics.

The method topics() give us the complete list of abstracts with the number of the most likely associated topic.

```
Topic <- topics(PubMedTM[["VEM"]], 1)
```

Instead the method terms() give us a list of terms more related to the topic under consideration. In this case we are taking the 15 terms most likely.

```
Terms <- terms(PubMedTM[["VEM"]], 15)
```

The variables Topic and Terms are used in the next section for adding information and resources to the RDF document.

3.2 Build an RDF document that describes the Topics

In the section 2.4 we have well described the use of the R libraries and the methods for the engineering of an RDF document in R. Substantially the concepts and the sources behind the creation of this RDF document is the same. So we will be more synthetic on certain aspects. We have to form the RDF document formalizing the concepts of terms Topic-related, abstracts Topic-related and Topics. In a certain sense we think to the i-th topic as a double list where there are the terms, prefix "qn" as in precedence, and abstracts, prefix "qt" (these type of resources are described in the introduction of this chapter). A little piece of the schema is showed as follow in TURTLE:

```
@prefix q:     <http://query.org/> .
@prefix qt:    <http://query.org/query/0001/topic/> .
@prefix abs:   <http://www.ncbi.nlm.nih.gov/pubmed/> .
@prefix qn:    <http://query.org/node/> .

<http://query.org/query/0001/topic/topic#3>
  a          q:topic ;
  qn:node    <http://query.org/node/node#dna>, <http://query.org/node/node#hcv>[.];
  qt:abspm   abs:26395490 [.] .

<http://query.org/node/node#dna>
  a        q:node .

<http://query.org/node/node#hcv>
  a        q:node .

abs:26394722   a   abs: .

# The symbol [.] means that there are some resources that are not represented.
```

Remember that if we want to merge two RDF documents together we need to use the same prefixes between Classes and between instances that we want to merge. The method for adding all the prefixes is as follow:

```
addRDFPrefixTopic <- function(qcode, storet)
{add.prefix(storet, "q", "http://query.org/")
 add.prefix(storet, "qn", "http://query.org/node/")
 add.prefix(storet, "qe", paste("http://query.org/query/",qcode ,"/edge/",sep=""))
 add.prefix(storet, "qt", paste("http://query.org/query/",qcode ,"/topic/",sep=""))
 add.prefix(storet, "abs", paste("http://www.ncbi.nlm.nih.gov/pubmed/")) }
```

The new Classes and their properties as human-like comments are the following:

```
#Topic Class
add.triple(storet,
           subject = "http://query.org/topic",
         predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
               obj = "http://www.w3.org/2000/01/rdf-schema#Class")

add.data.triple(storet,
           subject = "http://query.org/topic",
         predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#comment",
              data = "Resource that describes the Topic Models, executed with
                      LDA - VEM. The class for each instance of Topic presents
                      15 terms most important for a specific Topic and the
                      abstracts linked.")
  #-------------------------------------------------------------------------#
  # PubMed Abstract
  add.triple(storet,
             subject = "http://www.ncbi.nlm.nih.gov/pubmed/",
           predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
                 obj = "http://www.w3.org/2000/01/rdf-schema#Class")
  # PubMed Abstract Comment Property
   add.data.triple(storet,
            subject  = "http://www.ncbi.nlm.nih.gov/pubmed/",
           predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#comment",
                data = "This resource is an external resource that refers to the
                        service of pubmed. We use this resource for to link the
                        abstracts related to a particular topic by their unique
                        identifier.")

   #-------------------------------------------------------------------------#
```

The use of Bag is the same of the section 2.4, the source code accordingly to that is the following:

```
add.triple(storet,
           subject = "http://query.org/query/0001/topic/bag",
         predicate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
               obj = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag")
```

Attaching to the Bag the informations iteratively could be structured in a more modular way of programming, giving a variable for each URI (topicURI, topicelURI, rdfURI, qnURI, qnelURI...).

