UNIVERSITÀ CA' FOSCARI – VENEZIA
Dipartimento di Scienze ambientali, Informatica e Statistica
Masters Degree programme Second Cycle (D.M. 270/2004)
in Computer Science

Final Thesis

Graduand: Alessandro Concina
Matriculation Number: 835543

Data cleansing
Different approaches for different datasets

Supervisor: Chiar.mo prof. Renzo Orsini

Academic Year 2014-2015

To Sofia and Michela

Contents

List of Figures

List of Tables

Acknowledgements

There are a lot of people that I would like to give thanks to. Starting from professor Orsini that helped me and have been so kind during the all the works presented in the thesis. Another person that I would really want to thanks is Paolo Soleni that supported me and helped me to find some solutions to complete my studies. Thank you to all my family, my friends, my colleagues and to my girlfriend Michela. I would like to say thank you to Enrico Steffinlongo that is also related to all my University career. And finally I would like to give thanks to all the people that have been extremely important during these years

Abstract

Data cleansing is a commonly encountered problem while working with documents with the purpose of extracting useful information. There exist several solutions that actually are related to the dataset type, the data organization and the working domain the dataset is related to. There exist different issues related to the datasets like the merge between different datasets, the data extraction from a database that can contains different errors. Errors in datasets' records is a well known problem that can affects the results of a data extraction operation. In the presented work some methods have been implemented in order to clean up data fields in a database in order to characterize and classify them.

Chapter 1

Introduction

1.1 A brief introduction

Datasets and databases represent a huge source of information by their definition. They actually were born to store data. From this derives the huge importance that these structures represent in most fields, going from Science to the Commercial one. A common issue is represented from the merge of different databases. In fact, this represents a problem while trying to merge data structures that can represent the same information represented in different ways. Another issue can be encountered while constructing datasets composed by measurements of phenomenon; it would be necessary to check the validity and the correctness of the values measured in order to efficiently check the process performances. In case of unavailability of some values or errors in measures the monitoring results could be invalid or incomplete. It would be necessary to reconstruct the missing values, correct the errors and obtain valid results using the available records. This process is also known as data reconcili-

ation and have been threated for industrial processes measurements [14]. These examples show the importance of providing datasets that contain values that have to be simple to read, understand (also from automatize instruments), consistent and correct. From these three features is possible to define some key point that data should respect:

- Records from a dataset should have a standard form: It's reasonable to think that a dataset that is composed by records that all have a specific format is more readable and understandable than an heterogeneous one;
- Each record should give the same type of information as another record of the same set: this means that each record of the same class is supposed to have the same properties of the others, they are so considered as consistent;
- Records should not contain errors. This feature is defined to avoid information loss;

Unfortunately datasets are usually affected from errors that often make difficult to read or understand it. Make data "clean" and readable (or able to be parse) is the main goal of data cleansing. There exists several studies on this problem, unfortunately they are strongly related to a specific field like health care [20] or other fields.

1.2 Motivations for data cleansing

Requirements for data cleansing operations derives from the origin of data records. In particular while working on datasets that comes from

different sources there will be different ways to represent a single piece of information. There are other problems deriving from different sources, like the way in which the records are stored; one example is provided from a set of data inserted from different users: there would probably be problems deriving from the lexical errors or, most commonly, different ways to represent some special terms like acronyms, abbreviations and other etc. Another problem that could be commonly encountered derives from the recognition of words that are synonymous or sentences that have the same meaning. However a good cleansing policy would be very helpful in order to improve mechanisms used for retrieving good information from the records. To achieve this purpose is necessary to understand which are the main errors that can occur in a set. Obviously the different problems that can be encountered depends both on the data type and the issues that can affect the records. It is obvious that a lexical error cannot be threated as a pattern mismatch error. In the presented case study different types of records have been encountered that have required different approaches starting from the general issues that commonly affect the given dataset defining a possible solution for each ones.

This thesis talks about the experimental result obtained after performing a cleansing operation on a institutional database that is a Italian Public Administration Data catalog that is one of the AgID (organization for Italian digital data) datasets. In this work will be first briefly illustrate general problems behind data cleansing and some techniques commonly used. After that there is a description about the issues affecting the AgID database and some solutions tested with their results.

Chapter 2

Content

2.1 Main error types

Before introducing the approaches of data cleansing is necessary to understand which could be the possible errors that can be encountered into a set. In particular, it would be interesting to understand which are the problems that commonly affect a dataset. The study mainly focus on a scenario in which two (or more) users have to write some information of different origins inside a grid table composed of different columns. The single field represent a piece of information, but the tuple related to other fields in the same row also represent an information. Sometimes a single field could not describe anything useful on the other hand it, as a part of its row tuple, is fundamental. An example is given from the table below:

Table 2.1: Table with record information composed by 3 elements

Name	**Surname**	**Role**
John	Red	Teacher
John	Green	Student
John	Red	Student
Emily	White	Teacher
Emily	Red	Teacher

In fact by only looking to the "Name" field in the table above someone could think that there's a record repetition.In fact the entry `John` is repeated for 3 times and `Emily` for 2 times.

Table 2.2: Table 2.1 only considering the `Name` field.

Name
John
John
John
Emily
Emily

It seems that the first 3 records and the last 2 are equivalent. This is obviously false because by looking at the "Surname" field is possible to notice that they are different.

Table 2.3: Table 2.1 considering the `Name` and `Surname` fields.

Name	**Surname**
John	Red
John	Geen
John	Red
Emily	White
Emily	Red

Also in the tuple `(name,surname)` there are records that looks to be repeated. By adding the last field `role` and considering the `(name,surname,role)` tuple is possible to notice how the represented records are not repeated in the table. This is verified assuming that a person in that table should only be a teacher or a student. This example shows how the single tuple record could be actually be considered as an atomic informative unit. Its components do not describe a specific phenomenon better than the entire tuple element. This concept could obviously be applied to records composed of hundreds of field and thousands of rows. While the purpose is to extract pieces of information from a dataset there are some problems that could make a query less effective, in fact some records could be corrupted and they cannot be as informative as expected. The errors that can affects a dataset have been divided in three categories: lexical, format and structural errors.

2.1.1 Lexical errors

This category represents all the possible errors like spelling errors, and in some cases lexical errors. In particular most of the common errors are related to word misspelling. This errors are common for hand inserted data. Considering a dataset that is composed by multiple input sources, the number of possible lexical errors dramatically grows. This category of errors can represent a problem while trying to automatically recognize words or terms. In fact, the use of exact matching approaches for strings to identify a record that is affected by a lexical error will give no results.

Table 2.4: Example of lexical error in `Type` field.

Name	**Cateogry**	**Type**
Tom	Animal	Cat
Jerry	Animal	Rat
Scar	Animal	Lion
Emily	Animal	Leopard
Mufasa	Animal	Leon

In the last row it's possible to notice how the `Type` field presents an error (`Leon` instead of `Lion`). Performing a query for matching the name of all the lions looking at all records that have `Type` equal to `Lion` that record won't be selected.

2.1.2 Format errors

This is a more specific category of errors. Is defined as a format error the one in which a specific given pattern is not respected or in which one or more fields are empty. In this case there could be problems while parsing the record or trying to categorize it. Most commonly a record affected from this issue could not give the expected information, this because would not be possible to categorize the content or understand it. If the record is empty the informative content could be incomplete, this because the related tuple is incomplete. Suppose to have a table in which users have to write the licenses of some softwares they use. What is expected is to find alphanumeric codes, or organizations names, unfortunately someone could misunderstand this field and simply specify if there is a license code (inserting "yes" in the "license" field) as shown below:

Table 2.5: Example of Format errors

Id	**License**
451	Creative Commons CC BY 3.0
452	Creative Commons e IODL 2.0
452	Creative Commons GNU GPL
453	Yes

This issue have also to be considered as a format error because the field contains an invalid value with respect to the expected result.

2.1.3 Structural errors

This is a problem that is specific to the case study presented in this thesis. It regards the shift of a field content to another field space. The tuple can contains all the information required but in the wrong position. This causes multiple errors in the same record. It is difficult to recognize this issue because the final effect is similar to a format error, but obviously, it cannot be threated as that category.

Table 2.6: Example of Structural error with a record shift

Complete name	**Address**	**Phone number**
Alex Brown	23 AA Street Venice	095678767
	John Max	3 ZZ street Rome
Mike Foo	12 BB Street Rome	444323321
Alice Blue	1 CC Street Milan	343221343
Tom Bean		25 FF Street Venice

This example shows a record shift that affected 2 rows (the second and the fifth), this caused the loss of an item of the tuple and the lost of meaning of the other item of the affected rows. Nevertheless the other fields are inside the row they cannot be used because of their erroneous position. For instance in the second record the address field contains a name (`John Max`) and the phone number contains an address. A good practice should be to reorder the items placing the values in the correct fields in order to try to recover the record. Unfortunately this isn't so simple, because there could be fields that can contain items that can be confused each others. For example in the second record a `John Max`

square or street could exists and that field should contain the correct information. Most of time these errors have to be solved my manual reordering operations.

2.2 Overview on data cleansing

The terms data cleansing refers to the operations of detection and correction of wrong or malformed records among a dataset. This could be related to the concept of data standardization. In fact is not strange that someone would like to perform an informative search among the records. This is strongly given from the datasets that can contain repeatible or clusterizable data. Unfortunately some records that would belong to a specific group could not be matched from a clustering algorithm because of their "dirtness". So it could be necessary to cleanse them up in order to perform efficient classification or clustering operations. Data cleansing actually is much more than simply substitutes dirty records with cleansed ones. It would be necessary to decompose a record into tokens that would be manipulated, reordered and often deleted in order to [9] According to one can break down the cleansing into six steps: elementizing, standardizing, verifying, matching, house holding, and documenting. Although data cleansing can take many forms, the current marketplace and the current technology for data cleansing are heavily focused on customer lists [9].

As written by Maletic and Marcus within the data warehousing field, data cleansing is applied especially when several databases are merged. Records referring to the same entity are represented in different formats in the different data sets or are erroneously represented [15]. This is

commonly known as the merge/purge problem. The main features of this problem will be described below.

Lexical errors described in the previous chapter represent one of the most frequent problem in each dataset. In particular this is strongly verified for records composed by at most two or three words. So is necessary to provide a system to efficiently detect and correct lexical errors. To achieve this purpose some strings similarity metrics have been tested and used. The two methods considered are the Levenshtein and the Jaccard distance.

2.3 Methods for Data Cleansing

There exists different methods to perform the cleansing operations, in particular this process is necessary to consider three main macro phases:

1. Record decomposition
2. Error correction
3. Record recomposition

These three steps allow to standardize the records in order to recognize possible errors or outliers. In particular the first step make the second phase possible: in fact splitting a considerable "big" record into known blocks also make visual groups recognition simpler than analyzing the original records. After that is possible to check if some errors occur in one or more blocks. Error definition should be given by human interaction, this because the correctness or the wrongness of it depends on the context. For instance in a medical environment would obviously be used terms that

are totally different than an institutional environment. This regards both the terms used and also the semantic rules that are well separated and totally different. There could also exists a set of common errors that can affect a specific field that datasets are related to.

An important issue that is necessary to consider while performing this phase regards the use of acronyms. Acronyms could make confusion between other grammar words (for example the "USA" acronym for United States of America is equal to the Italian verb "usa" i.e. "to use"). Acronyms present other issues like punctuation, in fact it's not too strange to find the same acronym written in different ways in a dataset (e.g. "USA" and "U.S.A."). Abbreviations are also common problems very similar to acronyms, this because there exists different ways to abbreviate a term (e.g. "cod. della strada", "cds" or "cod. strada").

