UNIVERSITY OF THE BASQUE COUNTRY

MASTER DEGREE THESIS

Accommodations Deduplication

Author:
Francis PEREZ

Supervisor:
Aitor SOROA
Oier LOPEZ DE LACALLE

Master in Computational Engineering and Intelligent Systems

Computer Science and Artificial Intelligence Department

September 24, 2018

UNIVERSITY OF THE BASQUE COUNTRY

Abstract

Faculty of Informatics
Computer Science and Artificial Intelligence Department

Accommodations Deduplication

by Francis PEREZ

The problem to address is the accommodations deduplication. The deduplication is a special case of entity resolution (ER) consisting in grouping different representations of the same entity, usually coming from different sources. The deduplication is a complex process that requires several phases, being the most common ones, blocking and pair resolution. A new phase is introduced in addition to the previous ones, clustering, that was not considered in previous work. We aim to build a framework able to cover the different phases and design a strategy of clustering maximizing the precision with the maximal possible recall.

Acknowledgements

Many thanks to Aitor and Oier for their support, help and patience. Thanks also to Manuel Sanchez, Hector Barriuso and Julen Telleria for being my official testers / reviewers.

Contents

List of Figures

List of Tables

List of Abbreviations

ER	**E**ntity **R**esolution
GMD	**G**eneralized **M**erge **D**istance
IQR	**I**nter **Q**uartile **R**ange
kNN	**k**-**N**earest **N**eighbours
ML	**M**achine **L**earning
OTA	**O**nline **T**ravel **A**gency
PHash	**P**erceptual **Hash**
RF	**R**andom **F**orest
SGD	**S**tochastic **G**radient **D**escent
TF-IDF	**T**erm **F**requency - **I**nverse **D**ocument **F**requency
VI	**V**arition of **I**nformation
XGB	**X**treme **G**radient **B**oosting

Chapter 1

Introduction

1.1 Problem Description

The problem to address is the accommodations deduplication. We will define deduplication as:

Definition 1. *Deduplication is a special case of entity resolution (ER) consisting in grouping different representations of the same entity, usually coming from different sources.*

Other definitions that we can find are:

"A crucial step in integrating data from multiple sources is detecting and eliminating duplicate records that refer to the same entity. This process is called deduplication" (Sarawagi and Bhamidipaty, 2002)

"Entity resolution (ER), the problem of extracting, matching and resolving entity mentions in structured and unstructured data, is a long-standing challenge in database management, information retrieval, machine learning, natural language processing and statistics. Ironically, different sub-disciplines refer to it by a variety of names, including record linkage, deduplication, co-reference resolution, reference reconciliation, object consolidation, identity uncertainty and database hardening" (Getoor and Machanavajjhala, 2012)

"We consider the entity resolution (ER) problem (also known as deduplication, or merge–purge), in which records determined to represent the same real-world entity are successively located and merged" (Benjelloun et al., 2009)

As we can see there are several interpretation regarding the **ER/deduplication/Record linkage** relationship, some consider them the same. But others **not**.

"From the result form of entity resolution, it could be classified into two types. One is pair-wise entity resolution. The results are pairs of data objects which refer to the same real-world entity. The other is group-wise entity resolution, whose result is a family of clusters with each one containing the data objects referring to the same real-world entity" (Wang, 2014)

In this case, record linkage and deduplication are considered special cases of ER where record linkage refers to the ability of relating two different entities and deduplication the fact of grouping all the equals' entities. Our Definition 1 is closer to this one.

1.1.1 Example

When a user searches for a specific hotel in Google, in some cases Google will show in a separate widget the results with the accommodations from different sources like figure 1.1.

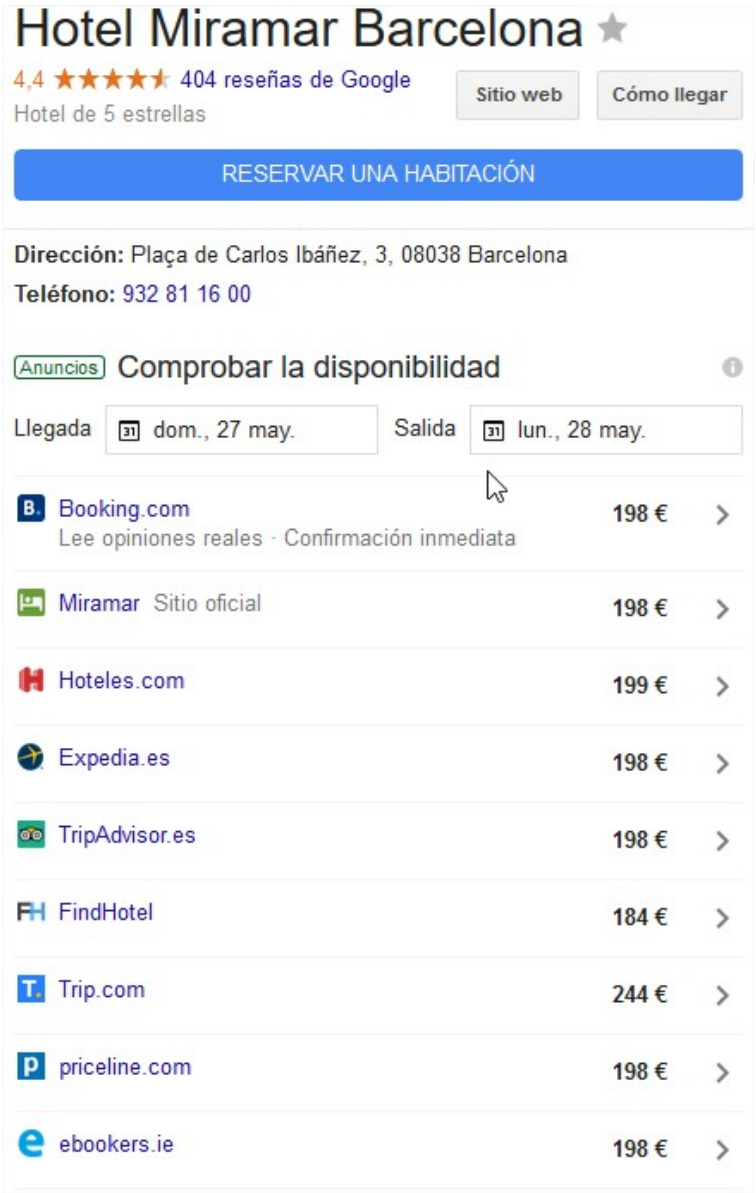


FIGURE 1.1: Google results widget for "hotel miramar" search criteria

The widget in figure 1.1 is shown after searching for "hotel miramar" in google. We can say that somehow and according to figure 1.1 Google has been able to deduplicate the accommodation referencing "hotel miramar" from the following sources:

- Booking.com
- Official website
- Hoteles.com
- Expedia.es
- TripAdvisor.es
- FindHotel
- Trip.com

- priceline.com
- ebookers.ie

We can expect that the different accommodation results share common elements among them or have similarities. The idea behind deduplication is that different sources may have different, but similar, attributes. If we consider the name and address from some of the different links provided by Google we will get the table 1.1.

id	source	name	address
1	Booking.com	Hotel Miramar Barcelona GL	Plaza Carlos Ibañez, 3
2	Official website	Hotel Miramar Barcelona	Plaça de Carlos Ibáñez, 3
3	Hoteles.com	Miramar Barcelona	Placa Carles Ibanez 3

TABLE 1.1: Name and address of accommodation by source

We can see in table 1.1 that the names are quite similar between them but not the same. The same happens with the address. It is not always going to be possible to find a strong correlation between attributes. Sometimes, some of them could be similar and others not, other times is a combination of several attributes what gives enough information to be able to deduplicate, etc.

1.2 Motivation

The deduplication acquires relevance in e-commerce Company that offers accommodations. We can identify mainly two types of those e-commerce companies. First, we have the *Online Travel Agency* (OTA) that allows the user to book an accommodation. The OTA will select what entity representation will be shown from the different providers/sources representing the same accommodation. On the other hand, we have the *metasearch engine* or *aggregator*, a metasearch engine will allow the user to continue the booking process in one of the different providers. The main difference between them regarding the deduplication information is that the metasearch engine will usually show the different options and the OTA won't. In the Google example, we can consider Google as a metasearch engine. Booking.com is an OTA.

From an OTA perspective, the deduplication process can be important because:

- Improves the user experience: The users don't have the same offer several times from different sources. The OTA select for them the "best" one.
- Contractual obligations: Agreements can be reached to offer some accommodations in exclusivity. That means that a specific source must be selected when there are duplicates.
- Optimizations: The final goal is to maximize the conversion and revenue; different logic can be implemented in order to decide what accommodation from what provider is shown to the user in case of duplicates.
- Enriched Content: More content for the same accommodation is available when we have more than on provider.

The first point also applies for the metasearch. The deduplication enables the functionality of showing the different options for the same accommodation. In this case, option means provider/source [1]. We have mentioned Google but a vast number of meta-searchers and OTA exhibit a similar behavior and hence they have a "deduplication process" in place.

We must be aware that there are several alternatives or ways of having a deduplication process, for example getting the information from outside, buying the data, deduplicating manually or automatically. In this work, we present an automatic deduplication system based on machine learning.

1.3 Goals

Having a set of entries representing an entity with a set of attributes like in fig 1.1 we must be able to group the different representations related to the same entity. The goals are:

- Building a generic framework/system in order to help in the deduplication process.
- Apply the framework to the specific case of accommodation deduplication.
- Generate a resolution strategy with a precision near to 100% and maximal possible recall.

In order to do that we had to:

- Build a binary classifier for accommodation comparisons
- Study and implement attributes for the classification
- Implement global techniques of assignment and resolution of inconsistencies

[1] we will use both terms indistinctly.

Chapter 2

State of the Art

Most of the work related to deduplication is focused in efficient methods of grouping entities minimizing the number of comparisons in order to detect duplicates (Benjelloun et al., 2009; Köpcke and Rahm, 2010; Christen, 2012). But this can be considered only one of the different common phases within the deduplication process.

2.1 Deduplication Framework

The deduplication process usually consists in different phases. These phases sometimes are clearly differentiated from an implementation point of view and other times are not, they mix. We won't cover all the possibilities because it is out of the scope of this work, these are the most significant phases:

2.1.1 Blocking

Blocking makes reference to the fact of grouping the records into blocks (Whang et al., 2009). The intention is to reduce the number of comparisons to do. If we have, let's say, 3 million of accommodations and we decide to check any possible pair, we would have a big number of comparisons. Exactly the combination of 2 elements chosen from 3000000 without repetitions $\binom{3000000}{2}$.

Doing a comparison and evaluating it has a cost. There is a need to reduce it as much as possible without loosing relevant comparisons.

If we create blocks of, let's say, 1000 elements from those 3000000, we could reduce the number of comparisons to: $3000 \times \binom{1000}{2}$. Ideally, all of those 3000 blocks should maximize the probability of any comparison between the elements of the block to be a duplicate. Those 3000 blocks in this example are disjoint, but the blocking is not restricted to that, the comparison could be overlapping, meaning that the same element or entity representation could be inside more than one block (Köpcke and Rahm, 2010).

For our purpose, we are going to redefine the term blocking as:

Definition 2. *Blocking is any indexing mechanism with the purpose of reducing the number of comparisons.*

We redefine it because the term "block", as a noun, give us a notion of something already grouped and not something dynamic. The usual interpretation is that we "block", as verb, elements and in order to know if two elements are comparable we ask if they belong to the same block, so we can compare freely any two elements members of the same block.

Our comparison is done based on a predicate that states what conditions two entities have to meet in order to be comparable (blocking predicate or membership predicate).

Our proposed specification for the blocking predicate would be:

Being S our initial set of entities, f_b the blocking predicate. We could say that our comparisons will be done over the set:

$$\forall x \forall y (x \in S \wedge y \in S \wedge x \neq y : f_b(x, y))$$

For example:

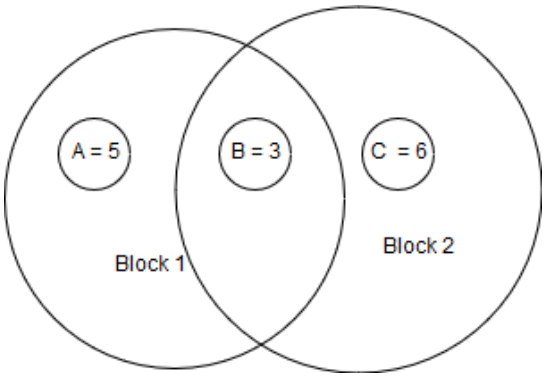


FIGURE 2.1: Overlapping blocks

In figure 2.1 we have 3 elements: A, B, and C. And 2 blocks: "Block 1" and "Block 2". Both blocks share the element B. We can compare A and B because belong to the same block and we can compare B and C because both pertain to Block 2.

For example, if any single element A, B, and C has an attribute value, for $A = 5$, for $B = 3$, for $C = 6$. We can reach the same block scheme by stating:

Any pair with a sum of their values $<= 9$ *pertain to the same block.* Or *Any pair with a sum of their values* $<= 9$ *can be compared*.

The last statement could be translate as: being f_v a function applied to an element and returning its value, our definition of the blocking predicate will be:

$$f_b = f_v(x) + f_v(y) <= 9$$

2.1.2 Matching

Given a single comparison between entities, we must decide whether those entities represents the same concept or not. We will make use of a machine learning based model in order to decide this. As said before, that is our core approach.

We will have a matching commutative function f_m that receives two entities e1 and e2. That function will return true indicating that those entity representations are duplicates or false otherwise.

For example for figure 2.1 f_m will be invoked with f_m(A, B) and f_m(B, C).

In our work, we consider the case where the matching function uses a pre-built classifier in order to determine the result (duplicate or not duplicate). But usually a classifier receives a record R and returns the class (true or false) for that record:

$$f_c : R \rightarrow \{true, false\}$$

As we have two parameters e1 and e2, we need a function f_d, also commutative, that maps e1 and e2 to a record (r). It maps at an entity level, not at attribute level, that means that several attributes from the entity could be mapped to a unique attribute in the record. The opposite is also true; the deriving function could produce several record attributes based in a unique entity attribute.

The merge function will be the composition of the deriving function and the classifying function.

$$f_m = f_d \circ f_c$$

Feature Extraction

In the examples from Table 1.1[1], every row contains the following data: id, source, name and address. We will consider every row an entity representation. We can do up to 3 comparisons between them: 1 and 2, 1 and 3, 2 and 3.

A record r is a set of different attributes and their values. f_d must convert the entity information: source, name and address, from entity 1 and entity 2, to a set of attributes and values. Those attributes should give a "hint" to f_c in order to classify as duplicate or not duplicate.

When speaking about comparisons there are two special case of measures:

- Similarities: a value that give us a sense of how close are the elements being compared.
- Distances: It is the opposite of similarity.

For example, let's suppose that we have the entities in Table 2.1 with the attributes latitude and longitude.

id	latitude	longitude
e1	41.379707	2.176595
e2	41.37974	2.1765547

TABLE 2.1: Geo entities information

The resulting record from comparing e1 and e2 could have a unique attribute "distance in meters". Table 2.2. We say that we derive the attribute distance from the latitude and longitude information in the entities.

record	distance
r1	4.97

TABLE 2.2: Geo entities record

Classification

Our classifier has to be trained in order to be able to classify unknown cases.

id	name
1	Hotel Miramar Barcelona GL
2	Hotel Miramar Barcelona
3	Miramar Barcelona
4	Fake

TABLE 2.3: Entities name

[1] In page 3.