In this piece of source we are showing the way of adding to the Bag the most relevant 15 terms for each topic. Firstly we create an instance of the class Topic then the i-th Topic instantiated becomes a member of the Bag, the terms are resources of the i-th topic and in particular are instances of the class Node.

```
qcode = 0001

topicURI   = paste("http://query.org/query/", qcode, "/topic/", sep="")
topicelURI = paste("http://query.org/query/", qcode, "/topic/topic", sep="")
rdfURI     = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
qnURI      = "http://query.org/node"
qnelURI    = "http://query.org/node/node"


for(i in 1:length(Terms[1,]))
{


   #The i-th Topic is an instance of the class Topic.
   add.triple(storet,
              subject = paste(topicelURI, i, sep="#"),
            predicate = paste(rdfURI, "type", sep="#")
                  obj = paste("http://query.org/topic", sep=""))

   #The i-thm Topic becomes a member of the Bag.
   add.triple(storet,
              subject = paste(topicURI, "bag", sep =""),
            predicate = paste(rdfURI, i, sep="_"),
                  obj = paste(topicelURI, i, sep="#"))


    for (j in 1:length(Terms[,1]))
    {

    #The j-th term of the i-th Topic is instantiated.
    add.triple(storet,
              subject = paste(qnelURI, gsub(" ", "_", Terms[[j,i]]), sep = "#"),
            predicate = paste(rdfURI, "type", sep="#")
                  obj = "http://query.org/node")


    #The j-th term is linked to the i-th Topic
    add.triple(storet,
              subject = paste(topicelURI, i, sep="#"),
            predicate = qnURI,
                  obj = paste(qnelURI, gsub(" ", "_", Terms[[j,i]]), sep = "#"))

     }


 }
```

In this method we are adding to the i-th topic iteratively the resources that links the topics to the abstracts. Remember that these are the abstracts present in the corpus.

```
qcode = 0001

abstURI = paste("http://query.org/query/",qcode,"/topic/abspm", sep="")
absURI = "http://www.ncbi.nlm.nih.gov/pubmed/"

for(i in 1:length(Terms[1,]))
{
    pubidst <- names(Topic[Topic==i])

    for (j in 1:length(Topic[Topic==i]))
    {

        #Instance of the j-th abstract related to the i-th Topic.
        add.triple(storet,
                   subject = paste(absURI, pubidst[j], sep=""),
                 predicate = paste(rdfURI, "type", sep="#"),
                       obj = absURI )


        #Links the abstract to the i-th Topic.
        add.triple(storet,
                     subject = paste(topicelURI, i, sep="#"),
                   predicate = abstURI,
                         obj = paste( absURI, pubidst[j], sep=""))
    }



}
```

With the method *combine.rdf* we can combine two triple store and obtain our final triple store. The syntax is very simple, we have called the store of the section 2.4: "store", and the triple store of this section: "storet". Sorry for the lack of imagination... we will call the merged store: "finalstore". At this point we can save the merged RDF document in different formats: RDF/XML, RDF/XML-ABBREV, N3, TURTLE, and N-TRIPLE.

The R source is the following:

```
finalstore = combine.rdf(store, storet)

save.rdf(finalstore, "finalStoreTURTLEAbs.xml", "TURTLE")

save.rdf(finalstore, "finalStoreRDFAbs.xml", "RDF/XML")
```

The figure 3.3 shows the complete RDF/RDF Schema of the merged RDF documents. The circles in blue indicates the classes of the Schema and the pinks one the instances.