Finally there are two problems related respectively to the presence of terms of a foreign language with respect to the dataset language, very common in the case of language different from English and probably this is the most common problem related to orthography errors.

2.3.1 Record decomposition

The first phase allows to separate each record in a set of distinct informative units. This is important for dividing a record into a set of informative units (words). Record decomposition actually deals with the problem of error matching among a sequence of words. If we found a good method to split a complex record into simpler tokens we can try to understand their morphology and try to correct erroneous tokens. This operation actually improve the efficiency of the error detection. It also allows to

detect what we can define as a keyword that actually represent a term that could be very important for categorizing the record with respect to the set. The main idea is to split sentences into set of words in order to analyze records word-by-word. To achieve this purpose is necessary to remove all spaces or separation characters and punctuation characters; also conjunction characters and stop words could be removed because they would not be useful while looking for a consistent categorization. The final output obtained after this phase would be a set of reasonable keys without stop words like articles that will be processed in the next step.

2.3.2 Error correction

Error correction is not so simple as it appears. In fact there exists some issues deriving from the presence of ambiguous terms that could avoid the error correction process. Word disambiguation is a well known problem and it has been threated with different approaches. One of the most commons approach works by using terms dictionary or machine-readable thesaurus. This methods are a part of the knowledge based methods for word disambiguation and are simple and very effective methods, on the other hand they are not too flexible and don't consider semantic relations. Another problem that have to be considered in the error correction phase derives from spelling mistakes. These are probably the most difficult errors to recognize and correct. One solution could be represented by the use of a string similarity measure to match the correspondence between the erroneous word and an element of a list of correct terms of a dictionary. This is a very expensive solution and it's difficult to use while

having big datasets. There also exists some tools used for spelling errors detection and correction but unfortunately they are strongly related to the language words set so they could give good performance for English words but not for other languages and this could not work for idiomatic or slang terms. There also could be a problem related to the presence of international words that could make correct recognition very difficult.

2.3.3 Record recomposition

The last phase could include records grouping. There exists different methods to achieve this purpose. In this thesis will be used this different methods:

- Clustering: this method consists in assigning records choosing them from a fixed (or not) list of clusters. Usually some distance measure (e.g. Euclidian distance) are used to perform the assignment. This method could work efficiently if the number of clusters or the centroids are known. It is also important to define a good distance measure in order to perform the correct assignments;

- Association rules: solution introduced by Agrawal with the market baskets analysis problem [5]. This solution considers the relations between two or more terms that are frequent.The meaning of "frequent" represent the minimum number of times in which the relations are matched in the dataset. A possible use of this approach would be represented by considering two terms that appear together at least in `s` records of the set they are considered as related. So, they represent an unique token. This could be also used to reconstruct incomplete records. In this scenario the `s` value

represents the support of a specific association on a dataset, so the value is very important to enhance the relation between the dataset and generated association rules. This method needs a good system for generating the association rules in order to correctly cover the dataset;

- Pattern based techniques: this method works considering specific record format. Everything that would not be similar to one pattern will be considered as an outlier. The patterns have to be defined and depends on the context. This method is very effective while considering well trying to obtain well structured output records;

2.4 Clustering

There exists several approaches implemented in order to perform data cleansing over datasets that are supposed to contain records with the same informative content. By this assumption is possible to consider an approach based on the concept of cluster. The clustering operation is mainly based on the definition of distance between two objects. In particular given a distance function $d(x, y)$ while having n clusters that have associated $c = \{c_1, c_2, \cdots, c_n\}$ centroids an element e will be associated to the cluster which respect:

$$assoc(e, c) = argmin(d, c_i) \text{ for each } i \in [1, n]$$

One of the simpler and most used clustering algorithm is the K-Means that is the one used here [9] generally requires fixed centroids.

2.4.1 K-Means

This algorithm clusters a group of data vectors into a predefined number of clusters. It starts with randomly initial cluster centroids and reassigns the data objects in the dataset to cluster centroids based on the similarity between the data object and the cluster centroid. It is possible to reassign the cluster centroids until them satisfy convergence criterion. In general when the centroids become more stable (changes are under a given threshold) or while reached a maximum number of iterations previously specified. The similarity function have to be specified and depends of the objects that are going to be clustered. When considering clustering between strings a similarity function is equal to approximate string matching. The main phases of K-Means are:

1. Initial random centroids selection;

2. Clustering of all the objects
   $assoc(e, c) = i = argmin(d, c_i)$ for each $i \in [1, n]$

3. Centroids reassignment:
   $c_i = \frac{\sum_{j \in [1, \|c_i\|]} d_j}{\|c_i\|}$

This algorithm is robust and very simple to implement but unfortunately could not work if the number of clusters is unknown. This means that is not possible to consider a case in which some new clusters could be encountered while performing the clustering operation. There exists clustering algorithms that are more flexible like pair-wise ones but generally they have high resources and time requirements so will be not considered.

Another drawback for this algorithm is the dependence between the final results and the choice of the initial centroids. This depends on the distance between the centroids. In fact by choosing two or more close centroids it could be possible to split objects that should belong to the same cluster in two different ones. At the same time objects that shouldn't belong to the same cluster should be assigned to the same one. While the dataset content is unknown it is very difficult to correctly choose both the the initial centroids and the number of them.

2.5 Association rules

The concept of association rules has been introduced from Agrawal and Srikant in the market basket analysis [5]. The use of this techniques allows to define relations between two or more items with respect to their frequency in the dataset. There exists an algorithm to find association rules called Apriori algorithm developed from Agrawal and Srikant. This algorithm introduces the concept of frequent subsets, this means that a couple is defined as frequent if the two elements that compose it are stored together in the set and that couple appears in it for at least n times. The frequency for a collection of recurrent items with respect to a set is called `support`. The n parameter is defined as minimum support for accepting the association.

2.6 Pattern detection

This methods work by some predefined patterns, they show the possible different ways in which items should appear. Pattern detection is useful

while having well-known classes. The idea is to associate each item to the class that is closest to it. The closeness of an item to a class have to be define with a measure of similarity or distance. This concept is very similar to the clustering operation in which objects are assigned to cluster depending on the distance between the item and the centroids. Pattern based approaches could also be used to detect and associate records that are composed by token that does not respect an expected order. For example considering a pattern for a postal delivery string composed by:

```
<street nr.>+<street>+<Road,Square,etc.>+<Zip Code>+<City>+<Country>
```

A correct record is:

```
125 Trafalgar Road 10524 London UK
```

But there are other possible correct record that respects a different order or there could be missing tokens:

```
Trafalgar Road 147 London 10524 UK
Trafalgar Road London 127 UK 10524
  Trafalgar Road 10524 London UK
   147 Trafalgar Road London UK
```

These records are expected to be considered similar each others. This because it could be necessary to detect and correct errors or to complete incomplete records and recognize tokens with no reference to their position. Detecting a pattern among a set of records could help to recognize

and correct "structural" errors. To associate the single token is reasonable to apply a similarity function to detect which section belongs to the specified token. Once associated all the tokens is possible to identify which token are empty or is possible to apply an error correction algorithm. This approach needs to be fed with the patterns that will be searched. This could also represent a drawback for this method.

2.7 String similarities measures

Defining similarities between strings is a quite complex process in which, given two strings s_1 and s_2, there exists a value d that represents the similarity or the distance between them. The definition of a metric d have to satisfy the triangular inequality, this means that given three strings s_1,s_2 and s_3 if s_1 is similar to s_2 and s_2 is similar to s_3 this doesn't mean that s_1 must be equal to s_3. Moreover also the reflexive and symmetric axioms have to be verified. For example consider: D Similarity function
$D(s_1, s_2) \wedge S(s_2, s_3) \not\Rightarrow S(s_1, s_3)$
$s_1 =$ "`Meaning`"
$s_2 =$ "`Bean`"
$s_3 =$ "`Bin`"

For example defining S as the number of operation of insertion, deletion and substitution that are required to make a string equal to another the result for the three strings shown above is:

- Reflexive axiom: $D(s_1, s_1) = 0$ (No insertion, deletion or substitutions needed)

- Symmetric axiom: $D(s_1, s_2) = D(s_2, s_1)$ (Sum of insertions, deletions and substitutions are equal for both results)
- Triangular inequality:
  $D(s_1, s_3) \geq D(s_1, s_2) + D(s_2, s_3)$
  $D(s_1, s_2) = 5$ (4 insertions and 1 deletion)
  $D(s_2, s_3) = 3$ (2 deletions and 1 insertions)
  $D(s_2, s_3) = 8$ (2 deletions and 6 insertions)

This metric based on insertion deletion and substitution is called edit distance and there exists many different solutions. Edit distances are commonly used in bioinformatics for DNA sequences similarity. There exists other string similarities metrics based on most common prefix/suffix that could be useful for composed terms. The main metrics considered are:

- Jaro-Winkler;
- Levenshtein;
- Jaccard;

2.7.1 Jaro

It's a non edit metric based on the number and the position of the common characters between two strings. As the jaro-Winkler it actually is a metric that calculates the similarity by looking at the transposition required on a string to make it close to another one. This index, represented in general as a number between 0 (totally different) and 1 (equal) is defined as follow: Given two strings s_1 and s_2 such that $s_1 = a_1, a_2, \cdots, a_n$

and $s_2 = b_1, b_2, \cdots, b_k$ is possible to say that a character $a_i \in s_1$ is said to be common in s_2 if exists a character $b_j \in s_2$ that respects the equation $b_j = a_i$ for $i - \frac{min(\|s_1\|,\|s_2\|)}{2} \leq j \leq i + \frac{min(\|s_1\|,\|s_2\|)}{2}$. Considering $s_1^{'} = a_1^{'}, \cdots, a_n^{'}$ with $s_1^{'} \subseteq s_1$ and $s_2^{'} = a_1^{'}, \cdots, a_k^{'}$ with $s_2^{'} \subseteq s_2$ it is possible to define a transposition such that $a^{'}i \neq b^{'}i$. Let T be the number of the possible transpositions between $s_1^{'}$ and $s_2^{'}$ divided by 2, so the Jaro similarity index is defined as:

$$Jaro(s_1, s_2) = \frac{1}{3}\left(\frac{\|s_1^{'}\|}{\|s_1\|} + \frac{\|t_1^{'}\|}{\|t_1\|} + \frac{\|s_1^{'}\| - T}{\|s_1^{'}\|}\right).$$

2.7.2 Jaro-Winkler

Is a variant of the described Jaro distance proposed by Winkler in 1999. It's mainly used in the area of record linkage for duplicate detection. The higher the Jaro-Winkler distance for two strings, the more similar the strings are. It looks for the longest common prefix between two strings, it works better for short strings like names.
Considering P as the longest common prefix up to a maximum of 4 characters the Jaro-Winkler index is defined as:

$$Jaro - Winkler(s_1, s_2) = Jaro(s_1, s_2) + lP(1 - Jaro(s_1, s_2))$$

where l represents a scale factor.

2.7.3 Levenshtein

Probably the most famous (and) simple edit distance. It's simply based on the number of simple operation that have to be performed to make a string A be equal to another string B and viceversa. Given two strings

A and B the possible operations that can be performed to make B equal to A in the Levenshtein metric are:

- Insertion of a character
- Substitution of a character with another in B
- Deletion of a character.

