ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Entity Resolution in Historical Documents for Family Tree Discovery

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil Informatique à Finalité Web & Information Systems

Aurélien Plisnier

Directeur & Superviseur
Professeur Toon Calders

Service
Computer & Decision Engineering Department

Année académique
2014 - 2015

Entity Resolution in Historical Documents for Family Tree Discovery

Mémoire Présenté en Vue de l'Obtention du Diplôme d'Ingénieur Civil Informatique à Finalité Web & Information Systems, Promotion 2015

Aurélien Plisnier

Résumé Liminaire

Ce mémoire a pour objet l'étude et l'implémentation de procédés d'"entity resolution", aussi appelés "data matching" ou encore "record linkage" qui ont pour but d'associer des entrées faisant référence aux mêmes objets ou personnes mais appartenant à des bases de données différentes. Pour cela sont utilisés des attributs partiellement identifiants tels que des dates de naissance, des adresses, des noms et des prénoms.

Notre objectif est d'associer entre eux des certificats de naissance, mariage et décès faisant référence aux mêmes personnes de manière à faciliter des recherches généalogiques et historiques. Emis aux Pays-Bas entre 1580 et 1811, ces certificats ont été initialement manuscrits par des prêtres puis plus récemment numérisés par des volontaires du centre de recherche BHIC [1]. Disponibles au format XML, ils sont bien souvent de piètre qualité: des fautes d'orthographe apparaissent souvent dans les noms de lieux et de personnes, des dates ne respectent pas le format standard, chacun applique sa propre convention de nommage, des champs ne sont pas remplis ou sont utilisés à mauvais escient, ...

Mais la plus grande difficulté dans notre cas vient de l'absence de "training data", ou "gold standard": ce sont des paires de certificats pour lesquels nous sommes sûrs qu'ils réfèrent bien à la même personne. En effet, la vaste majorité des procédés actuels d'entity resolution fait appel à un gold standard, sur base duquel sont appris les paramètres permettant de déterminer au mieux si deux certificats donnés réfèrent oui ou non à la même personne. Ainsi, la qualité des paires découvertes lors du procédé d'entity resolution dépend fortement de la qualité du training set. L'obtention d'un gold standard, recouvrant toutes les configurations accessibles aux attributs d'une paire de certificats tout en présentant le plus faible biais possible, est une tâche difficile, longue et rébarbative impliquant en général l'inspection à la main de plusieurs milliers de paires de certificats.

Ne possédant pas les ressources nécessaires à la mise en oeuvre d'une telle tâche, il nous faut donc concevoir, implémenter et évaluer une technique alternative d'entity resolution ne requérant pas de gold standard. Elle fait appel à un modèle statistique bayésien substituant à celui-ci des distributions de probabilités conditionnelles obtenues empiriquement. Nous verrons que si cette technique permet réellement de se passer de gold standard et conduit à des résultats encourageants, elle ne peut être appliquée qu'à des certificats présentant suffisamment d'attributs.

Keywords. *Genealogy, Multi-Source Entity Resolution, Data Mining, Historical Record Linkage, Bit Vector Tree Indexing, Bayesian Classifier*

[1] Brabants Historisch Informatie Centrum `https://www.bhic.nl/het-geheugen-van-brabant`

Abstract

Entity resolution, also called data matching, record linkage or deduplication is the process of linking together records from different data sources or merging together duplicate records from a single source. This is done by comparing partly identifying features such as person name, birth date, living place, ... The purpose of this thesis is to support genealogical and historical research by linking together birth, marriage and death certificates issued in the Netherlands between years 1580 and 1811. Data contained in those records is often of poor quality as they were initially handwritten by priests and recently manually indexed by volunteers. The main difficulty, however comes from the lack of labeled training data for our data set. Training data are pairs of records for which it is sure whether or not they are matching (i.e. referring to the same entity). As most state of the art entity resolution techniques require a large amount of such training data, a workaround technique allowing to perform entity resolution without training data has to be found. The contribution of this work is the definition, implementation and evaluation of an alternative entity resolution approach based on a Bayesian statistical model: instead of labeled data, it uses conditional probability distributions collected in a data driven fashion based on the information excess principle. We will show that it allows to learn an entity resolution model without training data as long as there are multiple record features available for matching.