Let's suppose we have the entities in Table 2.3. All of them belong to the same block so we are going to do all the possible comparisons. only entities 1, 2 and 3 are duplicates. The f_d will compute the Levenshtein distance of the name(Ristad and Yianilos, 1998). By the moment we don't need to understand the calculation of the Levenshtein distance, we only need to remember the meaning of *distance* and take into account that a greater value means a greater difference (dissimilarity) between the elements being compared.

id1	id2	levDistance	duplicate
1	2	3	true
1	3	9	true
1	4	24	false
2	3	6	true
2	4	21	false
3	4	15	false

TABLE 2.4: Matching example

With the *levDistance* attribute (result of the Levenshtein distance calculation) and the class attribute duplicate in Table 2.4 the classifier will be able to learn a function f_c that given a record with the attribute *levDistance* will return true or false, meaning it is a duplicate or not.

Of course, this is a simplification, and in this case we could think in a fixed function like

$$f_c(e1, e2) = distance(e1, e2) < 15$$

No need for a machine learning based classifier for this simple case. But what will happen if the "rule" changes frequently? Are we going to change it accordingly any time new examples are given? What will happen if we have many data? ? Tons of examples with many different attributes? What about table 2.5?

name1	name2	levDistance	duplicate
hostal q	hostal 56	2	false
hotel acta azul barcelona	acta azul	16	true

TABLE 2.5: Name levenshtein distance

Table 2.5 will be hard to learn also from a ML (machine learning) perspective. We will need more "hints", we will need more or different features, maybe the address distance? The geographical distance? The type of accommodation?. We will treat it later.

2.1.3 Clustering

"Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters)" (Wikipedia contributors, 2018a)

In this work we will speak frequently about "clustering" as a different phase, but "clustering" as a phase is introduced in this work, no references has been found in previous papers.

Once we have all possible comparisons, how do we group those entities in order to conform the final cluster? Sometimes depending on the resolution strategy this point can be included within the classification part, any classified pair would be immediately grouped within a cluster.

Let's suppose that Table 2.4 is the result of our deduplication process until the matching phase. The relationship between entities can be modeled as a graph. An edge meaning that the two vertex are duplicates and the absence means that they are not duplicates.

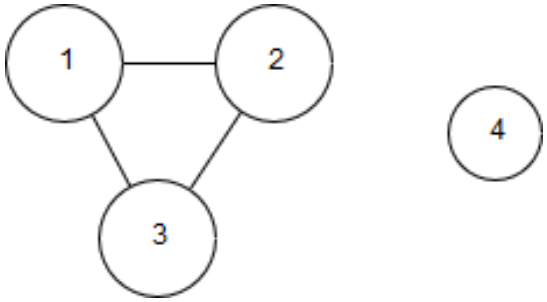


FIGURE 2.2: Example cluster 1

The cluster is the grouping where all of the elements represents the same real world entity, so the cluster itself is the real world entity and the elements are, let's say, a projection of a part of the information of the entity. Those elements refers to only one real entity, so they can only be in one cluster.

In the figure 2.2 we can make two clusters: {1, 2, 3} and {4}. It is a easy case, but what about something like figure 2.3?

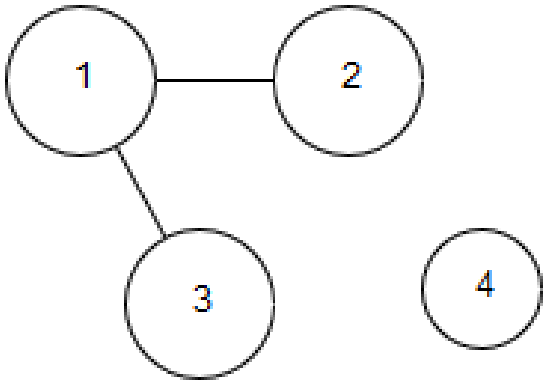


FIGURE 2.3: Example cluster 2

We could say that there are 3 clusters {1, 2}, {1, 3} and {4}, but as per our definition that can't be possible, the element 1 cannot be part of two clusters. We could assume transitiveness and group it like: {1,2,3} and {4} or maybe we could even group it as: {1}, {2}, {3} and {4} or {1,2}, {3} and {4} or {1}, {2, 3} and {4}.

The clustering phase deals with those results and the grouping in order to get the closest configuration of the real clusters.

2.2 Regarding Accommodations Domain

There are several references to hotel deduplication and local premises deduplication as a more generic scope (Zheng et al., 2010; Benjelloun et al., 2009; Dalvi et al., 2014).

The most explicit one, (Kozhevnikov and Gorovoy, 2016), makes a comparative from the business point of view and the different costs for the solutions analyzed.

In the technical part, it mentions the features used, but provides little detail about the implementation of the different phases and there is no metric at a cluster level.

In the following points we will treat the approach used in (Kozhevnikov and Gorovoy, 2016) for the different phases.

2.2.1 Blocking

They divide the world in regions of a fixed size and make comparisons within the same region and adjacent. There is no details about the implementation. The number of comparisons gathered were 9193 with 12 providers.

One of the features/attributes they use is region density, which only has sense when there is a previous grouping like this.

2.2.2 Classification

They used a random forest with 30 decision trees and 22 features grouped by:

- based in name
- based in geographical location
- based in images
- based in other features

They achieve a result of a 99.1% precision and 98% recall. Those values can be considered a baseline to compare with our classifier performance. Of course, we don't know the exact set of data at their disposition but is the most similar work that we can compare with.

2.2.3 Clustering

No reference about clustering is provided. Possibly, they make the assumption of transitiveness that we will treat later.

Chapter 3

Methodology

As described in the previous chapter, deduplication process includes several steps in order to get a deduplication for a set of entities representation, this chapter is structured around those steps and their order.

1. First of all, before starting the process we need a preprocess of the available data that will serve as input for our deduplication process.
2. The process starts with the blocking step in order to reduce the number of comparisons as stated in section 2.1.1.
3. Once we have the pairs of entity representations we need a set of features to feed our machine learning model.
4. The trained model is used to make predictions.
5. The previous prediction are used to cluster the entities representation being evaluated.
6. We evaluate the quality of the predictions and cluster results.

3.1 Data Description & Preprocessing

The basic input data includes the fields:

- **address** the physical address of the accommodation.
- **supplierCode** the provider/distributor.
- **city** the city where the accommodation is located, can be considered as an extension of the address.
- **name** name of the accommodation.
- **coordinates** geographical location of the accommodation, it is composed by latitude (lat) and longitude (lon).
- **chain** owners of the accommodation.
- **stars** makes reference to the accommodation category, from 0 to 7.
- **type** type of the accommodation. The different types of accommodation are:
  - Hotel
  - Apartment

  - Hostel
  - Guest House
  - Bed and Breakfast
  - Resort
  - Residence
  - Motel

- **images** url's to the provider images.
- **id** unique identifier for an accommodation in a provider.

With the previous information we have something in the following format:

```
{
    "address" : "placa carles ibanez 3",
    "supplierCode" : "***",
    "city" : "barcelona",
    "name" : "miramar barcelona",
    "coordinates" :
            "lat" : 41.37074,
            "lon" : 2.17041
    } ,
    "chain" : "***",
    "stars" : "5",
    "type" : "Hotel",
    "images" : [ ],
    "id" : "1"
}
```

LISTING 3.1: Basic Information Example

The *supplierCode* and *chain* are omitted intentionally because of confidentiality.

3.1.1 Address Parsing

We use external address formatters, google and libpostal. These formatters are able to standardize the address and to extract the different components of that address. The main observed differences between them are that libpostal in some cases is language sensitive. It doesn't return the address formatted exactly the same way when the addresses are in different languages (Euskara or Catalan for instances). However, google geocoder has more restrictive uses conditions. We add to the basic information:

- **expandedAddress:** The libpostal formatted and standardized address
- **road:** the name of the street according to libpostal
- **houseNumber:** number of the address building according to libpostal
- **level:** floor in the building according to libpostal
- **unit:** within the building and level, the unit of the address according to libpostal

- **googleFormatted:** the google formatted / standardized address
- **googleHouseNumber:** the house number according to google.

The extended data would be:


```
{
    "address" : "placa carles ibanez 3",
    "supplierCode" : "***",
    "city" : "barcelona",
    "name" : "miramar barcelona",
    "coordinates" :
            "lat" : 41.37074,
            "lon" : 2.17041
    } ,
    "chain" : "***",
    "stars" : "5",
    "type" : "Hotel",
    "images" : [ ],
    "id" : "1",
    "expandedAddress" : "placa carles ibanez 3",
    "road" : "placa carles ibanez",
    "houseNumber" : "3",
    "level" : "",
    "unit" : "",
    "googleFormatted" : "Plaça de Carlos Ibáñez, 3, 08038 Barcelona, Spain",
    "googleHouseNumber" : "3",
}
```


LISTING 3.2: Extended Data Example

Libpostal

We don't use directly libpostal but a wrapper: libpostal-rest. Libpostal-rest offers two services

- /expand gives us the standardized address = expandedAddress
- /parser gives us the different components for an address

For example, for the parser query:

Libpostal rest parser request


```
{
   "query" : "placa carles ibanez 3"
}
```


we get the response:

Libpostal rest */parser* response


```
[
      {
            "label" : "road",
            "value" : "placa carles ibanez"
```

```
    },
    {
        "label": "house_number",
        "value": "3"
    }
]
```

The request is the same for the expand endpoint but the response is different:

Libpostal rest */expand* response

```
[
    "placa carles ibanez 3"
]
```

Geocoder

Geocoder give us the same information than libpostal-rest but within a single call like:
`https://maps.googleapis.com/maps/api/geocode/json?address=placacarlesibanez3&key=YOUR_API_KEY`[1]
We would get a response like:

geocoder response

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "3",
               "short_name" : "3",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Plaça de Carlos Ibáñez",
               "short_name" : "Plaça de Carlos Ibáñez",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Barcelona",
               "short_name" : "Barcelona",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Barcelona",
               "short_name" : "Barcelona",
               "types" : [ "administrative_area_level_2", "political" ]
            },
            {
               "long_name" : "Catalunya",
               "short_name" : "CT",
```

[1] a key must be provided

```
                "types" : [ "administrative_area_level_1", "political" ]
            },
            {
                "long_name" : "España",
                "short_name" : "ES",
                "types" : [ "country", "political" ]
            },
            {
                "long_name" : "08038",
                "short_name" : "08038",
                "types" : [ "postal_code" ]
            }
        ],
        "formatted_address" : "Plaça de Carlos Ibáñez, 3, 08038
Barcelona, España",
        "geometry" :
            "bounds" :
                "northeast" :
                    "lat" : 41.3705931,
                    "lng" : 2.1716681
                },
                "southwest" :
                    "lat" : 41.370015,
                    "lng" : 2.1711519
                }
            },
            "location" :
                "lat" : 41.3703336,
                "lng" : 2.1713897
            },
            "location_type" : "ROOFTOP",
            "viewport" :
                "northeast" :
                    "lat" : 41.3716530302915,
                    "lng" : 2.172758980291502
                },
                "southwest" :
                    "lat" : 41.36895506970851,
                    "lng" : 2.170061019708498
                }
            }
        },
        "partial_match" : true,
        "place_id" : "ChIJ1cYr0ESipBIRkF6CsSTA5Mk",
        "types" : [ "premise" ]
    }
],
"status" : "OK"
}
```


After getting the results from Google, we have to filter the address type. In addition,

we set one of the optional parameters (*bounds*) indicating a bounding box where the address is located. This is done because the same address could be found in different cities or countries. The bounds are related to the coordinates of the accommodation. For example, if we search for "General Salazar 9" we will get several addresses:

- Salazar Jeneralaren Kalea, 9, 48012 Bilbo, Bizkaia, España
- 9 Salazar Ln, Bernalillo, NM 87004, EE. UU.
- 9 Salazar Ln, Ponderosa, NM 87044, EE. UU.
- Salazar 9, Magisterial, 60460 Tancítaro, Mich., México
- ...

Nevertheless, if we add the parameter &bounds=43.2498558,-2.9379479|43.2618558,-2.9599479 being 43.2498558,-2.9379479 the left bottom coordinates of our bounding box and 43.2618558,-2.9599479 the right upper ones, we will get only 2 addresses.

- Salazar Jeneralaren Kalea, 48012 Bilbo, Bizkaia, España
- Salazar Jeneralaren Kalea, 9, 48012 Bilbo, Bizkaia, España

The difference between both of them is that the first one has the type "route", and the other one has the type "street_address". This behavior is the reason why we prioritize the allowed types, being the allowed types and their order:

- **lodging**
- **premise**
- **street_address**
- **route**

The order goes from more specific to more generic.

3.1.2 Duplicates Information

We have information that allows us to answer the question: What accommodations are duplicate?
In some datasets, the information of the clusters is stored with the entity representation information, as an extra attribute. In https://www.cs.utexas.edu/users/ml/riddle/data.html we can find some datasets that follow that approach. We instead have decided to store the clusters information as a separated file.
The format for that file is:

example clusters json file