For example the words $A = dog$ and $B = fog$ have a distance equal to 1: in fact starting by *dog* it is only necessary to substitute the `d` in A with `f` in B. Let's take a more complex example: $A = houses$, $B = mouth$ the step according to Levenshtein distance are:

1. Substitutes the first char `m` in B with `h` char: $B = houth$
2. Substitutes the fourth char `t` in B with `s` char: $B = housh$
3. Substitutes the fifth char `h` in B with `e` char: $B = house$
4. Insert at the end of B the `s` char: $B = houses$

Table 2.7: Levenshtein matrix example

		H	**O**	**U**	**S**	**E**	**S**
	0	1	2	3	4	5	6
M	1	1	2	3	4	5	6
O	2	2	1	4	5	6	7
U	3	3	2	1	2	3	4
T	4	4	3	2	2	3	4
H	5	1	2	3	3	3	4

According to Levenshtein the distance between *mouth* and *houses* is $Lev(mouth, houses) = 4$.
As written before this is a very simple and intuitive method. On the other hand by its construction Levenshtein distance is not capable to identify as equal two strings which presents misspelling error: `bread` and `braed` are supposed to be equal. Unfortunately the Levenshtein distance between these two strings is 2 (1 insertion and 1 deletion). For example the word `read` according to this distance metric is more similar to `bread` than `braed`. To solve this problem there exists a variation of the Levenshtein algorithm that also considers transposition between contigous characters. This is called Damerau-Levenshtein distance and is commonly used in spell checkers. This algorithm has been tested while performing the error correction that will be presented in this thesis with good experimental results. Unfortunatelly it has computational high costs and makes string comparisons too heavy, so other techniques have been considered.

2.7.4 Jaccard

Introduced by Paul Jaccard this represents a similarity index. The corresponding distance that is complementary to the index measures the dissimilarity between strings. In fact given two objects s_1,s_2, $Jaccard(s_1, s_2)$ returns a value between 0 (totally equal) and 1 (totally different). This is a statistical index that works by looking at the features of two sample sets. It calculates the similarity by looking at the intersection and the union of these two sets.

$$J(s_1, s_2) = \frac{\|s_1 \cap s_2\|}{\|s_1 \cup s_2\|}$$

The Jaccard distance is defined as:

$$J_d(s_1, s_2) = 1 - J(s_1, s_2) = 1 - \frac{\|s_1 \cap s_2\|}{\|s_1 \cup s_2\|}$$

Actually it looks at the common terms between the two strings. This is useful while trying to perform a similarity measure between medium size sentences and it is commonly used in Information Retrieval.

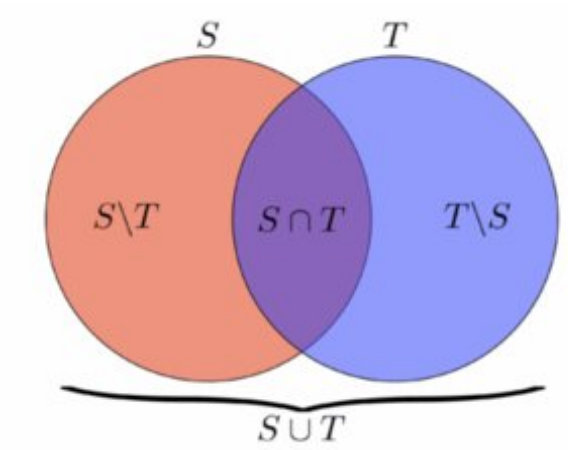


Figure 2.1: Schema of union and intersection between the sets `S` and `T`

2.8 IR-Information Retrieval

Information retrieval is the set of operations and techniques that are used to store, categorize, associate and find documents and information on the web. It's a core part of the web document indexer system. It allows to perform search using search engine. The idea is to consider all the resources as documents. IR could also be used to facilitate "semistructured" search, for example while looking for documents that contain a specific term. IR field also regards the processing of retrieved documents, like clustering in order to return documents that have been consider to

be similar basing on the argument inside them. This operation looks like a categorization of the retrieved documents and represent a very powerful instrument while performing searches. Documents are composed of words and obviously as soon as someone is interested in the content of a document as words are important. Words are combined each others in sentences. This means that the order of these words has a relevancy. Also the frequency of a word should be considered, in fact a word that appears for a certain number of times would probably be more relevant than other words. On the other hand is necessary to consider that there exists words that are very common in natural language but they don't have an high informative content like articles, prepositions, etc. They are also known as stop words. One technique works by recognizing the relevant terms using them to tag the documents (or the item) considered. The importance of some terms, that could be considered as "hot keys" actually represent what an indexer works for: in fact hot keys represent the features for an indexer system that will mark documents with a set of tags. A tag is an information that categorize a document in order to make it accessible or ready for a consequent clustering operation. Documents are first parsed in order to construct an `inverted index`. An inverted index is helpful to efficiently perform fast query search, this because it marks the documents with some tags that make more simple the search operations. The purpose is to find the documents where a word occurs. First a forward index is developed, which stores lists of words per document, it is next inverted to obtain an inverted index. These operations allow the obtained index to access to a list of documents by a list of words that represent them.

2.8.1 BoW - Bag of Words

Before describing the inverted indexes the Bag of Words model is introduced. Bag of Word is a term to define a model commonly used in IR. This works considering the words that a document contains. This is similar to think as a set of tags that define a specific document[20]. This representation does not consider the sentences or words order in a document but only its presence or absence. This means that the sentences syntax is not considered. This is a powerful feature while dealing with items that have different syntaxes, or can contain errors. For instance it's possible to match two items that shares at least some common words ignoring them compositions:
For example let's suppose that there's a set of documents S and two documents A,B such that $A, B \in S$. The documents are composed as follow: $A = Paul\, loves\, to\, live\, in\, Liverpool. Liverpool\, is\, in\, England$ and $B = Paul\, loves\, England$, applying a BoW model the result is:

$A = \{Paul, loves, to, live, in, Liverpool, is, England\}$
$B = \{Paul, loves, England\}$
The two lists are used to compose the "bag" for the two documents:
$Bag_{(A,B)} = \{Paul, loves, to, live, inLiverpool, is, England\}$
For each document a list of term frequencies is created with respect to the bag such that the element in position i represents the number of time in which the item $Bag_{(A,B)}[i]$ is in the document:
$Bag_A = \{1, 1, 1, 1, 2, 2, 1, 1\}$
$Bag_B = \{1, 1, 0, 0, 0, 0, 0, 1\}$

While performing a search among documents looking for the words:

$Q_1 = \{Paul, loves, England\}$ the results will fit both document A and B. If the search of the word $Q_2 = \{Liverpool\}$ will only match the document A.
The two lists above actually represent term frequency lists. Define the term frequency is important while trying to understand if there exists words that are more important than others. It actually represents the weight of a term with respect to a document. Otherwise it would be necessary to consider groups of terms, this because there exists a strong relation between a certain number of items: this groups of items are also known as n-grams. Looking for n-grams instead of single terms could make more accurate and specific a search. A more specific approach is described here [6] Consider for example a set of document:
$A = \{Paul\, loves\, to\, live\, in\, Liverpool.\, Liverpool\, is\, in\, England\}$
$B = \{Paul\, loves\, England\}$
$C = \{Geography:\ England\, is\, a\, country\, that\, is\, part\, of\, the\, United\, Kingdom\}$
$D = \{Geography:\ New England\, is\, a\, region\, which\, comprises\, six\, states\, of\, the\, Northeastern\, United\, States\}$

Suppose that someone want to perform a search to all the documents that regards the geography field and contains the term England. On one hand by performing a simple search using the key $Q = \{England\}$ among the set of documents $Set\{A, B, C, D\}$ all the documents will be retrieved. On the other hand using the digram $Q = \{Geography, England\}$ the resulting documents retrieved will be C and D that looks to be more close to the user's purpose. N-grams are used to deals with the problem of ambiguity of some terms. The method that will be used to generate n-grams is the Most Frequent n-grams, similar to one of the methods described in

[6] and works as follow: Given a list of items (sentences) for each item:

1. Split the item into a set of tokens
2. Take the entire list of n tokens (an n-gram)
3. count how many time it appears in the other items and calculates the frequency $f = \frac{\#ofmathes}{\#oftotalitems}$ (sentences).
4. If $f \geq k$ it's saved in the n-grams list.
5. otherwise one item is removed and the operation is repeated for the (n-1)-grams
6. Repeat until the number of items in the n-grams is higher than 1

2.8.2 Inverted index

The basic idea behind an inverted index is to keep a dictionary of terms and for each term compose a list of documents the term occurs in. Each item in the list is also known as a posting item, the list is so called postings list. All the postings lists taken together are referred to as the postings. The list is sorted alphabetically and the postings list are sorted by document ID. The main steps to construct an inverted index according to [16] are:

1. Collect the documents to be indexed:

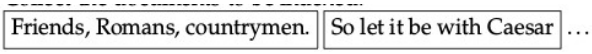


Figure 2.2: Collection of documents used for the index construction `S` and `T`

2. Transform each document into a list of tokens: this is commonly defined as bag of words. The idea is to consider a document as a set of words.

Figure 2.3: Terms tokenization `S` and `T`

3. Process the tokens in order to make them standard: This step could include multiple operations. For instance the words could be stemmed in order to consider the root of a word. For example it allows to consider the term "`compose`" and "`composition`" as equivalent. This is correct because they share a common root.

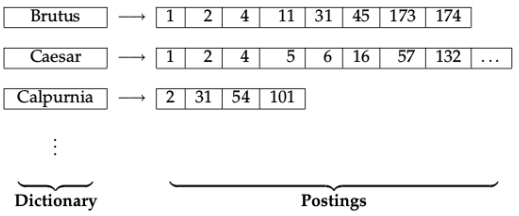


Figure 2.4: Structure of the obtained inverted index: terms-key on the left side column and the documents id's as values. `S` and `T`

4. Index the documents that each term occurs in by creating an inverted index, that consists of a dictionary and postings. The final result will be a dictionary in which the keys are the terms inside the

documents that have been considered while constructing the index and the values will be the indexes that refers to the documents.

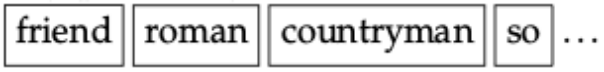


Figure 2.5: Terms standardization `S` and `T`

This solution is also known as boolean search model, this because it gives information about the presence or the absence of a term inside a document. Nevertheless, it doesn't give information about the frequency of a term inside a document. Term frequency is important because it allows to rank documents according the relevancy. The main idea is to consider a a document d_1 is more important than another document d_2 with respect to the term t if the frequency of t in d_1 is higher than the frequency of t in d_2. While performing a search is obvious that, the higher is the rank, the more important the documents are.