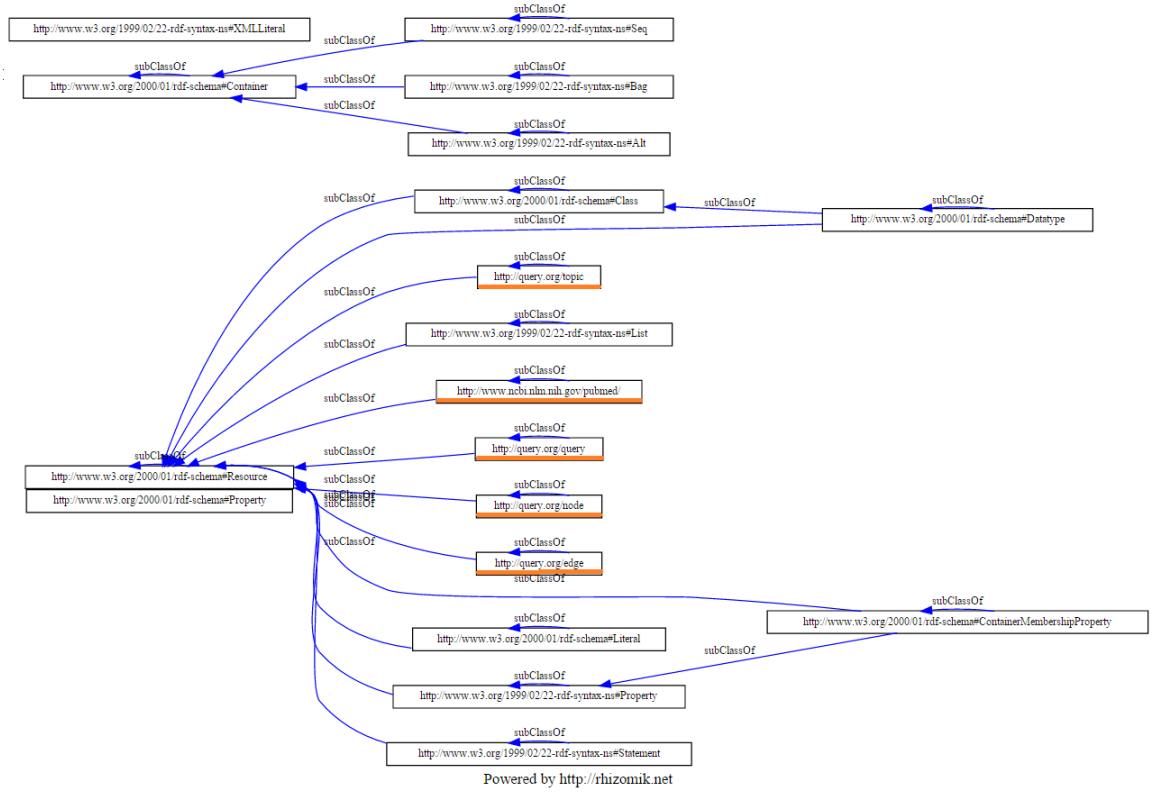

FIGURE 3.1: Diagram of classes of the finalstore generated by the service offered by http://rhizomik.net/html/redefer/rdf2svg-form/

Note that i.e. in this figure the same instance of the node for the term cancer is the same for the two Bags.

3.3 Future Developments

Some ideas about future developments are described along the document. Here we want only to present a way of using MeSH not basing only our information retrieval on PubMed services but also using a SPARQL query on the SPARQL EndPoint MeSH 2015 in R language.

The difficulties of using this endpoint is the configuration of the parameters of the query, we will show in the source a working example. For deep understanding of the parameters there is a wide documentation online[11].

Suppose that we want to search the term "cancer" because we want to extend our vocabulary or we want to refine our RDF document by a better definition of the concept of term.