```
[
  [
    "1"
  ],
  [
```

```
      "2" ,
      "3"
   ]
]
```

It is a collection of collections of ids. In this case we have two clusters {"1"} and {"2", "3"}. This id is the same attribute id in the entity representation.

3.2 Blocking

We apply the blocking at two levels.

1. Worldwide blocking, we compare accommodations within the same city
2. Internal blocking, we compare accommodations complying a predicate

For the internal blocking, we have decided to select the geographically nearest accommodations. The maximum distance is parameterizable but the reference in use is 600 m. We decided this distance because is the highest found in our training dataset. The distance is calculated based in the coordinates from the basic information shown before.
The following data reflects the distribution for the geographical distance of all the duplicate pairs in our dataset:

distance in meters
Min. : 0.000
1st Qu.: 4.955
Median : 12.321
Mean : 31.076
3rd Qu.: 28.321
Max. :596.376

The minimum found distance is zero; the median is 12 m and the mean 31 m. The inter quartile range

$$(IQR) = 3rdQ - 1stQ = 28.321 - 4.955 = 23.366$$

The formula 3.1 is a common one to identify outliers.

$$q > Q_3 + 1.5 \cdot IQR \tag{3.1}$$

If we follow it, we will arrive to the conclusion that everything > 63.37 m is an outlier. That means that our reference, 600 m, is quite permissive.
In fact in figure 3.1 we have a better view of the density distribution for distance, we can see that most of the values are accumulated in distances smaller than 100. This data is obtained from the records classified as duplicate in our training dataset.
Moreover, we don't do comparisons for entities coming from the same provider. Actually, that case happens with a very small incidence (0.02% in our main dataset). Conversely, there is a near linear correlation between being duplicate and the distance.
Being f_d the geographical distance between two entities and p_p a predicate that says if 2 entities have the same provider. Our blocking predicate is 3.2

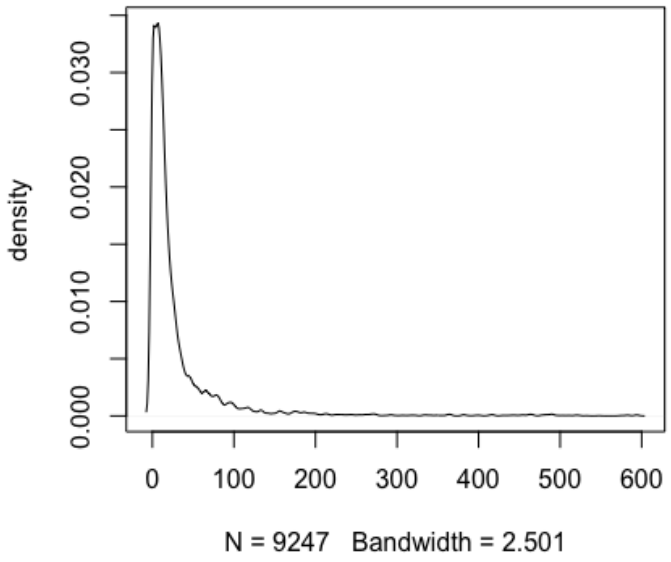


FIGURE 3.1: Density distribution for distance in duplicate entity representations

$$f_b(x, y) < 600 \wedge \neg p_p(x, y) \tag{3.2}$$

In Appendix C there is an analysis of another common scenario that doesn't apply to our case.

3.3 Feature Extraction

Previously we introduced a deriving function f_d that takes two entities and return a record that a classifier can process. As our record variables are the product of a combination between two entities, we will have always a transformation to combine those entities and create new variables / features.
The process of transforming information to a new set of features it is what we call feature extraction.
We are going to use a limited number of feature deriver types:

- EqualsDeriver: takes two arguments and returns true if are equals, and false otherwise.
- NullableEqualsDeriver: a special case of EqualsDeriver. If any of the arguments is null it will return null and it will return the evaluation of a EqualsDeriver otherwise.
- GeoDeriver: will return the distance between two coordinates.
- IntersectImageFeatureDeriver: given two set of images it will return true if there is at least one combination of two images from the different sets whose pHash (perceptual hash, explained in subsection 3.3.1), hamming distance is less than a threshold. Simplifying: indicates if there are images similar enough.
- LevenshteinFeatureDeriver: return the Levhenstein distance between two strings (subsection 3.3.2).
- CosineSimilarityFeatureDeriver: given two strings calculate the tf-idf cosine similarity, explained in subsections 3.3.3 and 3.3.4.
- IntersectionDeriver: Given two arguments if they are equals the value is returned, otherwise null is returned.

All of these types has been implemented in our generic framework.
In order to understand the previous deriver some concept explanations are needed:

3.3.1 PHash (Perceptual Hash)

A perceptual hash is a fingerprint of a multimedia file derived from various features from its content.
For example: we have three images from 3 distinct url's for Hotel Miramar Barcelona.
From a human point of view the figures 3.2, 3.3 and 3.4 are obviously the same. However, what we know from those images from the software point of view is that:

- they have different url's
- they have different sizes
- they have different resolutions

The perceptual hash is returning a hash that can be compared with a string similarity function like hamming distance in order to know the differences. [2]

[2]The hamming distance for two strings measures the number of distinct elements for those strings, in other words, the number of elements that differ from one string to the other one.

FIGURE 3.2: Miramar Booking.com

FIGURE 3.3: Miramar Hoteles.com

FIGURE 3.4: Miramar Expedia

For example, according to http://www.phash.org/demo/ the Expedia and Booking.com images have a distance of 20. A threshold for the distance is defined with the value 26; anything less than 26 means the images perceptually equivalents.
PHash is robust to some transformations like resizing, compression, contrast adjustment and many others. But it is vulnerable to others transformations like cropping. There are other options in order to compare images that are more robust than pHash, but pHash is simple and fast to process.
One alternative which works well for cases like fig 3.5 was openImaj (Hare, Samangooei, and Dupplaw, 2011). However, as said before, the processing time to download and compare the images is too big. Consider an average of 47.22 images per accommodation.

FIGURE 3.5: Failing pHash

3.3.2 Levenshtein Distance

We have already defined some edit distance metrics like GMD, Hamming, etc. What they have in common is that those metrics measure the number of changes done in the candidate object in order to become equals to the original one.
"In information theory, linguistics and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other." (Wikipedia contributors, 2018b)

3.3.3 TF-IDF

TF-IDF (Term Frequency - Inverse Document Frequency) is a measure of how many repetitions have a term within a document (variable) compared to the number of times that term appears in other documents.
For Example. We have the property name and we have three entities whose variables name are:

1. Hotel Miramar Barcelona GL
2. Hotel Miramar Barcelona
3. Miramar Barcelona

The term frequency for the word "Hotel" in "Hotel Miramar Barcelona GL" is 1. For "Hotel Miramar Barcelona" is also 1 and For "Miramar Barcelona" is 0.
The word "Hotel" appears in two "documents", "Hotel Miramar Barcelona GL" and "Hotel Miramar Barcelona". So its idf will be 1/2, remember that the "i" of *idf* stands for inverse. If we multiply the tf * idf we get our tf-idf value for a specific word in the context of a single variable, because tf applies to the variable.

The formulas used are:

$$tf = f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$$

$$idf = \log \frac{N}{n_t} = -\log \frac{n_t}{N}$$

where $n_t = |\{d \in D : t \in d\}|$, N is the number of documents and $f_{t,d}$ is 1 if the term t occurs in d and 0 otherwise.

3.3.4 Cosine similarity

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}} \tag{3.3}$$

Equation 3.3 shows the formal specification for cosine function. x and y are two vectors. In our case, they are two vectors of tf-idf values for the set of words in both variables.
For example let's suppose we have the following Table 3.1.

TABLE 3.1: Tf-idf

word	idf	variable 1 tf	variable 2 tf
hotel	0.67	1	1
miramar	0.03	1	1
barcelona	0.2	1	1
gl	0.01	1	0

With the given table 3.1. Our resulting cosine similarity for the comparison of variable 1 and variable 2 would be:

$$(0.67^2 + 0.03^2 + 0.2^2) / ((\sqrt{0.67^2 + 0.03^2 + 0.2^2 + 0.01^2}) \cdot (\sqrt{0.67^2 + 0.03^2 + 0.2^2}))$$

Note that for the numerator there is no references to 0.01 in "gl" row because the tf of variable 2 is 0.

3.3.5 Features

We have already introduced the concept of feature deriver and we mentioned seven types that we use.
With the available data and those derivers, we get the final features:

- **distance:** geo distance
- **nameLevenshteinDistance:** levenshtein distance of the name
- **addressLevenshteinDistance:** levenshtein distance of the address
- **expandedAddressLevenshteinDistance:** levenshtein distance of the expanded address
- **addressNameLevenshteinDistance:** levenhstein distance of the name component of the address

- **sameAddressNumber:** whether the address number components are equals
- **sameLevel:** whether the levels address component are equals
- **sameUnit:** whether the unit address component are equals
- **sameChain:** whether the chain is the same
- **sameType:** whether the type are equals
- **typeIntersection:** in case of being equals the types, what is it?
- **sameStars:** whether the starts are equals
- **providers:** ordered concatenation of the provider
- **nameCosineSimilarity:** the tf-idf cosine similarity for the name
- **addressCosineSimilarity:** the tf-idf cosine similarity for the address
- **expandedAddressCosineSimilarity:** the tf-idf cosines similarity of the expanded address
- **sameImages:** whether any image is contained in both entities (uses phash)
- **isSameGoogleAddress:** whether is same google formatted address
- **googleLevenshteinDistance:** the levenshtein distance of the google formatted address
- **sameGoogleAddressNumber:** whether the address number component for google is the same

3.3.6 Dataset Format

With the previous information, our final dataset format [3] would be:

```
@attribute id1 {A1, B1, C1}
@attribute id2 {A2, B2, C2}
@attribute distance numeric
@attribute nameLevenshteinDistance numeric
@attribute addressLevenshteinDistance numeric
@attribute expandedAddressLevenshteinDistance numeric
@attribute addressNameLevenshteinDistance numeric
@attribute sameAddressNumber {false,true,null}
@attribute sameLevel {null,false,true}
@attribute sameUnit {null,false,true}
@attribute sameChain {null,false,true}
@attribute sameType {false,true}
@attribute typeIntersection {null,Hotel,Apartment,Hostel,'Guest House','
    Bed and Breakfast',Resort,Residence,Motel}
@attribute sameStars {false,true}
@attribute providers {A, B, C}
@attribute nameCosineSimilarity numeric
@attribute addressCosineSimilarity numeric
@attribute expandedAddressCosineSimilarity numeric
@attribute sameImages {false,true}
@attribute isSameGoogleAddress {false,true,null}
@attribute googleLevenshteinDistance numeric
```

[3] in .arff, weka format