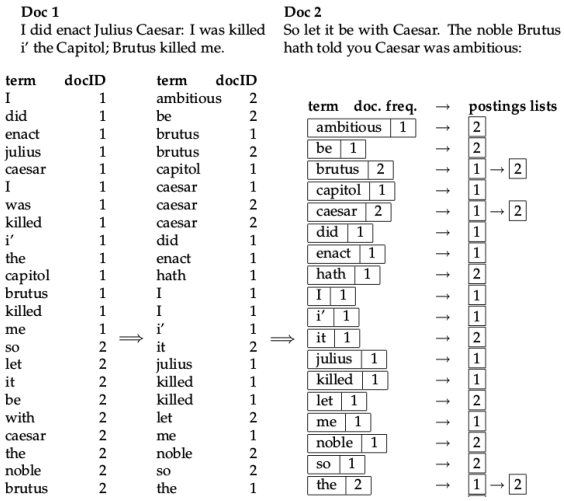


Figure 2.6: Index construction considering terms frequency `S` and `T`

Chapter 3

Related Works

When this work have begun an interesting tool called OpenRefine have been considered. It's a tool initially developed by Google that works with different type of data. It is a free software available online [4] that allows to perform different operations going from cleaning values to transforming them from a format to another one or integrating them with other datasets. It supports different data file formats input and export and could efficiently work with different types of data going from string to date type. Between the import and export there's occur different operations including clustering and syntax cleaning also performed via APIs or external web services. The workflow for a dataset (or a file) that goes from the import to the export is shown in the picture:

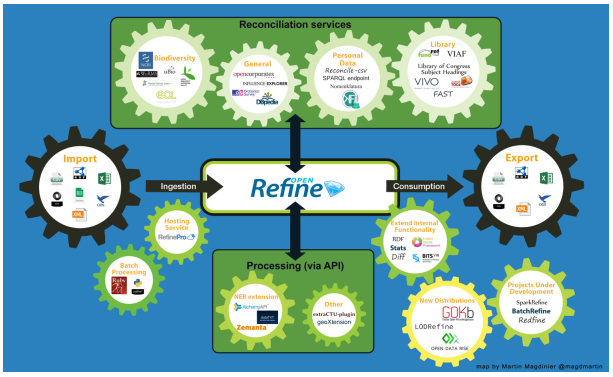


Figure 3.1: OpenRefine ecosystem schema

The operations that have been considered regard the recognition of similar records in order to group them. This means that the tool is developed to find items that are supposed to be similar (or equal) but have different representations. OpenRefine allows the user to use different similarity metrics and clustering algorithms [3] in order to compose the groups. It also suggests some hints that allows to merge similar clusters. This methods works only on the similarities between the raw data not considering the semantics but includes external services (like Freebase). The clustering algorithms used have been divided in two classes: key collision and Nearest Neighbor methods.

3.1 Key collision methods

This class of clustering methods work basically considering two components: keys and buckets. A key is a value that is meaningful to describe a set of items. In particular is a component of an item. Supposing to use strings, a key is represented by one or more words, indeed a bucket is

represented by all the items (or strings) that contains a particular strings. This representation recalls the IR model of BoW or the structure of an inverted index. When two or more strings share a common key it's called a key collision, as defined by the methods' name. All the methods belonging to this class are powerful, fast and cheap in complexity, in fact it's complexity is linear for all the methods.

3.1.1 Fingerprint

It's the fastest simplest method presented. It manipulates the records removing punctuation, white spaces and standardize the input substituting all accented letters with unaccented equivalent ones. After this operations the sentence is "tokenized", the duplicated token are removed and the token are lower cased, sorted and joined together again. This makes possible to match strings that contain the same tokens but in different order and deals with the problem of matching strings that are equivalent but differ from the presence or the absence of some accented or unaccented words.

3.1.2 N-gram fingerprint

This method actually is an extension of the Fingerprint method. Unlike Fingerprint, N-gram also consider tokens composed of n characters of a token. It performs the same operations described in Fingerprint algorithm but unlike it this version splits the string into substrings of n chars. After that they are sorted and recomposed. For example given the string:

$$s = "Apple"$$

with $n = 2$ becomes:

$$S = \{ap, pp, pl, le\}$$

after the sort operation

$$s = "apleplpp"$$

This method that looks like to be not so useful could be used to match two strings that presents small differences (for example a syntax error).

3.1.3 Phonetic fingerprint

This method differs from the previous two in the way that it considers the tokens. It clusters items that have the same phonetic, so it consider equal two sentences that have the same "sound" when they are pronounced. This means that two strings that are lexically different but have the same phonetic will be recognized as similar. This methods can detect misspelling errors that cannot be recognized from Fingerprint and N-gram Fingerprint. The main drawback for this algorithm is that it doesn't efficiently work with sentences that contain multi-language words. This phenomenon is very common while considering languages different from English that can also contain international (English) terms. This can cause misunderstanding in the recognition.

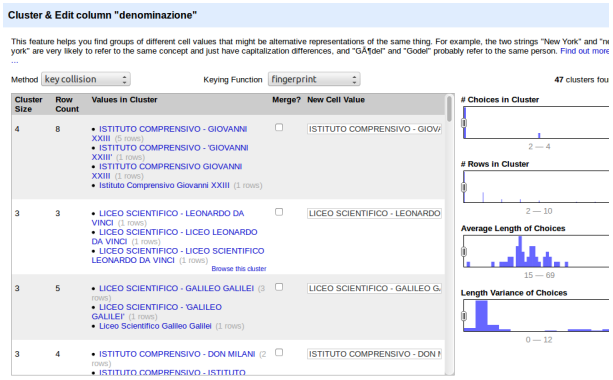


Figure 3.2: Example of Clustering using FingerPrint method

3.2 Nearest Neighbor methods

Key collisions algorithm are efficient and fast but unfortunately they look to be too much lax or on the opposite to strict. They present a lack of flexibility that makes them lost some item in the clustering operations. Nearest neighbor is commonly associated to the KNN that actually is a classifier. The concept of neighborhood has been used from [19] to perform a clustering called Nearest Neighbor chain algorithm. It works by considering a chain that represent the neighborhood between two clusters. It starts from n clusters where n is the number of items and starts to define the neighborhood chain. The clustering operations finish when two cluster are mutual neighbor. To define the neighbors a distance function is needed.

This method is very flexible but have the problem of the complexity: to compare n items $\frac{n(n-1)}{2}$ distances have to be calculated. While n grows the number of iterations dramatically grows. As written before, to

perform a chain Nearest Neighbor clustering is necessary to define a distance function that, taken two items returns the distance between them. The most simple and commonly used is the Levenshtein distance. This distance metric defines the distance between teo strings as the number of operations of insertion, deletion and substitution required to make the two strings equal.

3.2.1 PPM

Prediction by Partial Matching is a statistical data compression technique introduced by John Cleary and Ian Witten back in 1984 [7]. It uses the concept of Kolmogorov complexity to estimate the similarities between strings. PPM works predicting the next symbols by the knowledge of previous n symbols in the uncompressed symbol stream. The prediction could be make either with a Markov model, this is also called prediction by Markov model of order n, or with other models different from it. Starting from a sequence of n symbols the $n + 1$ symbol is predicted referring to the context, so the most probable symbol with respect to the sequence $X = [x_1, x2, \ldots, x_n]$ where x_i represents the i^{th} symbol. If a prediction for a n-sequence cannot be done the (n-1)-sequence is used for the prediction and so on. This method does not work as expected for short sequences, in particular while the number of symbols is shorter than 4. Another problem also related to the sequence length is the number of false positive generated.

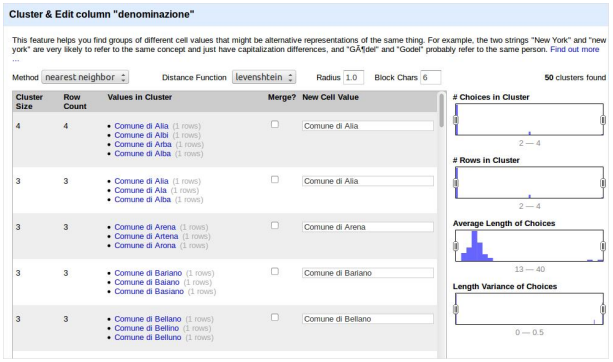


Figure 3.3: Example of NN using Levenshtein distance

3.3 Results

The tool looks to be very useful referring to the clustering operations. In fact given a dataset that contains repeated values or field that are expected to be equal both the two classes of methods look to work quite good. As expected there are some difference deriving from the method that has been chosen for the clustering operation. Probably the Nearest Neighbor based methods are more flexible and allow to match values that shares common prefixes (like sentences with the same subject and verb) by simply adjusting the radius. On the other hand they look to be expensive and it's difficult to use them for clustering more than 1000 items.

Chapter 4

AgID dataset cleansing

4.1 Introduction to AgID

This study refers to a project started in 2014 when Italian government published law n. 114/2014 [2] about data of public administrations. This law actually proposed to create a project of an Italian Public Administration Data catalog in order to make data of public administrations visible and accessible and fully accessible to italian citizens [1]. The organization that has been working manages this catalog is called AgID (acronym for Agenzia per l'Italia digitale). AgID asked to Ca' Foscari University to work on data, in particular to prof. Renzo Orsini and prof. Agostino Cortesi in order to make them available on the net and accessible to every user [1]. Data have been processed and several operations have been performed in order to obtain the actual structure of the database, shown in the image below.

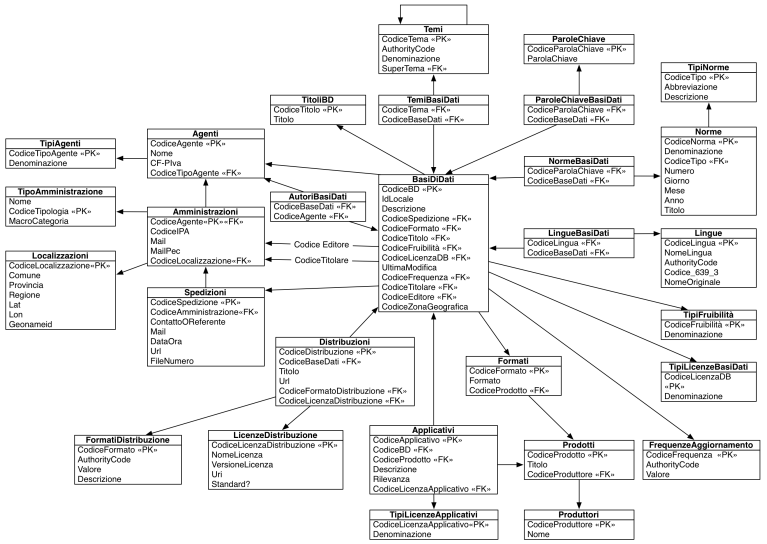


Figure 4.1: Structure of AgID dataset

Information stored in the database have been obtained from the organizations by making them compiling an Excel spreadsheet document. The way in which data has been taken caused the errors that affect the dataset. In particular several problems related to the "cell-shift" problem that caused both the generation of uninformative records and several outliers. There also is the problem related to the way in which data have been inserted. In fact by allowing people to write everything they want there could be a lot of different ways to express the same record value. An example could be the multiple ways in which different users could write an address: the main error that could be done is to consider two records that actually contain the same information written in different ways as two distinct items. This problem has been frequently encountered in the database, in particular in the `Tipi_licenze_app` and `Tipi_licenze_db` tables. These contain information about the softwares and databases li-

censes related to the products shown from the main db. The number of different licenses inside the two tables was greater than 300 different licenses while the expected, and correct, number of different types of them was about 5 or 6. Another table that contain records that have to be considered ambiguous is the `Temi` table. In fact there were almost 6000 different values while the categories related to that entity were only 13. There also were several syntactical and misspelling errors that make difficult, or sometimes impossible, to perform an automatic recognition. It was necessary to try to correct most of the common errors in order to obtain better results while clustering the items. Syntactical errors' correction is a quite simple operation when considering the recognizing of synonymous terms, problem that would require the use a WordNet processor. In the presented solution the recognition of synonyms have been performed using a manual constructed dictionary that looks to be not much flexible as using a WordNet processor but, on the other hand, because of the specificity of the lexicon it have been demonstrate to be a simple and effective solution.