For this purpose we use the R library SPARQL and we write a function like the following:

```
queryMeSH15 <- function(vect){

       require(SPARQL)

       # Build the SPARQL Query

       rex = paste( "REGEX(?dName,'" ,vect[1], "','i') ||
                     REGEX(?cName,'", vect[1] , "','i')", sep="")

       qr= paste("
                  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
                  PREFIX owl: <http://www.w3.org/2002/07/owl#>
                  PREFIX meshv: <http://id.nlm.nih.gov/mesh/vocab#>
                  PREFIX mesh: <http://id.nlm.nih.gov/mesh/>
                  PREFIX mesh2015: <http://id.nlm.nih.gov/mesh/2015/>

                  SELECT ?d ?dName ?c ?cName
                  FROM <http://id.nlm.nih.gov/mesh>
                  WHERE {
                         ?d a meshv:Descriptor .
                         ?d meshv:concept ?c .
                         ?d rdfs:label ?dName .
                         ?c rdfs:label ?cName
                  FILTER(",rex,")
                  }",
                  sep = "")



        #Execute the SPARQL Query on MeSH 2015 SPARQL EndPoint
        toret <- SPARQL(  url = "http://id.nlm.nih.gov/mesh/sparql/",
                          query = qr,
                          # Mesh 2015 SPARQL EndPoint Parameters
                          extra =  list(
                                        format="HTML",
                                        inference="TRUE",
                                        year="current",
                                        limit="20000",
                                        offset="0"
                                        )
                        )


        return(toret)
    }



d<-queryMeSH15("cancer")
```

d	dName	c	cName
<http://id.nlm.nih.gov/mesh/D005134>	"Eye Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0332431>	"Cancer of Eye"@en
<http://id.nlm.nih.gov/mesh/D004379>	"Duodenal Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0445961>	"Duodenal Cancer"@en
<http://id.nlm.nih.gov/mesh/D009369>	"Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0014584>	"Cancer"@en
<http://id.nlm.nih.gov/mesh/D013736>	"Testicular Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333446>	"Cancer of Testis"@en
<http://id.nlm.nih.gov/mesh/D001005>	"Anus Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0332111>	"Anal Cancer"@en
<http://id.nlm.nih.gov/mesh/D018567>	"Breast Neoplasms, Male"@en	<http://id.nlm.nih.gov/mesh/M0027831>	"Male Breast Cancer"@en
<http://id.nlm.nih.gov/mesh/D001063>	"Appendiceal Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0332117>	"Appendiceal Cancer"@en
<http://id.nlm.nih.gov/mesh/D001661>	"Biliary Tract Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0332162>	"Biliary Tract Cancer"@en
<http://id.nlm.nih.gov/mesh/D002430>	"Cecal Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0332204>	"Cancer of Cecum"@en
<http://id.nlm.nih.gov/mesh/D010478>	"Chemotherapy, Cancer, Regional Perfusion"@en	<http://id.nlm.nih.gov/mesh/M0016277>	"Chemotherapy, Cancer, Regiona
<http://id.nlm.nih.gov/mesh/D014571>	"Urologic Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333649>	"Cancer of Urinary Tract"@en
<http://id.nlm.nih.gov/mesh/D009062>	"Mouth Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0337712>	"Cancer of Mouth"@en
<http://id.nlm.nih.gov/mesh/D009959>	"Oropharyngeal Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333159>	"Cancer of Oropharnyx"@en
<http://id.nlm.nih.gov/mesh/D014565>	"Urogenital Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333641>	"Genitourinary Cancer"@en
<http://id.nlm.nih.gov/mesh/D010386>	"Pelvic Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333237>	"Cancer of Pelvis"@en
<http://id.nlm.nih.gov/mesh/D010412>	"Penile Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333240>	"Cancer of Penis"@en
<http://id.nlm.nih.gov/mesh/D052138>	"Genes, Neoplasm"@en	<http://id.nlm.nih.gov/mesh/M0483363>	"Cancer Genes"@en
<http://id.nlm.nih.gov/mesh/D014625>	"Vaginal Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0022484>	"Vaginal Cancer"@en
<http://id.nlm.nih.gov/mesh/D014516>	"Ureteral Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333627>	"Cancer of Ureter"@en
<http://id.nlm.nih.gov/mesh/D014523>	"Urethral Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333639>	"Cancer of Urethra"@en
<http://id.nlm.nih.gov/mesh/D014846>	"Vulvar Neoplasms"@en	<http://id.nlm.nih.gov/mesh/M0333665>	"Cancer of Vulva"@en
<http://id.nlm.nih.gov/mesh/D014411>	"Neoplastic Stem Cells"@en	<http://id.nlm.nih.gov/mesh/M0503692>	"Cancer Stem Cells"@en

FIGURE 3.2: A piece of the MeSH list retrieved from the MeSH 2015 SPARQL EndPoint

A part of the results of this query is showed in the figure 3.2. The field dName is name of the disease and is showed the relationship between its canonical name(cName). Is showed also the resource associated.