```
22 @attribute sameGoogleAddressNumber {false,null,true}
23 @attribute class {true,false}
```

LISTING 3.3: WEKA Arff Dataset Format

The attributes id1 and id2 are filtered out when we are training the model but we need them when evaluating a dataset in order to know to what elements the pair comparison refers.

3.4 Pair Classification

There is a wide variety of classifiers available, some of them require an exhaustive tuning and expertise to get the right parameters optimized. We have tested some common classifiers trying to consider the different bias introduced by them depending on their types. In other words, different types of classifiers adapts better to some problems, we have choose from: linear classifiers, bayes, bagging, boosting, neural network. Possibly, we haven't been able to get the best optimization for the most complex.
We know from other previous works which classifiers were chosen and what where the results for the scope of our problem. Concretely (Kozhevnikov and Gorovoy, 2016) and (Zheng et al., 2010) both use meta classifiers based on decision trees: Random Forest (Breiman, 2001) and Bagging of decision trees (Breiman, 1996) (both quite similar).
The goal in this phase is to rank the following chosen classifiers:

- **BayesNet**
- **MultilayerPerceptron**
- **IBk:** KNN (k-Nearest Neighbours) implementation in Weka.(Aha, Kibler, and Albert, 1991)
- **SGD:** Stochastic Gradient Descent (Bottou, 2010)
- **Logistic:** Logistic Regression in Weka. (Le Cessie and Van Houwelingen, 1992)
- **Random Forest**
- **Adaboost** (Freund and Schapire, 1996)
- **xgb** (Friedman, 2001)

3.5 Clustering

The first step to group the accommodations based on the pair resolution is to create the distinct graphs resulting for that resolution. Any vertex is the entity representation, and we will have an edge for any pair classified as duplicate by the binary classifier.

3.5.1 Predicate + Cut

To do the clustering we need a predicate that tells us if the graph (g) is stable (S_g), and a function f_s that split the graph according to some criteria.
Our target function is $f : G \mapsto \{G\}$, that function receives a graph and return a set of graphs, it is applied recursively.
The chosen split function f is a min cut algorithm (Stoer and Wagner, 1997), and to check the maximal cliques Bron–Kerbosch algorithm was used (Bron and Kerbosch, 1973).

$$f(g) = \begin{cases} \{g\} & \text{if } S_g \\ \cup\{f(x) : x \in f_s(g)\} & \text{otherwise} \end{cases} \tag{3.4}$$

Equation 3.4 shows the function used to calculate the partitions of a graph.
We tested some predicates that will be explained in detail later:

- Basic: (S $\iff$ true)
- Providers: (S $\iff$ no providers repetitions within graph)
- Clique: (S $\iff$ graph conform a maximal clique)

Basic Predicate

The predicate S $\iff$ true is our base predicate.
In an indirect way, it states that the graphs are always stable, we won't split any graph, all the elements within a graph are grouped to form the final cluster.
It assume transitiveness, let A, B and C be accommodations. If A == B, and B == C, then A == C. We will group A, B, and C as being the same entity.

Providers

Our providers predicates is S $\iff$ no providers repetitions within graph.
Indirectly it means that any graph containing 2 or more accommodations from the same provider should be split.

Clique

(S $\iff$ graph conform a maximal clique).
A graph is a Clique if all the vertices are connected with any other vertex within the graph.

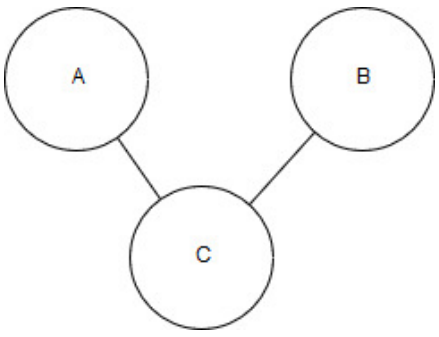


FIGURE 3.6: Non clique graph

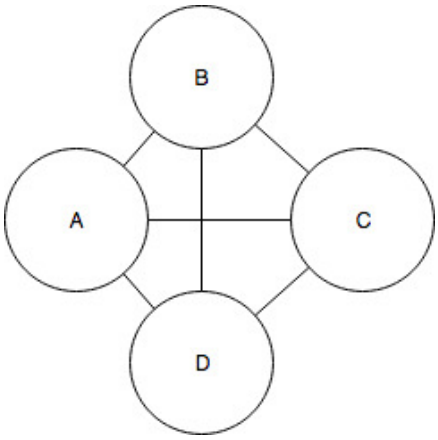


FIGURE 3.7: Clique graph

Figure 3.6 is not a clique; A and B are not connected.
Figure 3.7 is a clique, any pair of two elements is connected.

Min-cut

In graph theory, a minimum cut of a graph is a cut (a partition of the vertices of a graph into two disjoint subsets that are joined by at least one edge) that is **minimal in some sense**.
Our graphs are undirected, the edges has no orientation, the edge between A and B is the same as from B to A. Our edges also have weight and the weight of our edge is the probability prediction for being duplicates that our classifier provides.
Minimal in our undirected weighted graph refers to cutting the vertex with minimal weight or minimize the weight of the edges being cut. The reasoning behind is that we don't want to cut edges with high probability because high prediction is correlated to being duplicate.

Examples

In Table 3.2 we can see the table representation for a graph. We have three vertex (A1, B1 and C1) from three different provider's, A, B and C. The greater the prediction value is, the greater is our confidence on the two entity being duplicates.

	from	to	prediction
1	A1	B1	1
2	B1	C1	0.8

TABLE 3.2: Simple graph table representation

The resulting graph is figure 3.8. [4]
Our basic predicate state that this graph is enough, so we can group the results and get a final cluster as {A1, B1, C1}. This result would be shared with the providers predicates. There are only 3 providers in this graph (A, B and C) and none of them is repeated.
Based in our previous definition for a Clique, our second predicate would say that it is not a clique. As the graph is not stable[5] we need to cut it. If we apply the chosen

[4] the package igraph in R has been used to generate the graphs used in this example
[5] given our "stable" definition in 3.5.1

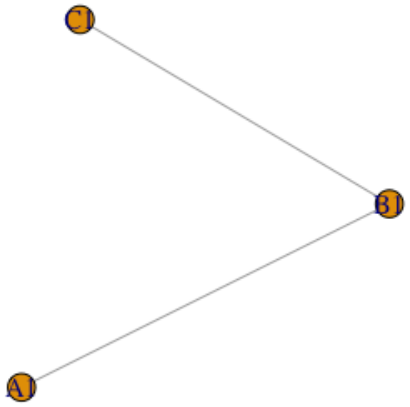


FIGURE 3.8: Simple graph

min cut implementation we will get two partitions, one composed by (A1, B1) and the other one C1. In the end we will have two clusters {A1, B1} and {C1}.
Our next example is a little more complex. The table to generate the graph would be Table 3.3.

	from	to	prediction
1	A1	B1	0.8
2	A1	D1	1
3	C1	D1	1
4	A1	B2	1
5	A1	C2	1
6	B1	D1	1
7	B2	C2	1

TABLE 3.3: Complex graph table representation

The resulting graph the figure 3.9.
Our first basic predicate,3.5.1, would cluster A1, B1, B2, C1, C2, D1
Our second and third predicate would apply the cut function and we would get two partitions: one containing only C1 and the other one containing the rest of the nodes. That partition is a single element and it would be grouped as a separated cluster {C1}. Our min cut algorithm decides the cut, here C1 is cut because it has a unique edge to the rest of elements (D1).
The second partition is the graph in the figure 3.10. That partition wouldn't be stable in neither our clique strategy (because is not a clique), nor our providers strategy (has repeated providers), a new cut would be done, the edges cut are A1–B1 B1–D1. The result for that cut would be two partitions, (A1, D1, C2, B2) and (B1). As in the previous step B1 is isolated, a cluster is created {B1}.

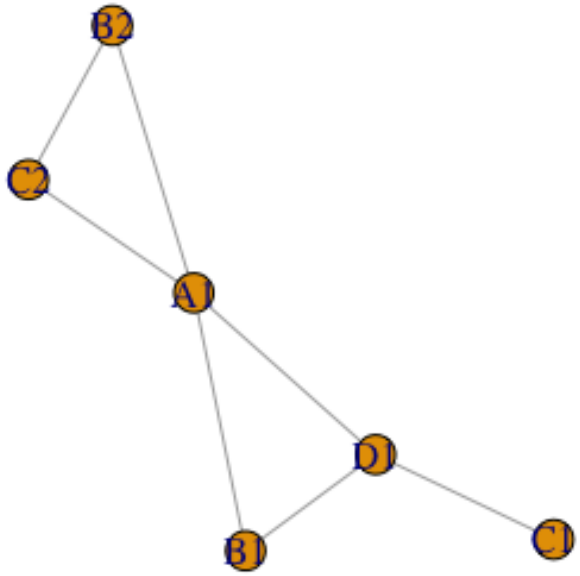


FIGURE 3.9: Complex graph

For the rest (A1, D1, C2, B2) according to our providers predicate is stable, so a cluster is also created: {A1, B2, C2, D1}.
For our clique strategy the resulting graph in figure 3.11 is no yet a clique. If we cut the D1 element will be isolated and the remaining is a clique, we get two clusters {A1,B2,C2} and {D1}
The final results for the strategies are:
Basic: {A1, B1, B2, C1, C2, D1}
Providers: {C1}, {B1}, {A1, B2, C2, D1}
Clique: {C1}, {B1}, {A1,B2,C2}, {D1}

3.5.2 Interpretation

We have mentioned the concept of clique. In a deduplication our concept of cluster have some special implications different to the meaning of cluster in other scopes. For example, when we say that A1, B1, C1 conform a cluster what we are saying translated to a graph is that they would conform a clique, where any edge is a relation of duplicity, A1 is duplicate of B1, A1 is duplicate of C1 and B1 is duplicate of C1. It is important not to misunderstand this conclusion, the real clusters being cliques, with the Clique strategy that tries to approximate those clusters/cliques.

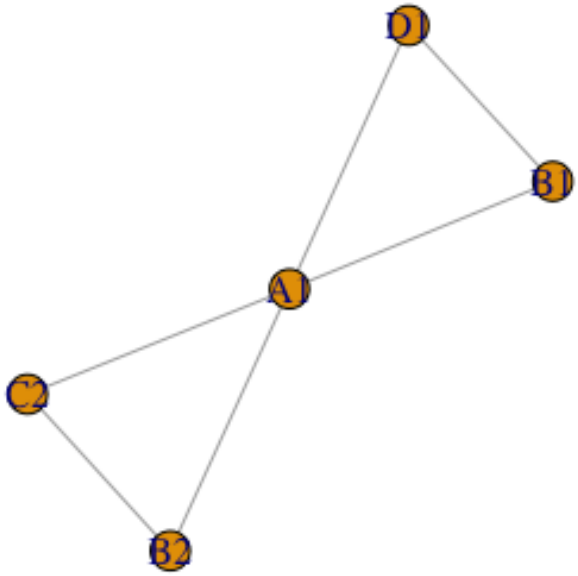


FIGURE 3.10: Complex graph partition after first cut

If we use our basic strategy to cluster the figure 3.6 what we are doing according to this interpretation is creating a relation or an edge between A and B.
Generalizing this concept we see that the clique strategy never creates edges, only cuts, the basic strategy never cuts, only creates edges and the providers strategy cuts in some cases (when the graph has a provider repeated) and creates in other cases.
Forgetting about the clusters and focusing in the graph, *our problem, given a graph with relations of duplicity between vertices, is the problem of approximating the real cliques*.

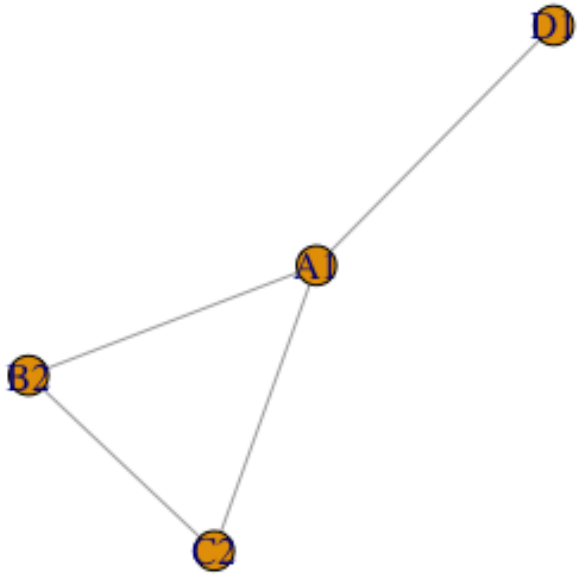


FIGURE 3.11: Complex graph paritition after second cut

3.6 Evaluation Metrics

We will have different solutions to a problem and we will need some way of comparing those solutions. The deduplication process includes several phases and all of them could potentially have their own metrics[6]. However, we will focus on the metrics for the classification part and the clustering part. We use different metrics for the pair evaluation and the evaluation at cluster level.

3.6.1 Pair Classification

For the evaluation of the binary classification model, we use ***precision and recall***. The technical definition would be (Ting, 2010):

- **Precision** = Total number of documents retrieved that are relevant/Total number of documents that are retrieved.
- **Recall** = Total number of documents retrieved that are relevant/Total number of relevant documents in the database.

[6]In fact, from the mathematical point of view, maybe we should be call it "measure" (Menestrina, Whang, and Garcia-Molina, 2010)

In our scope retrieved means comparisons that we have classified as duplicates. Moreover, relevant documents in the database means comparisons that we know that in fact are duplicates.
Therefore, the precision would be the number of real duplicates detected by the classifier vs the total number of comparisons classified as duplicates, including the ones that actually were not real duplicates (false positives).
The recall would be the number of real duplicates detected by the classifier vs the total number of real duplicates.

id1	id2	distance	real class	predicted class
1	2	3	true	true
1	3	9	true	true
1	4	24	false	false
2	3	6	true	true
2	4	21	false	false
3	4	15	false	true

TABLE 3.4: classification example

In table 3.4 we can see an example of the prediction of our chosen classifier compared with the known classification.
The precision is:

$$\frac{\#(predicted == true)\&(real == true)}{\#(predicted == true)}$$

In this case our precision is 0.75, 3 out of 4 predicted duplicates were in fact duplicates.
The recall is:

$$\frac{\#(predicted == true)\&(real == true)}{\#(real == true)}$$

Following the described formula, the recall for our example is 1, all the real duplicates where predicted as duplicates.
Our target metric for the pair classification is the precision; if there are several solutions with the same precision we choose the one with the best recall.

3.6.2 Clusters

In subsection 2.1.3 we explained that we can generate different cluster configurations for the same pair classification resolution and we finished saying that:
"The clustering phase deals with those results and the grouping in order to get the *closest* configuration of the real clusters."
Here we try to measure how close two cluster configurations are.
The metrics at a cluster level are quite more complex and less intuitive than the ones for the pair classification. We will try to explain the problems and the chosen metrics in the following lines.
Let's suppose we have the following predicted cluster configurations:

$$P_1 = \{\langle 1,2,3,4\rangle\}$$

$$P_2 = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle\}$$

$$P_3 = \{\langle 1\rangle, \langle 2,3,4\rangle\}$$

$$P_4 = \{\langle 1,2\rangle, \langle 3,4\rangle\}$$

We know that the "real" cluster configuration is:

$$R = \{\langle 1,2,3\rangle, \langle 4\rangle\}$$

In P_1 we have predicted that elements 1, 2, 3, 4 conforms a unique cluster.
In P_2 we have predicted that we have four clusters, one for every element.
In P_3 we have predicted that we have two clusters, one for 1, and another one for the rest.
In P_4 we have predicted that we have two clusters, one for 1 and 2, and another one for 3 and 4.
The reality (R) is that 1,2 and 3 conforms a cluster, and 4 conforms another cluster.
We propose two measurements to calculate how close are P_1 and P_2 to R:

- Generalized Merge Distance (GMD)(Menestrina, Whang, and Garcia-Molina, 2010).
- Variation of Information (VI)(Meilă, 2007).

GMD

In subsection 2.1.2 we said that a distance measurement indicates the dissimilarity. The GMD evaluates the number of splits and merges needed to modify the prediction in order to get the real solution.
In our P_1 example it would be enough splitting $\langle$1,2,3,4$\rangle$ in two clusters $\langle$1,2,3$\rangle$ and $\langle 4\rangle$. No merge operations are needed.
In our P_2 example we need to merge $\langle 1\rangle$, $\langle 2\rangle$ and $\langle 3\rangle$ in $\langle$1,2,3$\rangle$. No merge operations are needed.
In our P_3 example we need to split $\langle$2,3,4$\rangle$ in $\langle$2,3$\rangle$ and $\langle 4\rangle$ and later merge $\langle 1\rangle$ and $\langle$2,3$\rangle$.
In our P_4 example we need to split $\langle$1,2$\rangle$ in $\langle 1\rangle$ and $\langle 2\rangle$ and later merge $\langle 2\rangle$ and $\langle$3,4$\rangle$.

Prediction	Splits	Merges	Total Distance
P_1	1	0	1
P_2	0	2	2
P_3	1	1	2
P_4	1	1	2

TABLE 3.5: GMD cost for prediction examples

The Table 3.5 shows the summary of the operations needed and total distance, being the total the splits + the merges. However GMD allows us to define cost functions for splits (f_x) and merges (f_m). [7]
We have decided that our target is to have the minimum number of splits. The reason is that we prioritize the precision rather than the recall. In the Appendix D there are more details about how are related the GMD splits/merges from the cluster evaluation and the precision/recall from the pair classification. Therefore, in this scenario we have decided that we will only consider the number splits, the merge function will be always 0.