4.2 Table Temi

4.2.1 Table description

The most important field in the `Temi` table is given by the `denominazione` one in which are stored the information about the subjects of the reported databases.The main issue was to associate data values that could be equal but that were in a different form. In the db there were about 4000 different values related to at most 200 to 300 distinct subjects. Another problem derives from the requirement to categorize all the items in order

to obtain the following categories:

1. Agriculture, fisheries, forestry and food
2. Economy and finance
3. Education, culture and sport
4. Energy
5. Environment
6. Government and public sector
7. Health
8. International issues
9. Justice, legal system and public safety
10. Regions and cities
11. Population and society
12. Science and technology
13. Transport

These categories has been provided from the European Union. Unfortunately is quite difficult to associate the values inside the db with the given categories. This because classes have not been decided by looking at the records available in this database. Furthermore, people that gave data didn't know the categories a priori so it wasn't almost impossible to perfectly fit the given classes. This in shown by `energy` category, that will be empty at the end of the clustering operations. On the other hand there exists many subjects that could belong to more than one category. This is possible because there isn't an accurate description for each of the provided classes. So it is necessary to create an intermediate layer in which the items are categorized and clustered with the informative

content related with the values inside the dataset. The idea is to create some subcategories that would obviously be less than the total number of records and then "merge" them to obtain the final categories. To achieve this purpose a tagging system has been constructed. It works by picking the most relevant terms in order to construct a list of keys that would be considered as descriptors of the items.

4.2.2 Main phases

The cleansing operations on this tables have been performed by respecting the phases described in chapter 2 by considering some sub-phases. In the first phase some pre-filtering operations have been performed to correct well known and common errors. After that, in the second phase the item values are split into tokens and the non informative words (called stop words) are removed. Also punctuation is removed and all the words are transformed to lower case. Finally every accented letter is transformed to an unaccented letter and non Unicode characters have been removed. The second phase is performed by comparing the tokens each others, with a distance function in order to detect simple errors and correct them. The algorithm to perform this operation works as follow:

- For each token count how many time it occurs in the set
- Construct a dictionary d with key equal to the token and value the number of occurrence
- Take each pair of keys of the obtained dictionary k_1, k_2
- Calculate the distance $distance k_1, k_2$;

- if $distance k_1, k_2 \geq t$ then it looks for the key that have the higher "support", $max dk_1, dk_2$ substitutes every token with lower "support" with the higher one, sum the two values in the entry with higher "support" and delete the other entry from the dictionary.

- otherwise they are not considered as equal and another pair is considered

This is actually a fuzzy correction: the idea is to consider as correct values the ones with higher frequency in the set. There is also the problem of choosing the correct value for the threshold t, it have been adopted a strategy based on the words length. An acceptable t will be the one that is smaller than the 20% of the length of the smaller word. In practice $t = t = \frac{2\,min(length(s_1),length(s_2))}{10}$. Once performed the first phase two methods to perform data decomposition have been implemented and tested. One of these is related to the concept of neighborhood and it's similar to the nearest neighbor chain hierarchical clustering algorithm [19]. The key concept in this solution is similar to the chain one: in fact it tries to cluster together each cluster that are expected to be close. On the other hand the stop condition is defined from an user, that would decide when to finish the operation.

4.2.3 Neighbor clustering

This approach is similar to the one used by Open Refine tool for performing clustering operations. At the beginning of the operation each element is consider as a separate cluster. The algorithm tries to cluster all the items that have a distance smaller than a given threshold `maxDist` and assigns to all the items a specific value that will be considered as the centroid. The method requires a high human interaction. At each iteration the threshold parameter is incremented in order to try to match items that are less close than required from the previous step. The idea is to match the closest items and then make conditions less restrictive to extend the search. The candidates clusters are proposed to an user with using a GUI. He can decide to accept or reject the proposed cluster. The centroid value will be selected or specified from the user and all the clustered items are collapsed to the centroid; this means that every item value will be equal to the centroid one. This is to try to make items consequentially converge to a finite number of sets. This both provides some clusters and it also standardizes the subjects. Once a step is concluded the radius (that actually is the threshold for the distance function) is incremented and the operation is repeated until the user stops the execution.

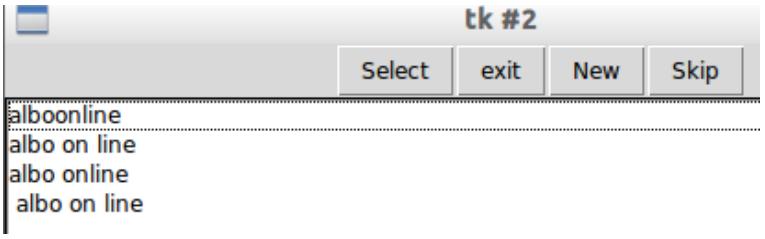
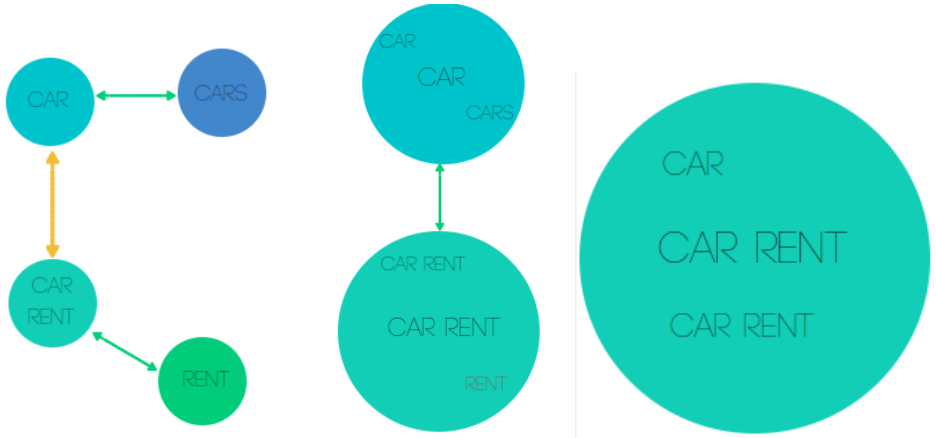


Figure 4.2: Execution of the neighbor method

This method has the big advantage to manage clusters with different radius and cardinality as required from the problem. It is also flexible and robust to lexical errors and fast and simple to implement and use. By adjusting the centroids it could be possible to match and merge cluster that look to be different but have a non empty intersection. Consider for example the items: S=(car, $cars$, $car\,rent$, $rent$) The first two groups will be g_1 =(car,$cars$) g_2 =($car\,rent$,$rent$) with the centroids set to car and $car\,rent$. The second step will merge the two groups previously obtained g_3=(car,$cars$,$car\,rent$,$rent$) with centroid set to $car\,rent$. It is impossible to cluster together $cars$ and $rent$ by only looking at their distance, but with merging neighbor two values that have a big initial distance can converge.



(a) distances calculation (b) join and centroid association (c) clusters merging

Figure 4.3: Example of the execution with 4 strings item

On the other hand it requires a consistent human interaction and is not properly a clustering algorithm but only a suggester for a manual

clustering operation. Unfortunately it also has the drawback of working fine with small strings composed at most by 3 words. The dataset is too much heterogeneous and there exists items composed by more than 20 words. In fact the use of this approach would cause strange results as shown in the picture below.

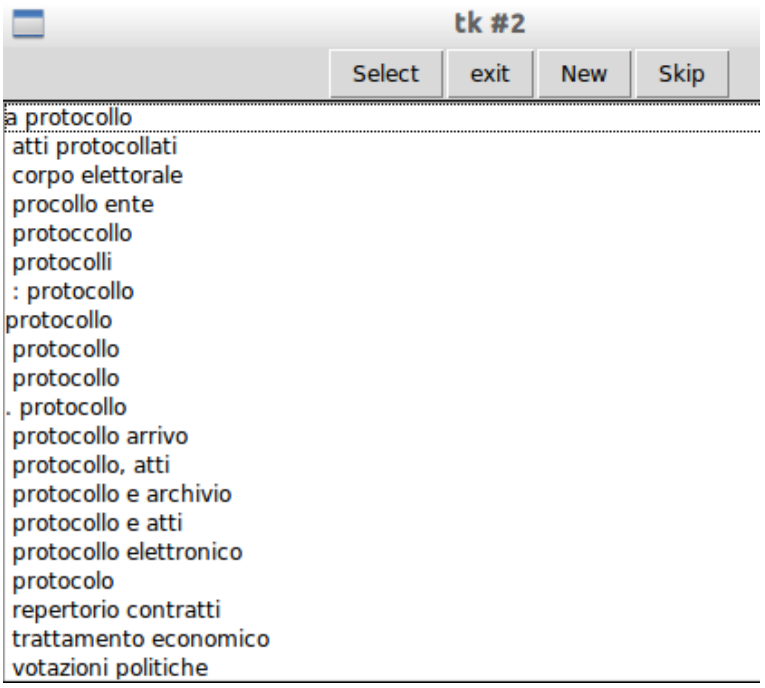


Figure 4.4: Problems with neighbor method

It has been noticed that grater the radius value becomes, smaller the accuracy is. This causes the inclusion of some outliers in the clusters, as shown from the picture. These problems make difficult the use of this strategy, in fact it would be more difficult to manually select the correct clusters than performing the associations without any semi automatic method.

4.2.4 Bag of Words

This approach have been tested in order to deal with the requirement of manipulating strings with different lengths. In fact records look to be very different from each others, there are record composed by only a word (sometimes a stop word) and other that actually are sentences, made of more than 20 words. This make difficult to find a method that efficiently works for each record type. The idea is to consider each record as a document: its content would theoretically describe its origin and nature. Obviously this isn't always true, in fact there are several records that actually don't provides any information or give information that are not related to the considered context. These two properties are also important because they can tell about the presence or not of a "shifted value". So the terms that have a relevancy in the set will be considered as a tag for the item in which they are. The relevancy of a term depends on the number of time that the record appear in the set. Higher is the frequency higher is the relevancy of that tag with respect to the given set. The main problem is to define how to make the tags. Initially the relevant single words have been obtained by stemming each word after the prefilter operation. The stemming operation is necessary to consider the root of the word in order to not make a distinction between different forms of the same word (e.g. a verb like became and `becomes` that are the same verb `become`). Consider the single stemmed words as the tag is a method simple and fast but on the other hand it does not allow to consider common word patterns. This means that there could be sequences of terms $S = t_1, t_2, \ldots, t_n$ that occurs in the set with high frequency. It is reasonable to think that the sequence S is a good tag for the items that contain it. So n-grams structure tags have been defined.

The algorithm for constructing the tags work as follow:

```
Create a tag dictionary dt
for each item
        Create n-words tag where n = # total distinct words in the item
        Look the frequency of the obtained tag in the set
        if the tag occurs for at least $t$ times
                it is inserted in dt
```

The obtained result is a set of tags that have a minimum frequency of $\frac{t}{\sum(distinct\ words\ in\ the\ set)}$. After this phase the list of items is parsed and each one is assigned to a tag. If an item have not been matched from any tag it is marked as "unassigned". Once all tags have been marked, the bags of words of each unassigned item are used to construct a new tagging set. It is constructed by considering $n-1$tags where n is the number of components of the tags constructed in the previous step. Then the unassigned items are assigned to new new tags. This operations is repeated since the tag components cardinality is greater than 1. A the end of these association operations the obtained tags will have different lengths, each item will be marked with the corresponding tag and a list of couples (i, j) where i is the index of the corresponding item and j the index of a tag associated to the tag is kept. The association phase is quite simple: if each token of an item is contained inside a tag it is assigned to that object. At the end of the operations there will be unassigned items.