3.4 Conclusions

We applied a technique of information retrieval to extract information and abstracts from PubMed. Therefore we started with a corpus generated from a list of abstract provided as a result of a query run on PubMed service. We performed text mining analysis and a topic analysis, then we created an RDF document that summed up the correlations of the terms queried, the topics of the corpus and the underlying concepts about this query-related analysis. We did everything using the language R and services provided by PubMed. We have showed a new way of querying the endpoint MeSH 2015 SPARQL EndPoint [11] for future developments.

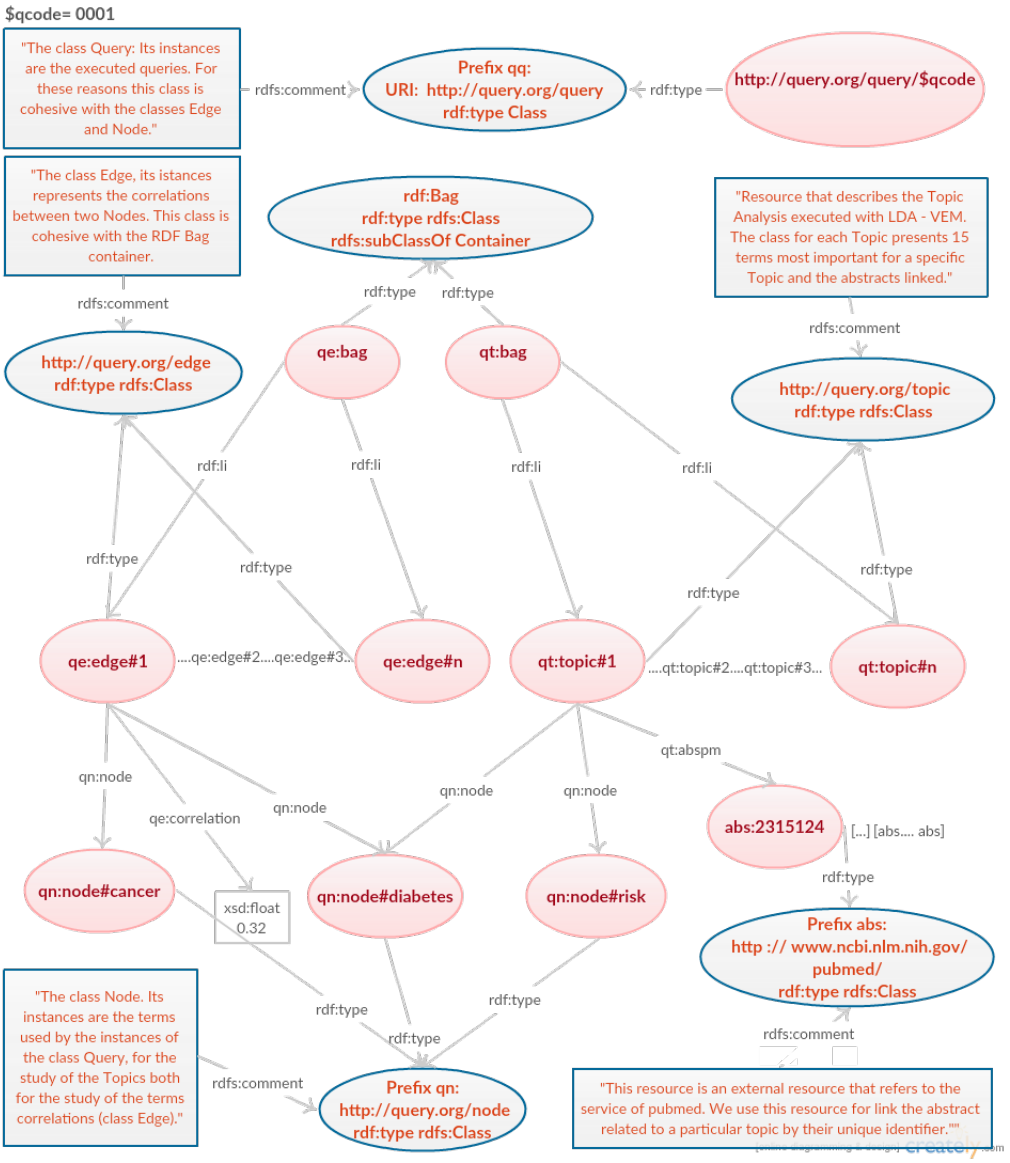

FIGURE 3.3: Diagram RDF/RDF Schema, the pink circles are the instances of the classes that are represented in blue. The diagram was generated with the service https://creately.com/ and you can modify the diagram for future developments at this URL: https://creately.com/diagram/hi8vy64d1/FT3998VglOlJUBd0SCOx27UasBg%3D.

Bibliography

[1] I. Feinerer. Text mining package. URL https://cran.r-project.org/web/packages/tm/tm.pdf.

[2] B. Grün and K. Hornik. An r package for fitting topic models. . URL http://www.jstatsoft.org/article/view/v040i13.

[3] B. Grün and K. Hornik. Topic models. . URL https://cran.r-project.org/web/packages/topicmodels/topicmodels.pdf.

[4] JavaCorp. Jena ontology api. URL https://jena.apache.org/documentation/ontology/.

[5] A. G. Jivani. A comparative study of stemming algorithms. URL http://www.kenbenoit.net/courses/tcd2014qta/readings/Jivani_ijcta2011020632.pdf.

[6] D. T. Lang. Package xml. URL https://cran.r-project.org/web/packages/XML/index.html.