[7] In our case we have modified a little bit the definition for the split and merge and the GMD calculation, details are provided in appendix A.

$$f_s(x) = |x| - 1$$
$$f_m(x) = 0$$

VI

The definition for VI is:
"The criterion, called variation of information (VI), measures the amount of information lost and gained in changing from clustering C to clustering C'."(Meilă, 2007).
From the previous GMD calculations, we can see that the number of merges and splits can be the same but the "size" of the clusters could be different.
For example the P_3 cost is 1 split and 1 merge. The P_4 cost is also 1 split and 1 merge.
If we calculate the VI for our example predictions we will obtain Table 3.6.

Prediction	VI
P_1	0.8112781
P_2	1.188722
P_3	1.377444
P_4	1.188722

TABLE 3.6: VI cost for prediction examples

We can see in Table 3.6 that the VI for P_4 is smaller (better) than the VI for P_3, even if they have the same split and merge cost.
In the GMD subsection we said that GMD allows to configure functions for the splits and merges costs. We can calculate the VI value modifying those functions:

$$f_s(x) = f_m(x) = -\sum_n \frac{|x_n|}{N} \cdot log_2(\frac{|x_n|}{t})$$

Where N is the total number of elements = the sum of elements in all clusters = all the entities, and the variable t is the total number of elements in x.
In case of having the same number of splits we will consider the best VI.

Combination of GMD and VI

We have defined two measurements for our cluster evaluation. But we don't use it as a unique metric, our definition, mixing the GMD and the VI part, can be synthesized as: "We want to minimize the number of splits and in case of being the same we will choose the smaller VI".
What we are defining is in fact a comparator. Being x and y two different solutions, vi() the VI calculation and splits() the splits number:

$$x > y \rightarrow splits(x) < splits(y) \vee (splits(x) = splits(y) \wedge vi(x) < vi(y))$$

If we rank the prediction examples according to this comparator, the order is:

1. P_2
2. P_1
3. P_4

4. P_3

Chapter 4

Experiments

In this chapter, we first give a description of our data and we mention the technologies used. Once these introductory topics are covered, we focus on the pair classification and clustering phase. In those sections, we explain how we conduct the experiments and get the measurements defined for each phase in the previous chapter. We also show the results for each phase. It is important to note that the clustering phase depends on the results from the pair classification.

4.1 Accommodation Data Description

We have used data consisting of a manual deduplication of a subset of accommodations located in:

- Madrid
- Barcelona
- Amsterdam
- Other cities

That data consist in:

- Information of the entity representations.
- Information about the clustering for those representations.

The format of the information was already explained in Section 3.1.

City	1	2	3	4
Barcelona	723	293	263	8
Madrid	303	140	184	7
Amsterdam	802	54	3	0
Other Cities	0	0	4	1270
Merged	1821	475	440	1277

TABLE 4.1: Distribution by city and cluster size

Table 4.1 shows four datasets with the number of gold clusters per size. The Barcelona, Madrid and Amsterdam data are disjoint between them and were generated manually checking those cities, but the "other cities" can contain elements from the other datasets, the "other cities" dataset was provided by external people.

The merged line is the result of joining all 4 datasets, we can see that when merged the results are not a mere sum of the different datasets due to the fact of not being disjoint sets as stated before. Our experiments are based in the merged one.
Summarizing:

1. We have $1821 + 475 + 440 + 1277 = 4014$ deduplicated clusters
2. We have $1821 + 475 \cdot 2 + 440 \cdot 3 + 1277 \cdot 4 = 9199$ accommodations

When we apply our blocking predicate to generate all the records for our dataset in the format specified in the Subsection 3.3.6 we get a dataset with **160907** records.

4.1.1 Duplicates Comparisons/Records

Based on the distribution of clusters size in Table 4.1 we can know how many dataset records we will generate if our blocking predicate includes all the real duplicates comparisons.[1]
Being d_1, d_2, d_3 and d_4 the number of elements for the clusters of size 1, 2, 3 and 4.
d_1 = 1821 elements are isolated; no comparisons are needed for them.
d_2 = 475 clusters of 2 elements
d_3 = 440 clusters of 3 elements
d_2 = 1277 clusters of 4 elements

$$\sum_{n=2}^{4} d_n \cdot \binom{n}{2} \tag{4.1}$$

Applying equation 4.1 we get the theoretical total of duplicates comparisons.

$$475 \cdot \binom{2}{2} + 440 \cdot \binom{3}{2} + 1277 \cdot \binom{4}{2} = 9457$$

From the 160907 records in our dataset we have **9248** labeled as duplicate. That means that we are losing $9457 - 9248 = 209$ records in the blocking process.

4.2 Tools & technologies

The basic framework for deduplication has been built in Java, the main reason to choose java is that it is widely adopted in the industry. Weka has been used to build the models and for the graph visualization we have used Neo4j. R has also been used for some statistical analysis and visualization. Specifically the library igraph from R has some useful functions to works with graphs and visualization.

4.3 Pair Classification

In this section we evaluate the different classifiers considered in the Section 3.4.

[1] Usually losing a few records labeled as duplicate doesn't have a big impact in the model created, it doesn't affect the training. However our blocking predicate should indeed include near the total of the real duplicates.

4.3.1 Evaluation

The evaluation in the pair classification is made based on cross validation 90/10. A cross validation does a partitioning of the dataset, in the pair evaluation it is applied to the dataset after blocking; it is applied to the 160907 records.

4.3.2 Hyperparameters optimization

As mentioned in the Section 3.4 some of the classifiers to test require tuning. We did grid search over the most relevant parameters. That means that we choose a range of values for the different parameters and do cross validations with the different combinations of parameters.
Sometimes manual tests were done before to determine the range of values to test.

4.3.3 Pair Classification Results

Position	Classifier	Precision	Recall	Build time
1	Adaboost	0.993	0.994	5215 s
2	xgb	0.992	0.994	455 s
3	Random Forest	0.991	0.99	251 s
4	BayesNet	0.984	0.981	35 s
5	Logistic	0.982	0.979	23 s
6	SGD	0.981	0.981	3 s
7	Multilayer Perceptron	0.981	0.981	127 s
8	IBK	0.979	0.960	0 s

TABLE 4.2: Classifiers ranking by precision and recall

Table 4.2 shows the ranking by precision in pair classification for the different classifiers tested. The recall and build time[2] are also considered. In the following lines, the final configuration is described for all of them.

BayesNet

The default *Searcher* in the Weka configuration for BayesNet uses K2(Cooper and Herskovits, 1991). That configuration gives us a precision of 0.908. Using SimulatedAnheling(Bouckaert, 1995) with 300 runs we get the 0.984 as we can see in the Table 4.2. We didn't use the default runs number, 1000, because of memory consumption.

IBK

IBK spent too much time in order to evaluate the records in our dataset. The result shown in the Table 4.2 is obtained with a mini dataset; the size of that mini dataset is 10% of our dataset.

xgb

xgb (Chen, He, and Benesty, 2015) is an implementation of gradient boosting. The main problem with xgb is that the implementation is found in different languages

[2]build time is the time taken to build the model.

and format but not in Weka, which is our main framework. That doesn't mean that we cannot use it, the model and prediction must be generated outside the framework. The way we do this is:

- Generate the dataset in Java using the framework
- Import the dataset and do the pair evaluation in R
- Export the predictions to plain text in R
- Import the predictions in Java using the framework and do the cluster evaluation using our framework

The performance of xgb is quite good. It has a top performance.
The default grid search from R resulted in the following configuration:
In the parameters tuning 'eta' was held constant and the rest of the parameters were optmized (alpha, lambda, nrounds). Accuracy was used to select the optimal model using the largest value. The final values used for the model were nrounds = 150, lambda = 1e-04, alpha = 0 and eta = 0.3.[3]

Random Forest

The final parameters for the random forest are 98 iterations / trees with seven features and 0.5 as prediction threshold obtained from the grid search.

Adaboost

The classifier chosen for AdaboostM1 is J48 (Weka C48 implementation) (Quinlan, 1993), 300 iterations.

Logistic

Used the default Weka configuration.

Multilayer Perceptron

Default Weka configuration except for the *trainingTime* parameter that it is override to 40.

SGD

Default Weka configuration.

4.4 Clustering

The purpose of the pair classification ranking was to decide what classifiers to use in our problem, but further tuning was done in order to get the best classifier taking into account also the cluster evaluation. From our analysis, and according to Table 4.2 ,it seems that the tree ensemble models (Adaboost, XGB, Random Forest) achieve the best results.

[3]This is actually the summary output from R evaluation on xgb.

4.4.1 Evaluation

In the clustering, the cross validation is applied before blocking, to the clusters set. In other words, we partition our 14014 clusters set. The blocking is applied to every partition of clusters resulting from the cross validation partitioning.
The reason of making partition this way is that otherwise some comparisons needed to recreate a cluster could be in another test partition. We need all original cluster comparisons to be in the same test partition in order to get accurate results[4]. The 140002 records is the result of blocking and combining 9199 accommodations. However, a cluster of four elements can generate up to 6 records. When doing the cross validation nothing enforce those 6 records to be in the same partition. With the second option, we are enforcing it because we first partition the clusters set and after that, we apply the blocking and combine the accommodations for that partition.
Another side effect of this decision is that we will have fewer combinations. For instance:
The combinations resulting from mixing 100 accommodations between each other and later partitioning in 10 partition will be greater or equal to the number resulting from partitioning the 100 accommodations in 10 partitions and then combining the accommodations within those 10 partitions.
However, we have to take into account that, as a drawback, we are missing comparisons. We won't lose comparisons that would result in a duplicate, but some of them could have an impact in the classifier.
For example, let's suppose that we have originally 4 clusters {A1, B1}, {A2, B2}, {A3, B3}, A4, B4, being A and B two different providers. Let's also suppose that cluster 1 is close to cluster 2, meaning that our blocking predicate will compare their elements and cluster 3 and 4 are also close. To simplify, our blocking predicate doesn't compare two elements in the same cluster, neither the same provider.
The total of comparisons by the blocking predicate in our first evaluation mode would be

- A1-B1
- A1-B2
- A2-B1
- A2-B2
- A3-B3
- A3-B4
- A4-B3
- A4-B4

No matter how we group these comparisons we will have all of them in our first evaluation mode. However in our second evaluation mode if we group the first cluster with the third cluster, and the second with the fourth. We will only have the comparisons:

- A1-B1

[4]The effect of missing comparisons is treated in Appendix D.

- A2-B2
- A3-B3
- A4-B4

Technically we could do some kind of stratified cross validation trying to group together for each cross validation fold clusters close to each other. Again, we would come back to the problem of clustering our clusters and getting a similar size for them.

4.4.2 Hyperparameters optimization

In the clustering, we do a grid search for these three classifiers optimizing our cluster evaluation measurement. We also take into consideration the prediction probability.

Thresholds

A classifier can give a value indicating the class predicted: "true" for duplicate and "false" for not duplicate but some of them also compute a probability for both classes, we will call this the *probability distribution*. For example the probability for true could be 0.8, and for false 0.2. The default value to classify something as true or false will be 0.5, note that the sume of probability for true and false must be 1. That means that a probability of 0.5 or more for true will result in the instance being classified as duplicate (true). That value (0.5) is our *prediction threshold*.
Depending on our strategy it could be convenient to modify the prediction threshold for our model (when probability distribution available). If the strategy tries to maximize precision, we could allow edges with less weight to give some margin to the strategy to compensate, or the opposite.

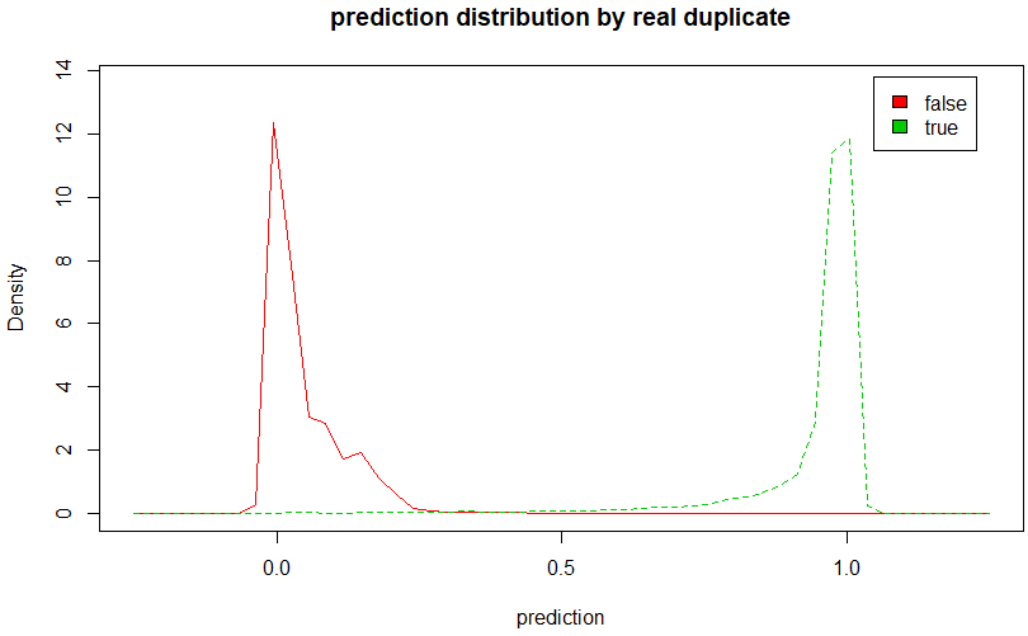


FIGURE 4.1: Prediction distribution by real duplicate

In the Figure 4.1 is an example of the density for the estimated duplicate == true probability for records that we know that they are duplicates (in green), and records that we know that in fact they are not duplicates (in red). We can see that most of the duplicates have a probability near one, and most of the non-duplicates have a probability near zero [5]. We can move the prediction threshold in order to classify more records as duplicate or not duplicate. The final impact as commented will depend on the clustering strategy in use. We will see later that performance in some strategies is optimal at a low threshold.

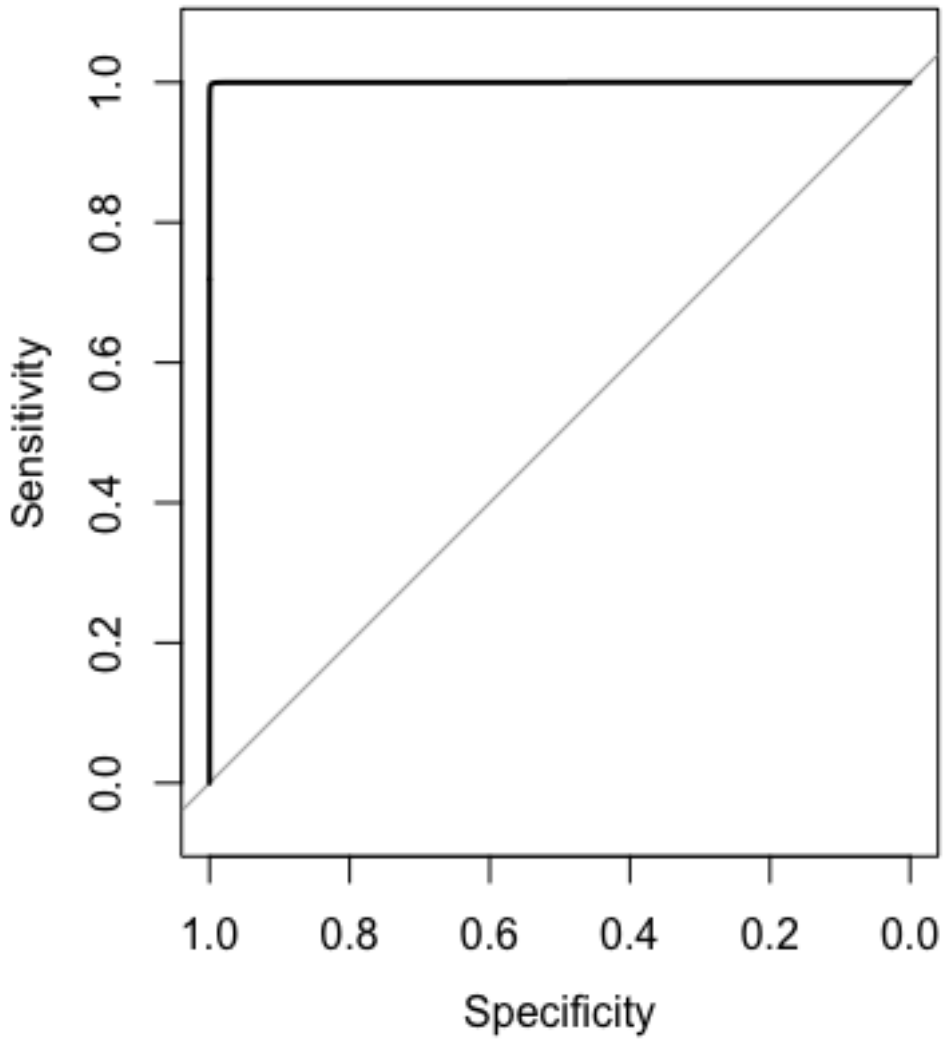


FIGURE 4.2: ROC

Another more common representation for the previous graph is a ROC curve like in Figure 4.2. *Area under the curve: 0.9998*.

[5] It is a density graph so the peak of densities are found in zero an one.

4.4.3 Clustering Results