Les procédés d'"entity resolution" ou "data matching", "record linkage" ou encore "deduplication" ont pour but d'associer des entrées faisant référence aux mêmes objets ou personnes mais appartenant à des bases de données différentes. Pour cela, on utilise des attributs partiellement identifiants tels que des dates de naissance, des adresses, des noms et des prénoms. L'objectif de ce mémoire est de faciliter des recherches généalogiques et historiques en concevant un procédé d'entity resolution permettant d'associer des certificats de naissance, mariage et décès émis aux Pays-Bas entre 1580 et 1811. Initialement manuscrits par des prêtres puis récemment manuellement numérisés sous format XML par des volontaires, ces certificats sont bien souvent de piètre qualité. La plus grande difficulté vient cependant de l'absence de "training data", ou "gold standard" pour ces données: ce sont des paires de certificats pour lesquels nous sommes sûrs qu'ils réfèrent bien à la même entité. En effet, la vaste majorité des techniques actuelles d'entity resolution fait appel à un gold standard. La contribution de ce mémoire consiste en la conception, l'implémentation et l'évaluation d'une technique alternative d'entity resolution permettant d'associer des certificats sans avoir besoin de gold standard. Elle fait appel à un modèle statistique bayésien substituant à celui-ci des distributions de probabilités conditionnelles obtenues empiriquement. Nous verrons que si cette technique permet réellement de se passer de gold standard, elle ne peut cependant être appliquée qu'à des certificats présentant suffisamment d'attributs.

Contents

List of Figures

Acknowledgment

We thank Toon Calders, Julia Efremova, Bijan Ranjbar-Sahraei, Marijn Paul Schraagen and Alexander Panchenko for their advice, guidance and support throughout this project.

Chapter 1

Introduction

The thesis description taken from the thesis catalog is available in Appendix A. In addition, a short popularising article describing the goals and achievements of this work is included in Appendix B.
As a person's life goes on, certificates recording his birth, marriage, the birth of his children then finally his death are emitted. In the province of North-Brabant in the Netherlands, from around 1580 to the appearance of Napoleonic era certificates in 1811, this task was carried on by priests: they recorded birth, marriage and death certificates in their churches registers. As time went on and people moved on, the life events of every single person were scattered across several certificates, often stored in different registers and churches. Now, those certificates are some of the last traces left by these people.

The purpose of this work is to support genealogical and historical research by linking together those certificates. As no personal identification number was available at the time, partly identifying features such as person names, birth date or living place are to be compared. The linking of those records must allow to trace back the lives of these ancestors and to build their genealogical tree.

There are three kinds of available data sources:

1. Before 1811: **Church records**, called DTB for Doop, Trouw, Begrafenis (Birth, Marriage, Death). At the time, religious authorities where responsible for the upkeep of population registers.
   DTB records are of low quality since they were completely hand written and their structure may vary from parish to parish. Figure 1.1 shows a sample death certificate.

2. After 1811: **Civil certificates** brought by the Napoleonic administration. There are different kinds of certificates: birth, marriage, death, memory of succession and census records. Those certificates consist of a printed text framework which had to be completed by the authority with dates and names. Being more structured, those files are of higher quality and contain more features. Appendix G shows a sample Napoleonic era birth certificate.

3. All years: **Notary acts** are plain text files recording property transfers, debt declarations and so on. 88 different kinds of notary acts are reported in [4]. Appendix F shows a sample testament notary act.