[7] C. D. Manning and P. Raghavan. Introduction to information retrieval. URL http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.

[8] L. Nolan. *XML and Web Technologies for Data Sciences with R*. URL http://www.springer.com/us/book/9781461478997.

[9] U. N. L. of Medicine. Pubmed. . URL http://www.ncbi.nlm.nih.gov/pubmed/.

[10] U. N. L. of Medicine. Medical subject headings, . URL http://www.ncbi.nlm.nih.gov/mesh.

[11] U. N. L. of Medicine. Medical subject headings (mesh) rdf linked data (beta), . URL http://id.nlm.nih.gov/mesh/.

[12] E. Sayers. E-utilities quick start. URL http://www.ncbi.nlm.nih.gov/books/NBK25500/.

[13] R. D. C. Team. R: A language and environment for statistical computing. r foundation for statistical computing. URL http://www.R-project.org.

[14] S. Urbanek. rjava: Low-level r to java interface. URL https://cran.r-project.org/web/packages/rJava/index.html.

[15] W3C. Rdf resource description framework, . URL http://www.w3schools.com/xml/xml_rdf.asp.

[16] W3C. Rdfs resource description framework schema, . URL http://www.w3schools.com/xml/xml_rdf.asp.

[17] W3C. Sparql query language, . URL http://www.w3.org/TR/rdf-sparql-query/.

[18] w3c Cons. Rdf turtle. URL http://www.w3.org/TR/turtle/.

[19] H. Wickham. devtools: Tools to make developing r packages easier. URL https://cran.r-project.org/web/packages/devtools/index.html.

[20] E. Willighagen. Accessing biological data in r with semantic web technologies. URL https://github.com/egonw/rrdf.

[21] D. Winter. Package rentrez. URL https://cran.r-project.org/web/packages/rentrez/rentrez.pdf.