One interesting note is that the ranking for the best classifiers in cluster evaluation mode and pair wise comparisons evaluation is different. Remember that for the pair classification, we are using as metrics precision and recall and for the clusters evaluation our measures are splits and VI. We will see in the following cluster evaluation results that the best performance is achieve by Xgboost, while Adaboost is second.

Adaboost

We tested values between 8 and 64 iterations with different thresholds obtaining best results with 32.
The best result obtained considering splits and VI is using providers strategy as we can see in Table 4.3.

strategy	threshold	splits	merges	VI
cliques	0.343	3	177	0.0453
providers	0.356	3	103	**0.0269**
basic	0.356	15	96	0.0295

TABLE 4.3: Adaboost splits / VI results

Random forest

The results from random forest are good also. Again, the best combination is the providers' strategy as shown in the Table 4.4. We tested values for iterations (number of trees) from 95 to 100. For features, we tested from five to seven and thresholds between 0.33 and 0.53. All the results in the Table 4.4 shares the same configuration, 98 iterations, 6 features and a threshold of 0.526.
We can see the evolution of VI for the different strategies by threshold, and without considering the splits, in figure 4.3.
As we can see the clique strategy has an optimum in a very low threshold. The basic strategy optimum is near to 0.5. The VI for the providers strategy is near constant in the chosen range of threshold thought it has a increasing trend.
Figure 4.3 is an example for random forest, but the graph is similar for others models.

There are several implications in the selection of our strategy.

strategy	splits	merges	VI
cliques	6	207	0.0541
providers	7	92	**0.0307**
basic (plain clustering)	32	106	0.0374

TABLE 4.4: Random forest splits / VI results

XGboost

The best results obtained by strategy are shown in Table 4.5. The best configuration is the same as for the pair evaluation (nrounds = 150, lambda = 1e-04, alpha = 0 and eta = 0.3).

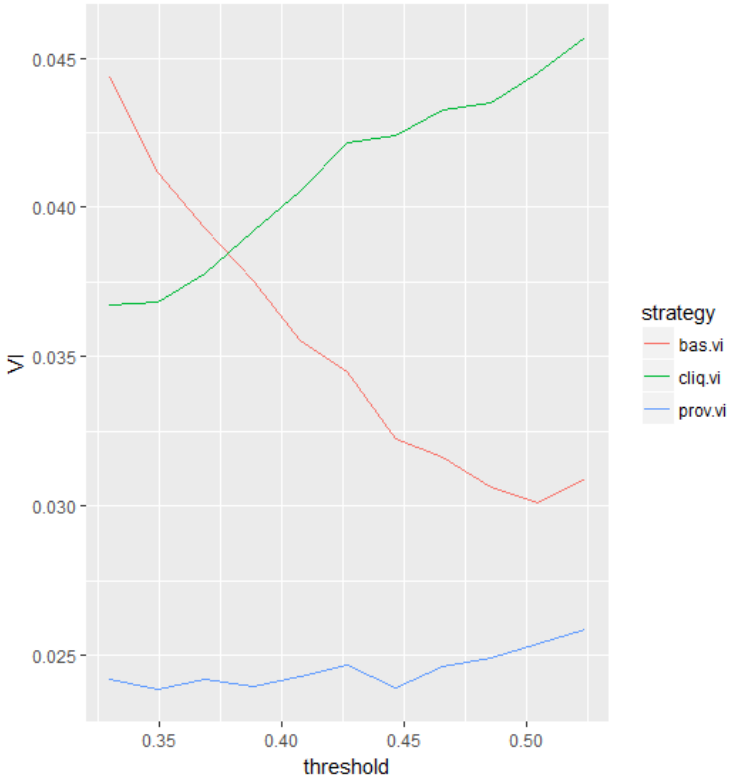


FIGURE 4.3: Random forest VI by threshold

Xgboost allows a wide range in the threshold, the reason is that the dispersion of the prediction for duplicates and no duplicates is high; most of the classified as no duplicate have a prediction very close to 0, and those classified as duplicate are very close to 1.

strategy	threshold	splits	merges	VI
cliques	0.22	1	135	0.0336
providers	0.42	1	85	**0.0210**
basic	0.22	5	85	0.0231

TABLE 4.5: Xgboost splits / VI results

4.5 Results by dataset

In section 4.1 we showed our final dataset composition, that is a mix of several datasets. Some experiments indicate that different models by city would predict better the clusters for that city instead of using a unique model trained on the combination of data from all cities. For example, a test with Adaboost with all 3 cities excluding the worldwide mix gives us a result of 4 splits and a VI of 0.0159. The Table 4.6 shows the best result by city using Adaboost, the results are consistent with section 4.3 being always the providers strategy the best one.
However, we know also that applying a model from one city in samples of another city will give us a worst result. As we don't have samples from all cities worldwide it is better to use the data combined in all cities in order to reduce the bias. It is also

city	splits	VI
Madrid	0	0.0208
Barcelona	0	0.0161
Amsterdam	2	0.0421

TABLE 4.6: Best splits/VI values in different datasets

possible a mix of both solutions, in case we have a model for one city use it, if not, use a generic one.

Chapter 5

Analysis

In the Section 3.5 we introduced several predicates that would define our clustering strategy. This chapter it is about the results from the previous chapter, we analyze the attribute selection and the behavior of the different strategies defined.

5.1 Attributes

For the attribute selection, we have used the importance computation given by the random forest and xgboost. The conclusion is that all attributes count.

5.1.1 Random Forest Attribute Importance

Weka defines the attribute importance as "Attribute importance based on average impurity decrease (and number of nodes using that attribute)".
We can see the results from that computation in the Table 5.1. [1]

0.46	(4351)	distance
0.44	(2033)	addressLevenshteinDistance
0.44	(362)	sameType
0.43	(1448)	expandedAddressLevenshteinDistance
0.43	(3023)	nameLevenshteinDistance
0.37	(1177)	addressNameLevenshteinDistance
0.37	(626)	sameChain
0.33	(749)	typeIntersection
0.31	(816)	sameStars
0.31	(3678)	nameCosineSimilarity
0.29	(352)	sameImages
0.28	(1719)	addressCosineSimilarity
0.28	(1524)	providers
0.27	(1404)	expandedAddressCosineSimilarity
0.27	(618)	sameGoogleAddressNumber
0.27	(1012)	googleLevenshteinDistance
0.24	(780)	sameAddressNumber
0.21	(332)	isSameGoogleAddress
0.18	(48)	sameUnit
0.18	(111)	sameLevel

TABLE 5.1: Attribute importance by Weka random forest

[1] raw output from Weka

5.1.2 xgboost Attribute Importance

The attribute importance calculated for the xbgoost is shown in figure 5.1 [2]. According to it the attribute *sameUnit* has 0 value. A lot of them have an importance near 0 except for *nameCosineSimilarity*,*distance* and *isSameGoogleAddress*.

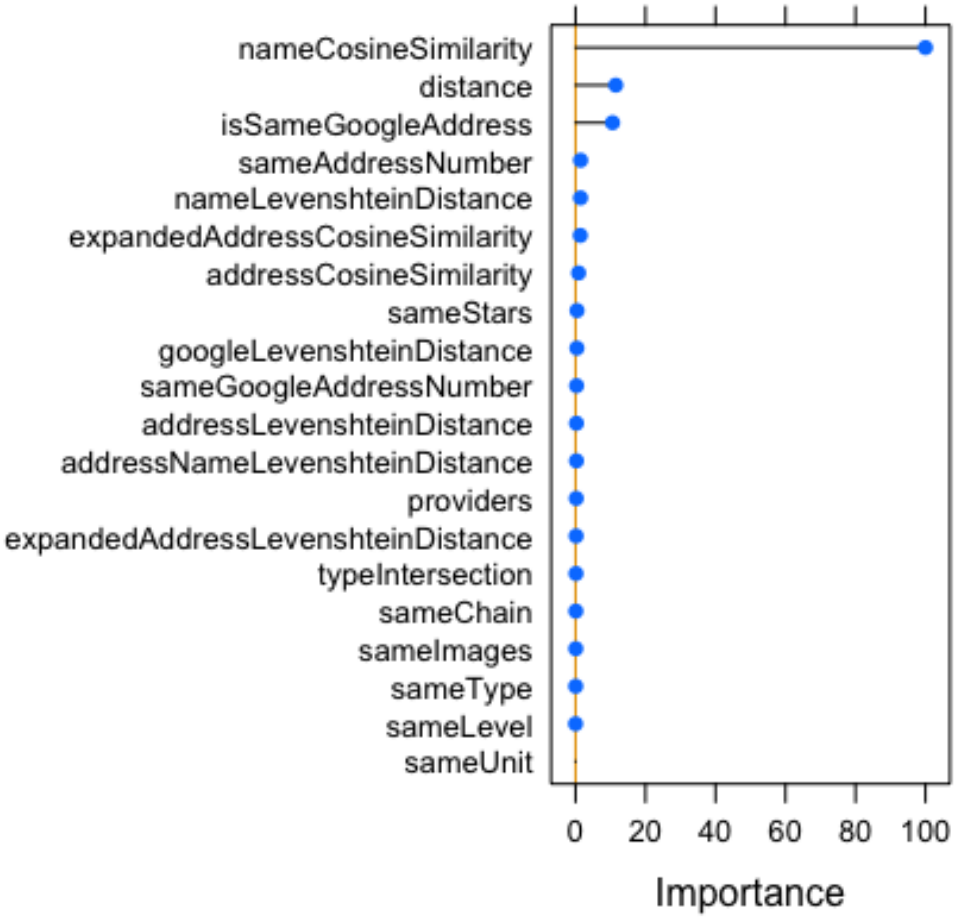


FIGURE 5.1: Attribute Importance xgboost

5.2 Clustering Strategies

First, we can try to answer the question: **is it possible a priori to know what strategy suits better for our deduplication?**
The answer would be that *we must consider too many parameters in order to get an approximation.*
It is worthy in our case to apply the different strategies and check the results.
Some of the different parameters that can affect our strategies are:

- **precision and recall**

[2] raw output from R

- **distribution of erroneous vertex degree**[3]
- **real cluster size proportion**

5.2.1 Precision and Recall

If we see our classifier as a probabilistic function of assessing the right class for a pair, we can say that in our Adaboost classifier a 99.3% (recall, r) of times it classifies a pair as a duplicate correctly. In addition, a 0.7% of times (1 - precision, p) it classifies a pair as a duplicate when it is not.
Let's focus on our classifier recall and its impact in the strategy. Let's suppose we have

$$497 \cdot \binom{2}{2} + 399 \cdot \binom{3}{2} + 1182 \cdot \binom{4}{2} = 8786$$

We have 497 clusters of 2 elements, 399 clusters of 3 elements and 1182 clusters of 4 elements. 8786 pairs.
The expected occurrence (X) of a cluster of size s not being complete (a clique) because of a miss-classification error (recall) would be:

$$E(X_s) = N(1 - (1 - \frac{1}{N})^n) \tag{5.1}$$

$$n = (1 - r) \cdot N \cdot \binom{s}{2}$$

Being N = number of clusters of size s.
We can approximate the expected number of incomplete clusters applying the equation 5.1 as:

$$\sum_{s=2}^{4} E(X_s) \approx 3.47 + 8.3 + 48.64 \approx 60.41$$

With a 99.3% of precision we also get 8786 pairs classified as a duplicate. From those only a 99.3% will be truly a duplicate.
Analogously to get the number of clusters of size s being oversized (having edges to vertices not in the original cluster) we substitute p (precision) for r (recall). However, we have another component, the number of isolated elements, lets suppose a 30% (P_c) of our accommodations are isolated (I). In our example that would be:

$$I = P_c \cdot \sum_{s=2}^{4} s \cdot N_s$$

$$I = 0.3 \cdot (497 \cdot 2 + 399 \cdot 3 + 1182 \cdot 4) = 2076$$

We don't know either the comparisons configuration. We will assume that there is no blocking predicate at all, our total comparisons would be

$$\binom{8996}{2} = 40459510$$

We know that we have $0.007 \cdot 8996 \approx 63$ erroneous pairs distributed along the $40459510 - 8786 = 40450724$ pairs.

[3] In graph theory, the degree (or valency) of a vertex of a graph is the number of edges incident to the vertex.

The proportion of elements by size would be:

- 23.08% for size 1
- 11.05% for size 2
- 13.31% for size 3
- 52.56% for size 4

We could expect that the erroneous pairs maintain the previous percentages.

- 15 erroneous pairs within clusters of size 1
- 7 erroneous pairs within clusters of size 2
- 8 erroneous pairs within clusters of size 3
- 33 erroneous pairs within clusters of size 4

Nevertheless, the elements will be paired with other elements already clustered, except for the isolated, so the total number would be $\approx$ half of the numbers above, without considering that one single element could be paired several times.
As said in the beginning we have to know too much detail in order to do a right guessing.

5.2.2 Erroneous Vertices Degree

The purpose of the previous calculations was to illustrate the impact of precision and recall in our strategy. However, the final solution is more complex than that. For example, the precision doesn't act randomly at a pair level, not all the pairs have the same chance of being miss-classified, some elements are incomplete or have weird data and any comparison done for that element even in the same "real" cluster will result in being classified as false.
Let's suppose that we have an element with a completely wrong address that is part of a cluster of size 4. In theory it doesn't matter if you loss one edge for one pair, we still would have 2 more edges connected to the rest of the elements of the clusters. However, as said before, there exist the possibility of not having any edges because it is the element itself what is different from the rest of the cluster. Even worse, it is possible that if it fails with one of the elements it could fail for the rest if they are similar. The same principle applies to the error of adding edges, if some element is classified as true for a comparison with a cluster it is possible that comparisons with other elements in the cluster could be also classified as a duplicate.
Example:
Having a sample of clusters with the following distribution:

- **Size 2:** 100
- **Size 3:** 91
- **Size 4:** 248

The number of elements will be: $100 \cdot 2 + 91 \cdot 3 + 248 \cdot 4 = 1465$
Those clusters should be clique so our degree[4] distribution should be:

[4]the degree of a vertex is the number of edges incident to the vertex

- **degree 1:** $\frac{100 \cdot 2}{1465} = 0.1365188$
- **degree 2:** $\frac{91 \cdot 3}{1465} = 0.1863481$
- **degree 3:** $\frac{91 \cdot 3}{1465} = 0.6771331$

If the blocking predicate includes all the pairs that are duplicated the distribution should be the one expressed by the previous figures, but it is not the case, it is very close but the real distribution is:

- **1 edges:** 0.1378840
- **2 edges:** 0.1918089
- **3 edges:** 0.6703072

If we apply our xgboost model, our predicted degree distribution is:

- **1 edges:** 0.1468579
- **2 edges:** 0.2076503
- **3 edges:** 0.6454918

In this case, our model predictions has less vertices of degrees 3, and the 2 degrees vertices and 1 degree vertices have been increased.
The degree distribution for the false negatives is:

- **1 edges:** 0.81395349
- **2 edges:** 0.16279070
- **3 edges:** 0.02325581

That means that contrary to what we said before there is a high chance that the false negative errors don't remove completely a vertex for big size clusters, like the 3 edges clusters (4 elements).

5.2.3 Clique

Regarding the splits, the clique strategy have big chances of removing edges not pertaining to the original graph, but it will greatly depend in the probability of a wrong edge (miss classification) having a weight (prediction) greater than any other edge in the graph. However, none of this applies to the case of generating a cluster from a pair of single isolated elements. The clique strategy wouldn't cut these cases. We haven't treated the case of mixing the errors of missing edges and wrong added edges, neither the case where we have more than one of those errors, the impact for our case is not big enough to take it into account. For the sake of simplicity, we have omitted it.
We can conclude that the clique strategy:

- has very low tolerance to bad recall; one single missing edge causes a split of the graph.

- it is biased, tends to broke edges, reducing the final splits but potentially increasing the merges because we have no control of the edges selected to cut. However, it will depend on the probability that a wrong edge has a weight (capacity) greater than a right one.
- it doesn't have an impact in clusters of size 2, a pair will always be a clique.
- It will give us in return a good precision (less splits). In order to add a wrong element to a cluster, the classifier has to classify as duplicate that element with the total of the rest of the elements in the clusters. For big size clusters, it is even harder that any error could be unnoticed.
- as we don't compare accommodations from the same providers, it is not possible to have a clique with the provider repeated. Originally, no comparisons with the same providers are done, so, that case won't ever be a cluster.

5.2.4 Basic

The number of splits and merges will be proportional to the number of clusters oversized and incomplete.
Conclusions:

- The cluster is not enforced to be compliant with our blocking predicate, as it assume transitiveness, there are relations in the cluster that were not even compared, like the same provider repeated.
- It isn't biased, the number of splits and merges should be directly proportional to the recall and precision because don't apply any correction from the pair classification results, In other words, we don't cut neither add new edges to the graph built based in the pair classification.
- Robust to missing edges. It doesn't need all the edges in order to cluster a graph.

5.2.5 Providers

The strategy will correct the clusters oversized that have more than one provider.
Conclusions:

- It is biased, tends to broke edges.
- It enforces the providers to not be repeated. Whenever we have two vertex with the same provider attribute, that graph is broken.
- Similarly to the basic strategy the graphs with missing edges should be auto corrected whenever there are not too many missing that split the graph. The bigger the cluster size the more robust it is.

5.3 Min-cut Error analysis