Table 4.1: Table with some of the most important tags found from the algorithm

Tag	**Matches**
COMUN ANAGRAF AMMINISTR PROTOCOLL PUBBLIC ATT ENT CONTABIL CORRISPONDENT ALB DELIB DOCUMENT INFORMAT FAS DECISION MENS	1160
ANAGRAF ELETTOR ATT CIVIL STAT DEMOGRAF RESIDENT CITTADIN EVENT UFFIC LIST RELAT POPOL	992
PERSON COMUN GEST RELAT GIURID RILEV ASSENZ PRESENZ CONTROLL DIPENDENT TIMBRAT	446
UNASSIGNED	204
STRAD SPECIFIC COMUN CODIC INFRAZION VERB SANZION SICUREZZ VIOLAZION ACCERT POLIZ CONTRAVVENZION	199
PERSON FISCAL PROTOCOLL TRIB DOCUMENT DEMOGRAF SEGRET BILANC AMMINISTR FINANZ	191
PERSON CONTABIL DEMOGRAF BILANC FINANZ	160
CONTRATT ORDIN CONSIGL PROTOCOLL DELIB ATT ENT DETERMIN	134
CULT AMMINISTR PROTOCOLL IMMOBIL ENT CONSUM STAT CONTABIL ACQUIST BILANC ELENC FINANZ MOBIL BEN FORNITOR SCUOL INVENT	136
ONLIN EGOVERNMENT DATABAS PUBBLIC CANON COMUN ALUNN ISTITUZION SIT ALB CONSULT WEB PORT	101

A postfiltering function is executed in order to detect if some tags are subsets of bigger tags. In particular it is possible if all the components of a tag are included in another tag, this is verified if:

$$T_1 \in T_2 \, if \, \forall t_i \in T_1 \rightarrow t_i \in T_2$$

If a tag T_1 is contained in another tag T_2 all elements assigned to T_1 will be assigned to T_2 and it will be deleted from the tags dictionary. Another postfilter step is to check if all unassigned items doesn't actually belong to any tag. Is in fact possible to have items that have not been assigned because their tokens contain errors that have not been corrected yet. The last phase complete the operation by pushing the tag's categories in the macro-categories described above. This is possible by using a list with the tags that are related to each of the 13 categories. This operation have been performed manually. After assigning each tag to a class the record have been finally categorized with maintaining two special categories: the empty (or not valid) records and the unassigned items. The last one have been checked manually in order to find clustering errors. All the steps of lexical error correction have been performed using two different string distance functions: the Jaccard and Damerau-Levenshtein distances. The first distance have the capability to detect the most common subsequences between two strings. It is effective to match two strings that are different but share a big common part. The differences can be associated to lexical errors at the beginning or the end of one of the strings. This similarities function could fail to associate two strings that actually are equal but one of them contains one or more errors ath the middle of the sequence. This is a common error while writing text using a keyboard, in fact it's frequent to find some text

with two or more swapped letters like `airlpane` instead of `airplane`. A good method to solve this problem is to use the Damearau-Levenshtein distance that considers `airlpane` and `airplane` as equal.

4.3 Tables Tipi_licenze_app and Tipi_licenze_db

4.3.1 Table description

These two tables represent information about licenses of applications and databases that have been declared from organizations. In the database have been found several different value forms going from a license number to a license type. This is actually a problem because it makes difficult to extract data and mine information from the table.

Table 4.2: Example of different license format in the table Tipi_licenze_db

Description
EULA (end user license agreement)
EULA (www.microsoft.com)
Free/oracle
GNU/Affero GPL
0707657488

For example, as shown from the picture, an item in which is reported the license number doesn't provide information about the license category or about the producer. It is necessary to define a standard format in which this information is reported and notify whenever a missing piece

of information is found. So a pattern have been defined in order to sort the values and check if there exist some missing parts.

4.3.2 Pattern detection

A pattern is used to categorize the different values that have been found in the database. The pattern that have been chosen with respect to the items is:

<MACRO><ABBR><TYPE><EXP><LICENSE NR.><ORG><LINK>

The blocks represent 6 sections that can occur or not in a record. Their meaning is:

- Macro: means macro category and simply distinguish the free softwares from the ones with fee.
- Abbr: shows the abbreviation of the license type with respect to the product and producer (e.g. MS-PL is Microsoft public license)
- Type: this field express the license type like End user license agreement (EULA)
- License nr.: for certain record the only information expressed is related to the specific number of the license. This field will contain that information.
- Org: means organization, when specified it shows the organization that produced the software.
- link: shows the web reference to the license or the organization.

The blocks have been defined with a set of specials keys used to detect it. The type categories have been obtained by the main licenses available and used from the most famous organizations. Before this phase some of the common lexical errors have been corrected and each record value has been lower cased. Each punctuation or special characters is removed and each record is parsed in order to detect one or more of the blocks.

Table 4.3: Some records after the pattern detection phase

MACRO	ABBR	TYPE	NR.	ORG	LINK
N.D.	MS-PL EULA	Closed Source	N.D.	N.D.	N.D.
N.D.	MS-PL EULA	Closed Source	5329674	microsoft software license account	N.D.
N.D.	MS-PL EULA	Closed Source	N.D.	sistema operativo windows server2003sp2	N.D.
N.D.	N.D.	N.D.	106334	N.D.	N.D.
N.D.	AGPL	Open Source	vers. 3	N.D.	www.gnu.org
Pay	N.D.	Open Source	N.D.	N.D.	N.D.

In some cases the license expiration have been declared. So another field has been added in order to represent this piece of information. This field has been added because some records only specified this information in the value. To standardize the records they have been divided in two categories: `tempo determinato` that means time expiring license and

`tempo indeterminato` that means license without time expiration.

Table 4.4: Example of records with expiration

MACRO	ABBR	EXP
N.D.	N.D.	Tempo determinato
Free	N.D.	Tempo indeterminato
Pay	N.D.	Tempo indeterminato

After the detection phase some items have not satisfied any block and they are marked as "unassigned". While another problem in this dataset is related to the shift error is necessary to detect the possible outliers. To achieve this purpose the idea is to use a method to consider all the records of different tables as similar. So the bag of words system used in the `Soggetti` table have been used. The idea is to construct a tagging list like in the previous table and look if there exist similarities between the two lists. Unfortunately it is necessary to use the whole license set because there could be terms that could be recognized from the first method that actually are outliers. The method would theoretically work better with data not affected from the shift issue.

Table 4.5: Some tags for license records

TAG	NR.
LICENZ APPLIC ESCLUS TEMP FORN DUS ILLIMIT WEB DETERMIN	759
LICENZ APPLIC PROPRIET OPEN SOURC ORACL GPL SVILUPP GRAT BSD HTTP RIUS LICENS PUBBLIC	566
Unassigned	236
LICENZ VERSION GPL AFF HTTP OGGETT LICENS LIB	108
PROGRAMM INFORMAT PROPRIET SVILUPP HALLEY SRL PROTOCOLL INTERN	79
LICENZ APPLIC PROPRIET REGION SVILUPP EMIL RIUS UTILIZZ GRAT AMMINISTR FORN INFORM CONVENZION VENET	75
LICENZ SAAS SERVIC MODAL	49
GESTION	28

The table shows some tags extracted. As expected with only 8 tags (excluding the unassigned) about the 80% of the record refers to license. This is demonstrate from the tag components (e.g. "licenz" is a tag for "license", "open sourc", that means open source represents one of the hot keys for the pattern detection and so on.). One interesting tag shown in the table is the `GESTION` that have been found in about the 20% of the `Soggetti` table. These records would probably belongs to that table. Probably there exists other outliers that belong to other table. It would be necessary to develop a method for detecting the outliers in every table

and understand the table they belong to. A good approach would be to try to construct a tag list for each table in the database. By looking to tags with low frequencies in a table it would be possible to find a table in which that tags would have higher frequency. The idea is that records from any table that are similar are supposed to belong to the same entity. This is not always possible. For instance two records from `tipi_licenze_db` and `tipi_licenze_app` could be very similar, almost equal, and this make impossible to detect an error.

Chapter 5

Conclusions and future works

5.1 Conclusions

The approaches presented in this thesis are actually fuzzy and does not perform an exhaustive cleansing and reordering operation. They have required a remarkable human interaction for constructing the list of terms that were looked to match the correspondences between records. On the other hand they tried to deals with the different errors that usually make difficult the efficient recognition using exact matching techniques like queries. Probably they represent a bootstrap phase that can be used to define a more efficient instrument for recognizing and standardize dirty records. In fact, some important information about the commonly used terms have been constructed by defining the tagging system. It actually represent a term frequency list for the `Soggetti` table. The main problems derive from the synonyms and from the misspelling errors. Ignoring them would make less efficient a possible mining operation. The method proposed required a manual construction of a dictionary of pos-

sible important terms and some lexical errors have not been recognized from approximate matching with different distances function. The use of an alternative structure will probably be more efficient than current approach: for example, a good and simple solution can be represented by the use of a WordNet processor to check synonyms. This will improve the detection and probably could extremely decrease the human interaction in the groups association operation. It would be also important to consider the semantic analysis in order to avoid false positive detection in approximate string matching, for example the word "bean" is similar to the past participle "been" of the verb "to be" but is also an existing and correct word. A possible solution is to understand the meaning of the sentence in which the term is placed and try to perform a prediction for correcting the word or not.

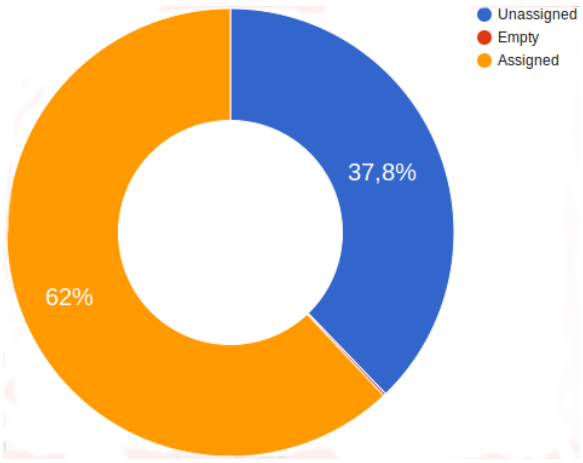
5.2 Results

The results obtained on `Soggetti` table show that using at least 25 tags, almost the 75% of the elements have been associated to a category (about 4445 elements). Considering all the tags the number of elements that have a tag associated to it is about the 95% of all the records (5638). It is interesting to notice that the operation for merging clusters with highest frequency with lower frequency ones produce the move of about 300 elements. This is important while correcting errors and refining the clustering operations because it helps to detect errors that in the previous steps were not identified. The use of non atomic tags have improved the initial results obtained with tag composed by only a term. In fact is possible to see that the number of unassigned elements decrease while

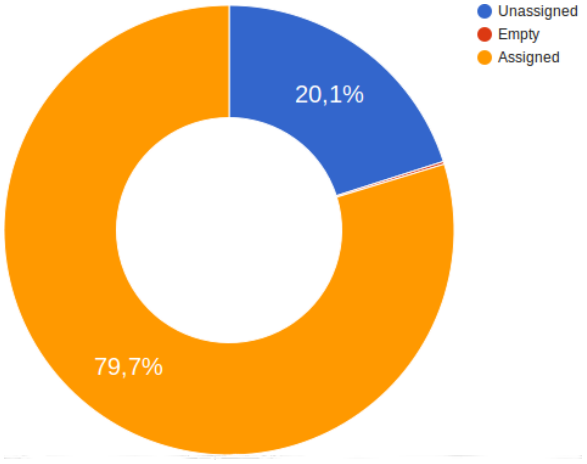
the number of components per tag increase.