The *max flow / min cut* implementation that we use have some limitations to our interests. Let say that in a graph we have several absolute predictions ($= 1$). The algorithm could cut some of these edges if there is not a better option.

Let's comeback to an example in previous chapter.
We have the Table 5.2 defining the edges, vertices and capacity for the graph shown in the Figure 5.2. If we try to partition this table applying our cut algorithm we will cut A1–D1, it is mandatory, it has a capacity of 0.9. After that, there are two options for cutting. We can cut either A1–B1 or B1–D1. By default it cuts B1–D1, the algorithm implementation used doesn't specify the behavior for this scenario. Having a capacity of 0.9999 in A1–B1 would force A1–B1 to be cut. We haven't provided any solution for this scenario.

TABLE 5.2: Table representation for problematic graph

	from	to	capacity
1	A1	B1	1
2	B1	D1	1
3	A1	D1	0.9
4	A1	B2	1
5	A1	C2	1
6	B2	C2	1

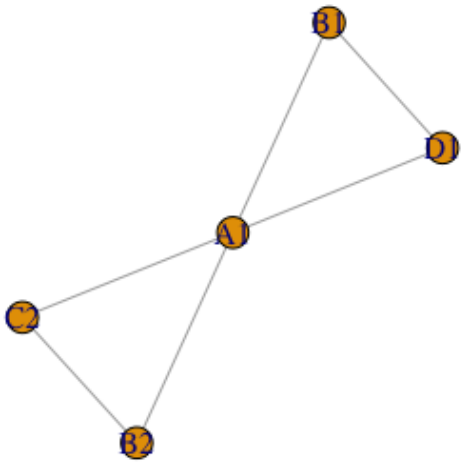


FIGURE 5.2: Problematic partitioning

5.4 Deduplication Update

We can have a deduplication scenario where there is already a deduplication that isn't complete. We can adopt several strategies in order to incorporate our results to an existing base depending on the confidence that we have for the existing one.

For example, we will usually have some data and we incrementally add elements to that base. . Either any new inclusion will be part of an existing cluster or we have to create a new one. We could have several alternatives:

- Deduplicate all again from scratch.
- Compare the new element with the nearest elements or clusters and update in case of being necessary.

The first option could have a big cost depending on the rate of new elements arriving. And the last one implies comparisons in some cases redundant over time. In cases of having incongruences in the last option usually due to splitting of existent clusters, we have to act accordingly and take into account our confidence in the existing clusters. We can opt for ignoring the splits, modifying the clusters or reviewing manually.
In our case we have seen that applying the clique strategy and merging any non-conflicting result with the basic strategy gives the same results as the provider strategy, but no further improvements has been achieved using the provider strategy.

Chapter 6

Conclusion and future work

6.1 Conclusions

We have analyzed the deduplication problem in accommodations from the machine learning perspective. Some of the contributions done compared to past works are:

- A new definition for blocking
- A dynamic blocking strategy based on distance
- New features based on address standardization and decomposition
- A framework for the commons phases in the deduplication process. It includes also utilities for calculating generic features, dataset creation, model creation and evaluation.
- We showed that Adaboost and Xgboost (not previously considered) can have a good performance comparable to random forest.
- Emphasis in the clustering phase
- A resolution strategy for clustering with GMD
- We provide a reinterpretation and implementation of GMD

6.2 Future work

Entity resolution in different scopes sometimes has commons elements. It would be interesting the possibility of generating models as reusable components that can be composed in order to build a final model. For example:
The address resolution for accommodations will possibly be the same as the address resolution for a restaurant, a hospital, etc... Is it possible to generate a model that given two address it return the similarity? We have to take into account that only for address we have 10 different features without considering the geo-location.
The name is a similar case. We could consider that the name of a person is different to the name of an accommodation, but is it different the name of an accommodation to the name of a restaurant? possibly there are common strategies in order to learn the similarity for a restaurant name.
With the existence of those reusable components, maybe it would be enough with a dozen of features in order to deduplicate any commercial premise as aimed in other works (Zheng et al., 2010).
Currently our framework is coupled with Weka. However, there are other ML libraries also supported for Java. We have seen that xgboost has a good performance

but we have not generated a model for xgboost even when there is a Java implementation for it. We have here several alternatives:

- Make the framework generic and add extensions in order to support several implementations, which implies that we will have a cost to add the extensions.
- Give up on the idea of having the entire pipeline within the framework and accept that any step of the pipeline (blocking, feature deriving, dataset generation, train and evaluations) could be done outside. That is currently supported but has some limitations. It will be more complex to tune the model according to our cluster evaluation if the evaluation is done with the framework but we build the model outside[1].
- If we do the previous one should we rely on Weka or use some standardized representation? Like PMML (Predictive Model Markup Language) and assume that the model will always be provided in that format.

[1] In fact this is what we did with xgboost

Appendix A

GMD considerations

In "Evaluating Entity Resolution Results" (Menestrina, Whang, and Garcia-Molina, 2010) there are some definitions that we change.
The definition for a split operation given is:
Definition 4.1: *A split is an operation $c \rightarrow c1, c2$ where $c1 \cap c2 = \emptyset$, $c1 \cup c2 = c$, and $c1, c2 \neq \emptyset$. The result of applying a split to a partition P is (P - { c }) ∪ { c1, c2 }. A split is a valid operation on P if and only if $c \in P$.*

My proposal is:
A split is an operation $c \rightarrow c1, c2$ .. Cn where $c1 \cap c2 \ldots \cap Cn = \emptyset$, $c1 \cup c2 \ldots \cup Cn = c$, and c1 ,c2… Cn$\neq \emptyset$. The result of applying a split to a partition P is (P - { c }) ∪ { c1 ,c2… Cn }. A split is a valid operation on P if and only if $c \in P$.

Analogously to the merge operation.

Why? Because of:
Definition 5.7.: *We say that a function F is operation order independent if it satisfies F(x, y) + F(x + y, z) = F(x, z) + F(x + z, y) for all x, y, z.*

With the new proposal, there is no need for several splits / several merges hence no need to satisfy order independent property.
Another auto-imposed limitations is that the functions receive the resulting splits or sourcing merges size, not the cluster itself, it could be useful to redefine it for clusters not size because other properties of the cluster could be useful to calculate the cost.
The changes to adapt the current algorithm to another one following these changes are minimal and doesn't affect the performance.
The basic idea of the algorithm is conceptually quite simpler than the original algorithm:

Definition. *Given the non-empty intersections between the clusters in an entity resolution E, and the clusters in the ground truth G. Those intersections can be grouped by the source clusters (splits) or by the target clusters (merges). The GDM cost will be the sum of applying the split and merge cost functions to those groups respectively.*

Doing it efficiently is a matter of indexing/hashing the elements.[1]
In the original paper an example is include that we are going to reproduce with our algorithm variation. The example is:

$$E = \{\langle a, c, e\rangle, \langle b, d, f\rangle\}$$

$$G = \{\langle a, b, c\rangle, \langle d, e, f\rangle\}$$

[1] https://github.com/francetem/deduper/blob/master/src/main/java/org/ehu/dedupe/metrics/distance/GMD.java contains a Java implementation

The first step is get the intersections of the clusters within E with the clusters within G.

$$\{(\langle a,c,e\rangle \cap \{\langle a,b,c\rangle), (\langle a,c,e\rangle \cap \{\langle d,e,f\rangle\ ), (\langle b,d,f\rangle \cap \{\langle a,b,c\rangle\ ), (\langle b,d,f\rangle \cap \{\langle d,e,f\rangle\ )\}$$

$$I = \{\langle a,c\rangle, \langle e\rangle, \langle b\rangle, \langle d,f\rangle\ \} \tag{A.1}$$

Equation A.1 shows the resulting intersections (I). If we group those intersections by the predicted clusters in E we get the splits:

$$e_1 = \{\langle a,c\rangle, \langle e\rangle\ \}$$
$$e_2 = \{\langle b\rangle, \langle d,f\rangle\ \}$$

If we group those intersections by the real clusters in G we get the merges:

$$g_1 = \{\langle a,c\rangle, \langle b\rangle\ \}$$
$$g_2 = \{\langle e\rangle, \langle d,f\rangle\ \}$$

Given a split cost function f_s and a merge cost function f_m the final cost would be:

$$f_s(e_1) + f_s(e_2) + f_m(g_1) + f_m(g_2)$$

For the basic merge distance f_s and f_m would be defined as

$$f_s(x) = f_m(x) = |x| - 1$$

The final value would be = 4.

Appendix B

Resources

Attached we can find the following resources:

- **all.arff** contains a dataset with after applying the blocking and feature deriving in arff format
- `https://github.com/francetem/deduper` link to the deduper framework, contains generic utilities that have been use to do the blocking and generate the dataset, can be used also to train and evaluate models in Java.
- **gmd.R** contains the gmd VI variation.
- **xgb.R** train and evaluate a xgb using all.arff file and compute attribute importance
- **rf.R** train and evaluate a rf using all.arff file and compute attribute importance

Appendix C

High Density Zones

We know that the following scenario exists: the distribution of the dataset could have high-density zones and our blocking is based on distance, so for those zones the number of comparisons can be very high. If we reduce the maximum distance, it will affect to every single accommodation independently of the zone because we don't take into account the density.

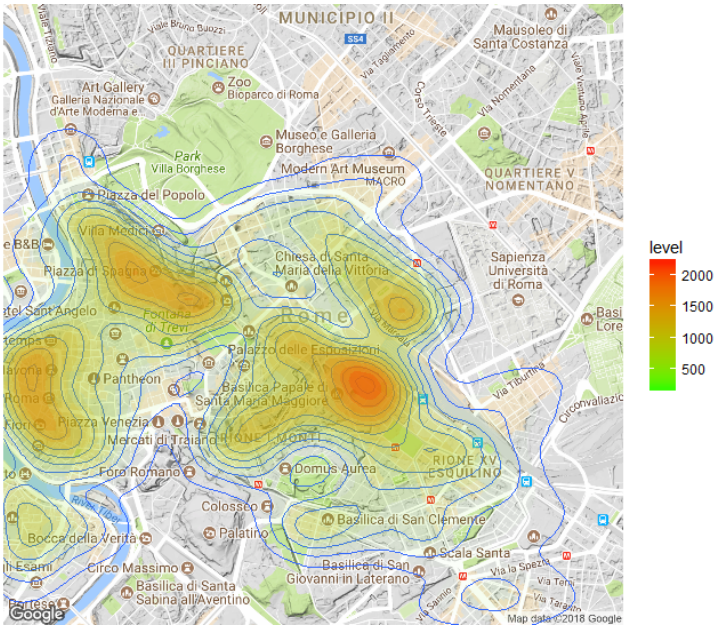


FIGURE C.1: Rome density

The figure C.1 illustrates the density for the center of Rome.
The ideal is that the level of granularity should be small enough to avoid this problem. In other words, instead of choosing a whole city to compare, we could decide to use a zone within that city. We could deduplicate at a neighborhood level instead of city for example. If it is not the case, one option is to create clusters of data by distance using some clustering algorithm like canopy, EM, Kmeans. The one that fits best to our expectations. In (Christen, 2012) we can find several related blocking techniques. These options would be a previous step to our blocking predicate.
Figure C.2 shows the centers for a kmeans clustering run. The clustering sometimes is a little tricky because we can in most of the cases; specify the number of clusters to use, but not the size of those clusters. In table C.1 we can see an example of the disparity of lengths. We have always the option of clustering iteratively any group bigger than a specific size. Alternatively, we can apply some logic to a hierarchical cluster.

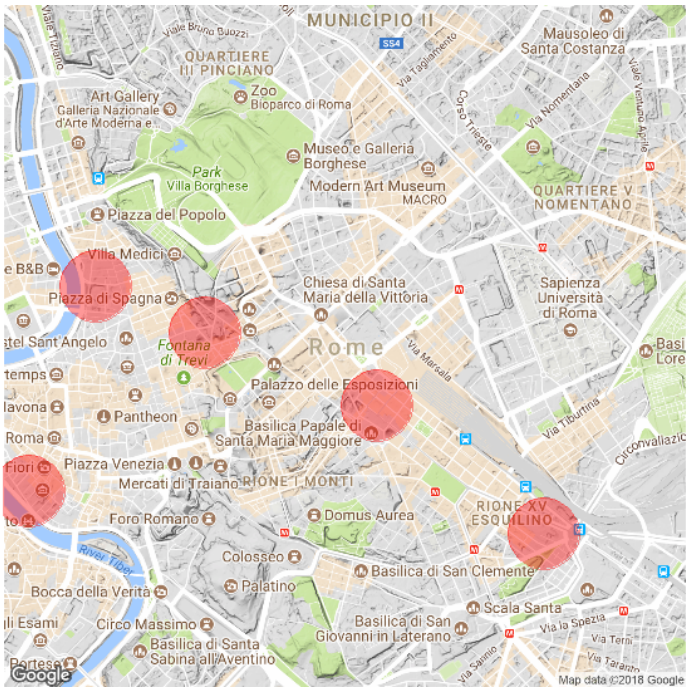


FIGURE C.2: Rome kmeans centers

TABLE C.1: Kmeans clusters size and center

	lat	lon	size
1	41.87	12.47	849
2	42.02	12.35	60
3	41.89	12.47	2623
4	41.91	12.46	2264
5	41.90	12.45	1643
6	41.81	12.46	162
7	41.88	12.54	318
8	41.92	12.53	609
9	41.90	12.42	443
10	41.00	12.00	7
11	41.87	12.45	311
12	41.95	12.42	182
13	41.90	12.48	2289
14	41.85	12.57	227
15	41.86	12.65	96
16	41.93	12.58	105
17	41.76	12.53	50
18	41.90	12.49	1914
19	42.00	12.50	92
20	41.91	12.50	1474
21	41.74	12.31	254
22	41.88	12.51	906
23	41.84	12.36	98
24	41.93	12.47	321
25	41.89	12.51	1747

Another option is to limit the number of comparisons using a logic like the X *nearest* accommodations. In this option it is important to have into account that the X nearest accommodations for an accommodation, let's say A, could include another one B, but the opposite won't always be true, B could not include A as one of the nearest accommodation. This is important in order to have a blocking algorithm not dependent on the order.

Appendix D

Workflow Measurements

A deduplication process includes several phases or steps. All of those phases can be evaluated individually but previous steps affects the next steps. That means that if for example there is a bad blocking, some duplicates comparisons won't be done, the fact of missing comparisons will affect the pair classification, missing pairs and miss classifying pairs will also result in a worst clustering.

D.1 Pair Classification / Clustering Evaluation

There is a link between precision and recall as metric for the pair classification and GMD as measure for the clustering. The pair classification generates duplicate relationships between entity representations. The results from the pair classification can be considered edges in the graph pre-clustering, being the vertex the elements.
If there is a perfect precision and recall any reasonable strategy applied will give us a perfect clustering because all the edges would be detected and no one would be miss classified. so there would be no need to split any graph neither merge.
If the precision is perfect, it means that there is no need to cut any edge. So no splits are needed.
If the recall is perfect, there is no need to merge any graph. All the edges between elements that are the same would be identified.
In conclusion: the precision is directly related to splits and recall is related to merges, a low recall requires merges to be done and a low precision will result in more splits. All of this only applies if the blocking predicate provides all the relevant pair comparisons.

D.2 Blocking Predicate / Clustering evaluation

If the elements {A1, B1, C1} are duplicates and the blocking predicate generates the comparisons {A1, B1} and {B1, C1}, and if the classifier detects all duplicates the graph in figure D.1 will be generated after the pair classification.
Depending on the chosen strategy there can be a merge in the cluster evaluation because there is a missing edge between A1 and C1.
To conclude: the blocking predicate could potentially have the same effect as the precision. The less it pairs duplicated elements the more merges will be needed in the clustering evaluation.

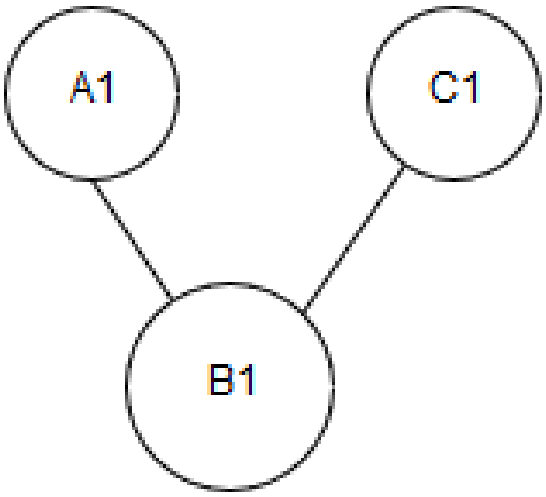


FIGURE D.1: Uncomplete graph

R GMD (With Modifications) VI

Francis Perez

03-06-2018