Figure 5.1: Decreasing of unassigned elements with different tag lengths

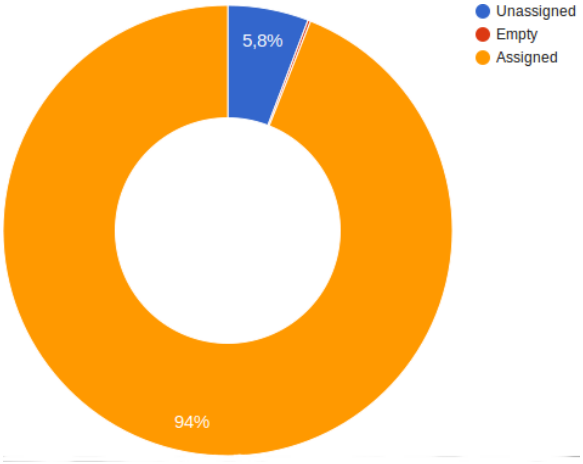
The results shows that starting from 2 components tags and going to tags composed by more than 4 tags, the number of unassigned elements decreases by almost the 90%. This is possible because the tags become less specific and allow to assign them to more elements. In fact the assigned elements graphs are shown below.



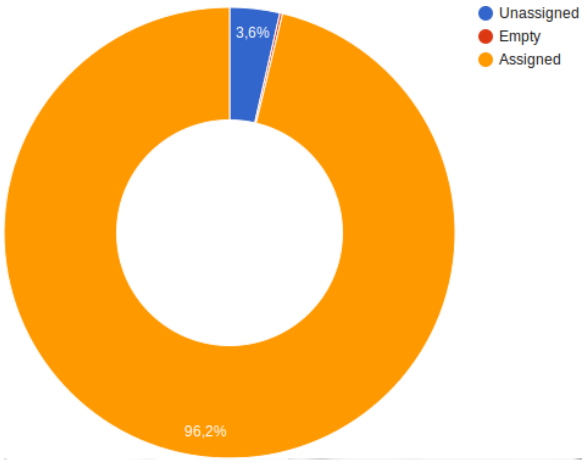
(a) Assignment with 2 components tags



(b) Assignment with 3 components tags



(c) Assignment with 3 components tags



(d) Assignment with multiple components tags

Figure 5.2: Example of the execution with 4 strings item

The results about the `tipi_licenze_app` shows that 1196 records haven't any field declared from the pattern defined. This means that about the 51 % of the records have not been recognized. This result isn't

good as the one obtained on `Soggetti` table. It is necessary to consider that there was about 20 empty records. By inspecting the records making a tag list is possible to notice that there were only 236 unassigned records. This shows that there could exists several records that do not fit the patterns defined. This can be related to the presence of several outliers or a non exhaustive definition of the pattern rules.

Table 5.1: List of tags with more than 20 occurrences

TAG	NR.
LICENZ APPLIC ESCLUS TEMP FORN DUS ILLIMIT WEB DETERMIN	759
LICENZ APPLIC PROPRIET OPEN SOURC ORACL GPL SVILUPP GRAT BSD HTTP RIUS LICENS PUBBLIC	566
Unassigned	236
LICENZ VERSION GPL AFF HTTP OGGETT LICENS LIB	108
PROGRAMM INFORMAT PROPRIET SVILUPP HALLEY SRL PROTOCOLL INTERN	79
LICENZ APPLIC PROPRIET REGION SVILUPP EMIL RIUS UTILIZZ GRAT AMMINISTR FORN INFORM CONVENZION VENET	75
LICENZ SAAS SERVIC MODAL	49
SISS APPLIC RETRIB CONTABIL ANAGRAF PRESENZ RELAT SCUOL PERSON ECONOM	48
LICENZ ASSISTENT CANON CONTRATT DOMIN ANN PAG	48
CONTIEN ATT APPLIC ATTRIB INFORM LEG INPS RELAT BILANC PROCED ARCHIV	40
LICENZ APPLIC COMUN TRIB PUBBLIC CONVENZION SVILUPP PROPRIET CONTABIL ENT LOC REALIZZ	33
GESTION	28
LICENZ ITAL UTENT ARG SRL	28

From the list of all the tags there exists some tags that have a strong relation with the `Soggetti` table like the ones shown below.

Table 5.2: List of tags with more than 20 occurrences

TAG	**NR.**
GESTION	28
DOCUMENT PROTOCOLL AMMINISTR SISTEM	13
CLOUD	8
ALUNN	6
MINIST	5
PROVINC	5
FISC	4
ACQUIST	4
SERV	4
ROM	4
REGISTR	3
ENTRAT	3
CLASS	3
MAGGIOL	3
SID	3
BEN	2
AUTORIZZ	3
INTERNET	2
GEST	2
Unassigned	236
Empty string	21
NESSUN	7

Better results have been registered by the `tipi_licenze_db` table that has about 4000 records. After the cleansing operations the number of items without any assigned block is less than 1400 items, that is close to the 35% of the total records. This can be considered as a good result because by applying the same algorithm used in the `Soggetti` table is possible to notice how the number of unassigned record is grater than 500 elements. Some elements are probably record that only contains a license number, but the other items are probably related to the `Soggetti` table. In fact record like "consultazioni elettorali" or "patrimonio ed inventario" have been frequently found in that table. Another evidence is represented from the presence of some records that contains the token "DLG" that is normally related to laws. This could demonstrate that most of the uncategorized records would probably belongs to other tables.

5.3 Possible improvements

As previously written the methods presented in this thesis need to be improved and integrated in order to make them more flexible and effective. One possible improvement will certainly be the use of a WordNet processor with an own dataset of synonyms. This will both resolve the problem of ambiguous words and will make possible to consider two synonyms as the same term. This will dramatically improve the effectiveness of these methods. Once constructed a token frequency list it would be interesting to use some statistical models to predict future record insert. This will provide a probabilistic error avoidance system. On of the most important improvement that will be important to achieve the purpose of record reordering, in relation to the problem of the record shift will

be represented by a method to "find and swap" record values. This is a shift with respect to a pivot column. The main idea is to recognize the items that contain values that are considered outliers for that table and compare that values with other table's tags. If a value is similar to a class of tags of that table it will probably belong to it. So the two records are swapped. The value swapped is compared to its new table's records list. If it's an outlier it is compared with the other tables in order to find a table that it can belong to. If a table is found, the values of the record are swapped and so on. This method is similar to a simple sorting algorithm for array structures. The main difference is that the measure used to move the values is related to the tags' list of each table.

Appendices

Some of the results presented in the last chapter have been reported in this section like the tag lists for the 3 tables that have been threated and examples of uncategorized elements.

.1 Soggetti table

Table 3: Example of unassiged elements and the associated tokens

Tokens	Category
acc, sp	Unassigned
accessibil	Unassigned
guard, nazion, repubblican	Unassigned
imprenditor	Unassigned
indicentist	Unassigned
infocam	Unassigned
infocam, scp	Unassigned
infocim	Unassigned
information, technology	Unassigned
istruttor, educ	Unassigned
labanalis, lagonegr	Unassigned
fruibil, dat	Unassigned
lp, 7, 11, 1983, n41	Unassigned
lp, n, 41, 1983	Unassigned
webgis	Unassigned
xml, avcp	Unassigned

These are examples of the records marked as unassigned. There are some outliers, like the ones related to laws, some record with a bad value and records that were not similar to other in the set.

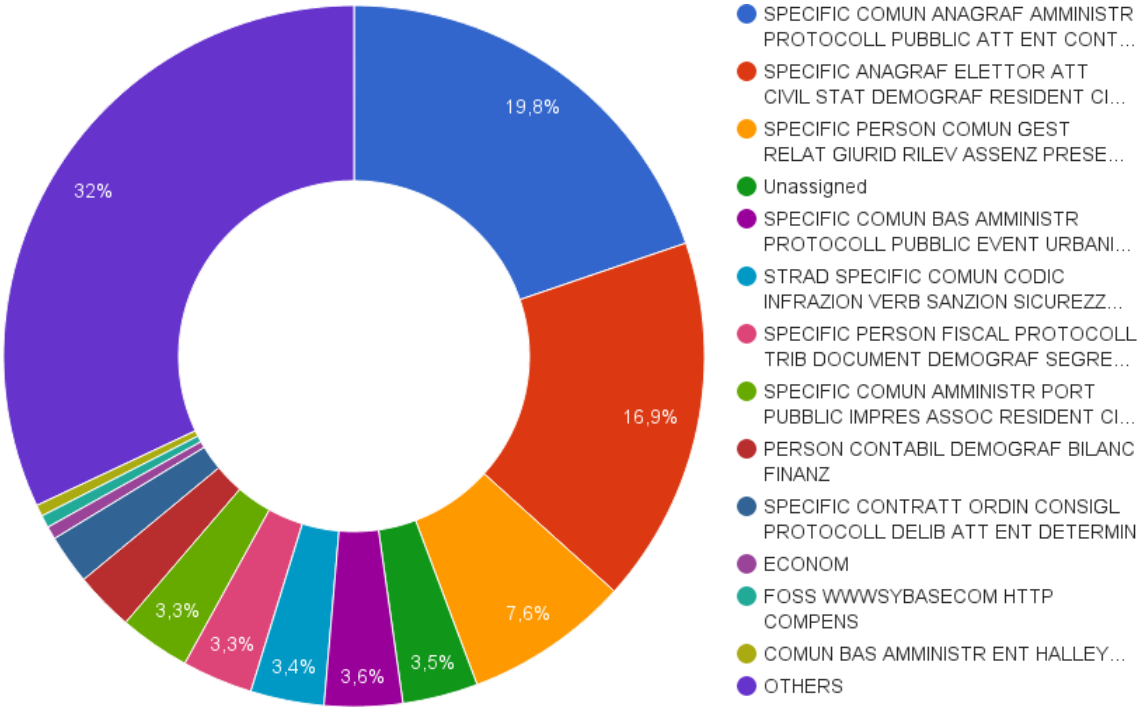


Figure 3: Tags frequency distribution

.2 tipi_licenze_db table

Table 4: Example of unassiged elements and the associated tokens

Tokens	Category
000939	Unassigned
0015604e198	Unassigned
02963	Unassigned
0383, tops190009	Unassigned
0707657488	Unassigned
0707753549	Unassigned
1	Unassigned
102075	Unassigned
10416	Unassigned
104709	Unassigned
104728, 18122009	Unassigned
contravvenzion	Unassigned
delib, determin	Unassigned
demograf	Unassigned
digitalizz	Unassigned
sanzion, codic, strad	Unassigned
ambient, territor	Unassigned
ambitosocialesinpnet	Unassigned
autodesk	Unassigned
axes	Unassigned
axiosdatabas	Unassigned
axs01215	Unassigned
ben, cult, turism	Unassigned
campus	Unassigned

Most of the records in the table are divided in 3 categories:

- licenses number: they have not been recognized because they are unique and does not match any terms found

- dirty records: a record composed only by stopwords or a single letter or number, it actually will be an invalid value for the record

- outliers: a lot of record refer to tags of the `Soggetti` table, in practice they are outliers.