```
#auxiliar function to calculate the cost for merges or splits
calculateCost <- function(to, by, costFunction) {
  splits <- split(to, by)
  cost <- 0
  for(n in names(splits)){
    groups <- splits[[n]]
    intersections <- split(groups$elem, groups[,1])
    cost <- cost + costFunction(intersections)
  }
  cost
}

#cl1 and cl2 must have the same length
gmd <- function(cl1, cl2, fs, fm){

  intersections <- data.frame(stringsAsFactors=FALSE)

  for(i in 1:length(cl1)){
    intersections <- rbind(intersections, c(cl1[i], cl2[i], i))
  }

  names(intersections) <- c("ori", "dest","elem")

  to <- intersections[,c(2,3)]
  by <- intersections[,1]

  splitCost <- calculateCost(to, by, fs)

  to <- intersections[,c(1,3)]
  by <- intersections[,2]

  mergeCost <- calculateCost(to, by, fm)

  return (c(splitCost, mergeCost))

}

#Variation of Information function
vi <- function(N){

  target <- function(elems){
    sizes <- sapply(elems, length)
    total <- sum(sizes)

    freqs <- sizes / total

    entropies <- -(sizes * sapply(freqs, log2)) / N
```

```
    return(sum(entropies))
  }
  return(target)
}

#use sample, compared to another library for calculating VI

library(mcclust)
```

```
## Loading required package: lpSolve
```

```
numElements <- 10

cl1 <- sample(1:3,numElements, replace=TRUE)
cl2 <- c(cl1[1:5], sample(1:3,5,replace=TRUE))

#vi.dist value
vi.dist(cl1,cl2)
```

```
## [1] 1.836453
```

```
#gmd vi value
gmdCost <- gmd(cl1, cl2, vi(numElements), vi(numElements))

#splits + merge cost
gmdCost[1] + gmdCost[2]
```

```
## [1] 1.836453
```

XGB

Francis Perez

03-06-2018

```
#allows to read arff files
library(foreign)

all <- read.arff("all.arff")

library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.2
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'default/
## Europe/Madrid'
```

```
library(xgboost)
```

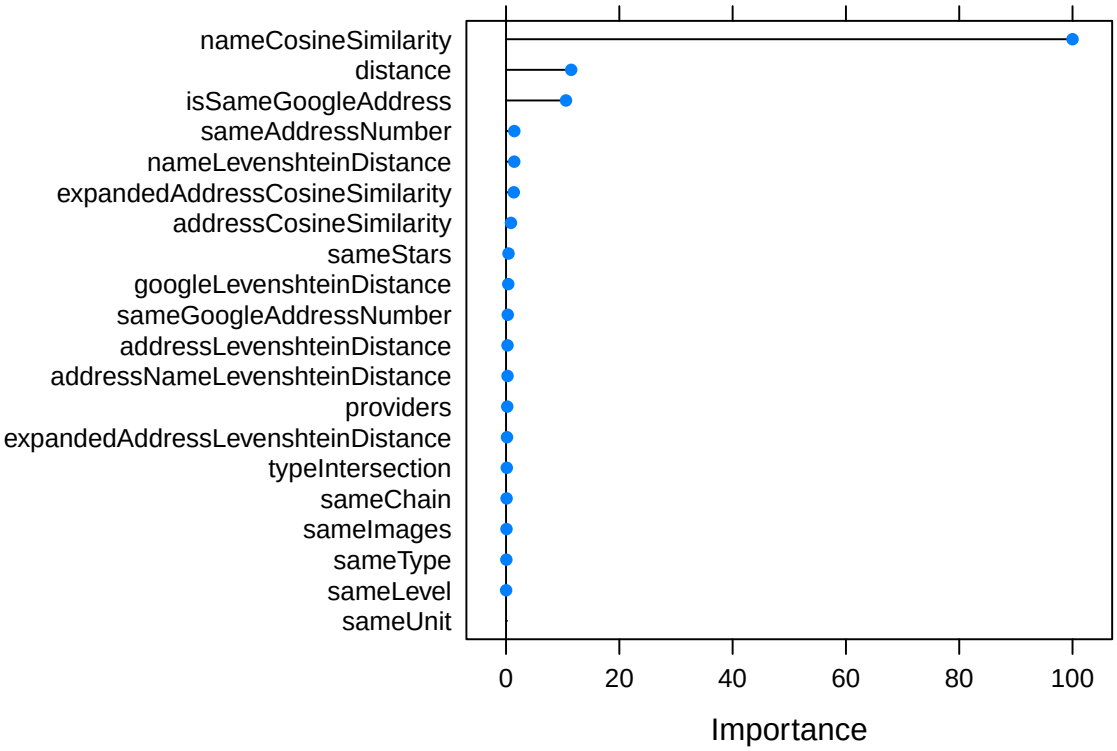
```
## Warning: package 'xgboost' was built under R version 3.3.2
```

```
#xvalidation 10 folds
ctrl <- trainControl(method = "cv", number = 10)

#fix values
xgb.grid <- expand.grid(nrounds = c(150),
                        lambda = c(1e-04, 1),
                        alpha = 0,
                        eta = c( 0.3))


predictors <- all[,3:22]
xgb.train <- train(
  x = data.matrix(predictors),
  y = data.matrix(all[,23]),
  trControl = ctrl,
  method = "xgbLinear",
  tuneGrid = xgb.grid,
  verbose = 1
)

#Print confusion matrix
confusionMatrix(xgb.train, norm = "none")
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##            Reference
## Prediction  false   true
```

```
##       false 131166     74
##       true      60   8702
##
##  Accuracy (average) : 0.999
```

```
plot(varImp(xgb.train))
```



RF

Francis Perez

03-06-2018

```
#allows to read arff files
library(foreign)

all <- read.arff("all.arff")

library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.2
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'default/
## Europe/Madrid'
```

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

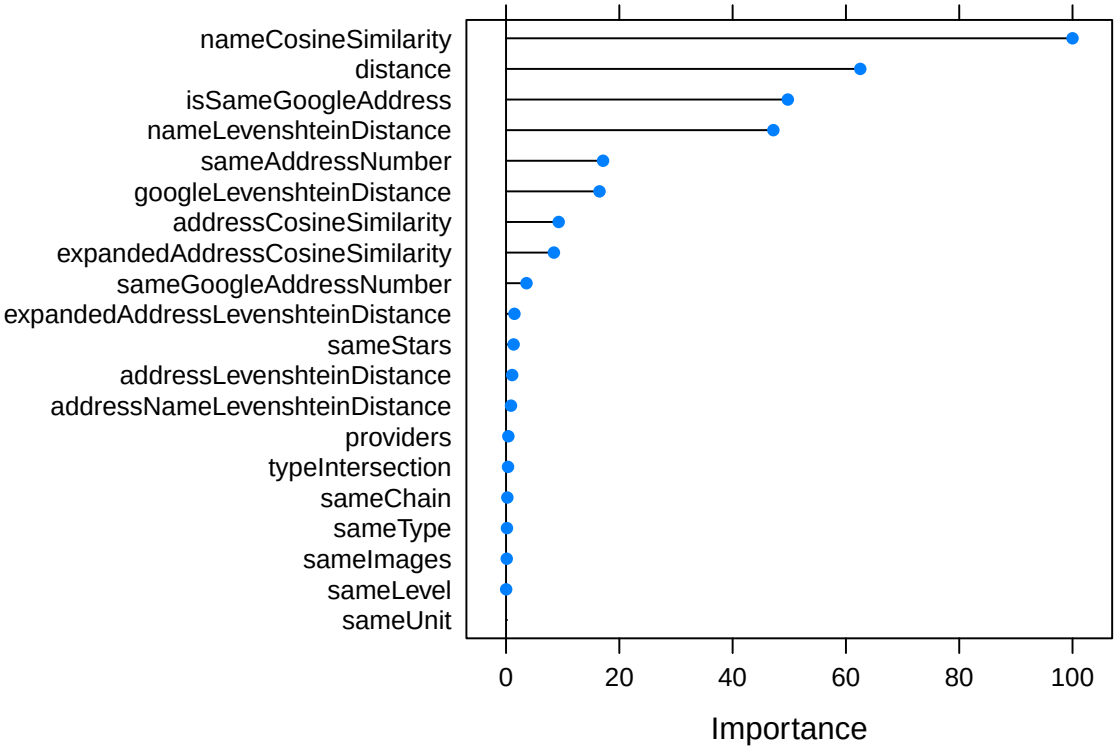
```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
#xvalidation 10 folds
ctrl <- trainControl(method = "cv", number = 10)

#fix values
rf.grid <- expand.grid(mtry = 6)


predictors <- all[,3:22]
rf.train <- train(
  x = data.matrix(predictors),
  y = data.matrix(all[,23]),
  trControl = ctrl,
  method = "rf",
  tuneGrid = rf.grid,
  ntree = 99,
  verbose = 1
)

#Print confusion matrix
confusionMatrix(rf.train, norm = "none")
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##           Reference
## Prediction  false   true
##      false 131155     89
##      true      71   8687
##
##  Accuracy (average) : 0.9989
```

```
plot(varImp(rf.train))
```



Bibliography

Aha, David W, Dennis Kibler, and Marc K Albert (1991). "Instance-based learning algorithms". In: *Machine learning* 6.1, pp. 37–66.

Benjelloun, Omar et al. (2009). "Swoosh: A Generic Approach to Entity Resolution". In: *The VLDB Journal* 18.1, pp. 255–276. ISSN: 1066-8888. DOI: 10.1007/s00778-008-0098-x. URL: http://dx.doi.org/10.1007/s00778-008-0098-x.

Bottou, Léon (2010). "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

Bouckaert, Remco Ronaldus (1995). "Bayesian belief networks: from construction to inference". PhD thesis.

Breiman, Leo (1996). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

— (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Bron, Coen and Joep Kerbosch (1973). "Algorithm 457: finding all cliques of an undirected graph". In: *Communications of the ACM* 16.9, pp. 575–577.

Chen, Tianqi, Tong He, Michael Benesty, et al. (2015). "Xgboost: extreme gradient boosting". In: *R package version 0.4-2*, pp. 1–4.

Christen, P. (2012). "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication". In: *IEEE Transactions on Knowledge and Data Engineering* 24.9, pp. 1537–1555. ISSN: 1041-4347. DOI: 10.1109/TKDE.2011.127.

Cooper, Gregory F and Edward Herskovits (1991). "A Bayesian method for constructing Bayesian belief networks from databases". In: *Uncertainty Proceedings 1991*. Elsevier, pp. 86–94.

Dalvi, Nilesh et al. (2014). "Deduplicating a places database". In: *Proceedings of the 23rd international conference on World wide web*. ACM, pp. 409–418.

Freund, Yoav, Robert E Schapire, et al. (1996). "Experiments with a new boosting algorithm". In: *Icml*. Vol. 96. Bari, Italy, pp. 148–156.

Friedman, Jerome H (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*, pp. 1189–1232.

Getoor, Lise and Ashwin Machanavajjhala (2012). "Entity resolution: theory, practice & open challenges". In: *Proceedings of the VLDB Endowment* 5.12, pp. 2018–2019.

Hare, Jonathon S., Sina Samangooei, and David P. Dupplaw (2011). "OpenIMAJ and ImageTerrier: Java Libraries and Tools for Scalable Multimedia Analysis and Indexing of Images". In: *Proceedings of the 19th ACM International Conference on Multimedia*. MM '11. Scottsdale, Arizona, USA: ACM, pp. 691–694. ISBN: 978-1-4503-0616-4. DOI: 10.1145/2072298.2072421. URL: http://doi.acm.org/10.1145/2072298.2072421.

Kozhevnikov, Ivan and Vladimir Gorovoy (2016). "Comparison of Different Approaches for Hotels Deduplication". In: *Knowledge Engineering and Semantic Web*. Ed. by Axel-Cyrille Ngonga Ngomo and Petr Křemen. Cham: Springer International Publishing, pp. 230–240. ISBN: 978-3-319-45880-9.

Köpcke, Hanna and Erhard Rahm (2010). "Frameworks for entity matching: A comparison". In: *Data & Knowledge Engineering* 69.2, pp. 197 –210. ISSN: 0169-023X. DOI: https://doi.org/10.1016/j.datak.2009.10.003. URL: http://www.sciencedirect.com/science/article/pii/S0169023X09001451.

Le Cessie, Saskia and Johannes C Van Houwelingen (1992). "Ridge estimators in logistic regression". In: *Applied statistics*, pp. 191–201.

Meilă, Marina (2007). "Comparing clusterings—an information based distance". In: *Journal of multivariate analysis* 98.5, pp. 873–895.

Menestrina, David, Steven Euijong Whang, and Hector Garcia-Molina (2010). "Evaluating Entity Resolution Results". In: *Proc. VLDB Endow.* 3.1-2, pp. 208–219. ISSN: 2150-8097. DOI: 10.14778/1920841.1920871. URL: http://dx.doi.org/10.14778/1920841.1920871.

Quinlan, Ross (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Ristad, Eric Sven and Peter N Yianilos (1998). "Learning string-edit distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.5, pp. 522–532.

Sarawagi, Sunita and Anuradha Bhamidipaty (2002). "Interactive deduplication using active learning". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 269–278.

Stoer, Mechthild and Frank Wagner (1997). "A simple min-cut algorithm". In: *Journal of the ACM (JACM)* 44.4, pp. 585–591.

Ting, Kai Ming (2010). "Precision and Recall". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, pp. 781–781. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_652. URL: https://doi.org/10.1007/978-0-387-30164-8_652.

Wang, Hongzhi (2014). *Innovative Techniques and Applications of Entity Resolution*. 1st. Hershey, PA, USA: IGI Global. ISBN: 1466651989, 9781466651982.

Whang, Steven Euijong et al. (2009). "Entity resolution with iterative blocking". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, pp. 219–232.

Wikipedia contributors (2018a). *Cluster analysis — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-June-2018]. URL: https://en.wikipedia.org/w/index.php?title=Cluster_analysis&oldid=845086044.

— (2018b). *Levenshtein distance — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-July-2018]. URL: https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=846659618.

Zheng, Yu et al. (2010). "Detecting nearly duplicated records in location datasets". In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, pp. 137–143.