Table 5: The most frequent tags for the table

TAG	Nr
HTTP FOSS SYBAS WWWSYBASECOM ORACL PRODUCTSPECIFICLICENSETERM PRODUCTSPECIFICLICENSETERMS SOFTWARELICENS LICENSIN	816
Unassigned	522
GPL LICENZ LICENS CLIENT PUBLIC INTERBAS LICENC BAS INREBAS	485
HTTP HALLEYWEBCOM INDEXPHP TRASPARENT EGOVERNMENT HTTPS WWWHALLEYWEBCOM	307
LICENZ FORN ORACL CONTEN APPLIC COMM STANDARD PROPRIET SRL ITAL GRAT	319
HTTP FOSS SYBAS ORACL EXCEPTION WWWMYSQLIT WEB LICENSIN	255
HTTP HALLEYWEBCOM HTTPS INFORMAPHPX= WWWHALLEYWEBCOM OPENCMS COMUN	206
LICENZ SYBAS CAL PRESS EDITION EXPRESS STANDARD SERV SQL MICROSOFT DBMS	175
HTTP GPL LICENZ SOURC RIUS FIREBIRD EDITION PUBLIC LICENC LICENS MICROSOFT OPEN	114
HTTP ABOUT LEGAL WWWMYSQLIT LICENSING LICENC LICENS	110
POEXLOGINPHP HALLEYWEBCOM POLOGINPHP INDEXPHP LOGINPHP HTTPS WWWHALLEYSACIT COMUN	63
LICENZ LIC ACCESS FIREBIRD EXPRESS SQL ENGIN	43
LICENZ NESSUN PROGRAMM APPLIC RILASC ALUNN PROPRIET	42
LICENZ SRL SCUOL ARG WEB SID ALUNN WIN	43
HTTP GPL PUBLIC VERSION LICENS OPEN	30
LICENZ UTILIZZ EDITION HALLEY STANDARD MICROSOFT	31
REGION	9
MINIST	7

This table show how most of the more frequent tags are related to licenses. The “Unassiged” category is not sufficient to explain the high number of not defined record. This because the first table shows that several records marked as “unassigned” are actually licenses. This situation shows an important drawback of the tags system: Invalid records have been picked up from the tags related to licenses. This could happen when two different records share a common element (like an URL).

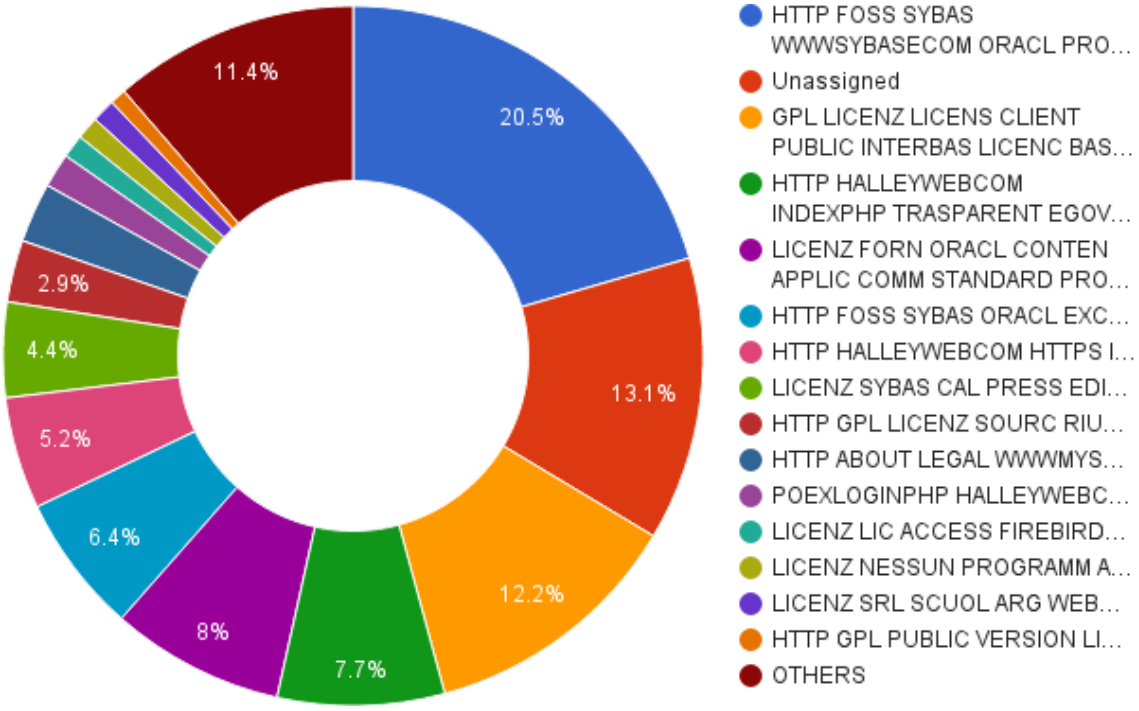


Figure 4: Tags frequency distribution

.3 tipi_licenze_app table

Table 6: The most frequent tags for the table

TAG	Category
0	Unassigned
0383, tops190009	Unassigned
4c8tptfa23, 8gj9d346n2, tu9dy5x8w1, qxj4vb1s4y	Unassigned
51190	Unassigned
51191	Unassigned
51192	Unassigned
51193	Unassigned
51195	Unassigned
51196	Unassigned
903	Unassigned
caric, serenissim, ristor, sp	Unassigned
caric, fornitor	Unassigned
accert, ic	Unassigned
accredit, asl	Unassigned
contravvenzion	Unassigned
b50036726985	Unassigned
gismast	Unassigned
giustiz	Unassigned
glocalval	Unassigned
gn, gpl2	Unassigned
gn, gpl3	Unassigned
hp	Unassigned
i8ly1cm0	Unassigned
n	Unassigned
nd	Unassigned
wwwsagait	Unassigned
wwwseacit	Unassigned

Unlike the the `tipi_licenze_db` table this looks to have more outliers that actually have been marked as "Unassigned". There also are fake positive records, for example the ones with the license numbers or some records showing a correct value.

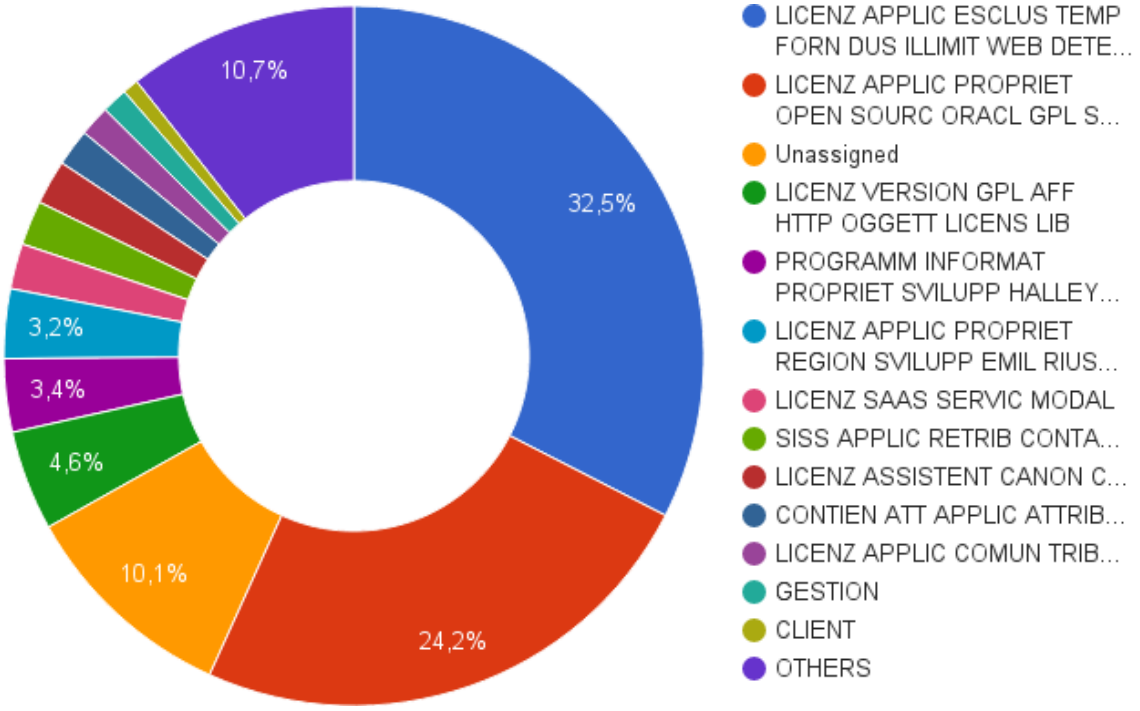


Figure 5: Tags frequency distribution

Bibliography

[1] Agenzia per l'italia digitale project. http://www.agid.gov.it. Accessed: 2016-02-07.

[2] Law for italian organizations transparency. http://www.gazzettaufficiale.it/eli/id/2014/08/18/14G00129/sg. Accessed: 2016-02-07.

[3] Openrefine docs - clustering in depth. https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth. Accessed: 2016-02-07.

[4] Openrefine website. http://openrefine.org/. Accessed: 2016-02-07.

[5] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules.

[6] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[7] John G Cleary and Ian H Witten. Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions on*, 32(4):396–402, 1984.

[8] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[9] Xiaohui Cui, Thomas E Potok, and Paul Palathingal. Document clustering using particle swarm optimization. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 185–191. IEEE, 2005.

[10] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 541–552. ACM, 2013.

[11] Evangelos Dermatas and George Kokkinakis. Automatic stochastic tagging of natural language texts. *Computational Linguistics*, 21(2):137–163, 1995.

[12] Wenfei Fan, Floris Geerts, and Xibei Jia. A revival of integrity constraints for data cleaning. *Proceedings of the VLDB Endowment*, 1(2):1522–1523, 2008.

[13] Patrick AV Hall and Geoff R Dowling. Approximate string matching. *ACM computing surveys (CSUR)*, 12(4):381–402, 1980.

[14] In-Won Kim, Mun Sik Kang, Sunwon Park, and Thomas F Edgar. Robust data reconciliation and gross error detection: the modified mimt using nlp. *Computers & chemical engineering*, 21(7):775–782, 1997.

[15] Jonathan I Maletic and Andrian Marcus. Data cleansing: Beyond integrity analysis. In *IQ*, pages 200–209. Citeseer, 2000.

[16] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[17] Mario Mezzanzanica, Roberto Boselli, Mirko Cesarini, and Fabio Mercorio. Improving data cleansing accuracy-a model-based approach. In *DATA*, pages 189–201, 2014.

[18] Heiko Müller and Johann-Christph Freytag. *Problems, methods, and challenges in comprehensive data cleansing.* Professoren des Inst. Für Informatik, 2005.

[19] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[20] Patrick Ruch, R Baud, and Antoine Geissbühler. Evaluating and reducing the effect of data corruption when applying bag of words approaches to medical records. *International Journal of Medical Informatics*, 67(1):75–83, 2002.