Several researchers are currently working on these data. As such, to not be redundant, our work has to take their previous input into account. Especially the following two projects are of interest:

- **Aspects of record linkage** [14] applies entity resolution techniques to Napoleonic era certificates to link people across birth, marriage and death certificates, thus retracing their lives. This PHD thesis, written by Marijn Schraagen, explores the subject of entity resolution and discusses several graph based techniques.

- **Investigation of a Baseline Method for Genealogical Entity Resolution:** [6] the MISS team tackles the matching of Napoleonic era certificates with notary acts using a state of the art entity resolution technique.

This is on church records that we will focus in this work. Those certificates, all from province of North-Brabant, are being digitized and indexed by hand by volunteers at the Brabants Historisch Informatie Centrum (BHIC) [1]. This resulted in XML files of which samples are given in Appendices C, D and E.

[1] `http://www.bhic.nl`

1.1 Problem Definition

In this work we focus on the matching of DTB records using a genealogical entity resolution schema. It is an application of **entity resolution** (also known as **data maching** or **record linkage**) [3], [7], which is an important research domain in **data mining** and **machine learning**. Entity resolution has been used in many domains such as business intelligence, marketing, digital libraries, medical and social science research, fraud detection, government administration, search engines and social networks. Its purpose is to connect different data sources with different structures and attributes. This allows to aggregate data which was initially scattered across several sources: a classical example is the matching of two commercial databases containing records of a firm's customers. As each database may contain different data about the customers, the merging of the two allows to build a more complete customer profile, enabling a better targeted marketing.
As no common entity identifiers are available, the links are to be found by comparing textual attributes (e.g. names and places) and numbers (dates). An important issue comes from the varying quality of these attributes :

- The original hand written data may contain errors (e.g. misspellings and swaps) and omissions. For example the place field may contain "Asmterda" instead of "Amsterdam" or can simply be empty.
- Name variations or alternatives for persons and places may occur during a life. The table 2.1 from subsection 2.1.2 shows such variations for the names Joosten and Jordanus for instance.
- Volunteers made mistakes when recording a field. In one of our birth certificate for instance, the volunteer filled in both date and place in the field provided for place, which resulted to "29-01-1806Boxmeer" being recorded as place of birth. Also, several birth records have the date "2017-00-00" filed in.

As a result, the process of entity resolution must deal with uncertainty.

The source data is a set of 1 081 532 birth certificates, 363 425 death certificates and 306 701 marriage certificates stored as XML files. These digitized versions of church records were initially handwritten by priests from the province of North Brabant and range from around 1580 to 1811. They were indexed by volunteers who had to read through the badly written certificates. Figure 1.1 shows a picture of a death record.

Needless to say the quality of the data set is quite poor. The goal is to match those certificates together so that the life history of people can be explored. Consistently available information is relatively scarce though, as shown in table 1.1:

- **Birth records** contain first name and last name of newborn, father and mother as well as the date and place of the birth's registration. The data extracted from a sample birth certificate is shown in table 1.2. The corresponding sample XML entry is shown in Appendix C.
- **Marriage records** contain first name and last name of bride and groom as well as the date and place of the marriage's registration. The data extracted from a sample marriage certificate is shown in table 1.3. The corresponding sample XML entry is shown in Appendix D.
- **Death records** contain first name and last name of deceased person as well as date and place of the death's registration. The data extracted from a sample death certificate is shown in table 1.4. The corresponding sample XML entry is shown in Appendix E.

Birth certificate	FirstName, LastName, Gender, BirthDate, BirthPlace, FatherFirstName, FatherLastName, MotherFirstName, MotherLastName
Death certificate	FirstName, LastName, Gender, DeathDate, DeathPlace
Marriage certificate	GroomFirstName, GroomLastName, BrideFirstName, BrideLastName, MarriageDate, MarriagePlace

Table 1.1: Consistently available features for each certificate type

First Name	Arnoldus
Last Name	Giimans
Gender	m
Place of Birth	Deurne
Date of Birth	1797-10-02
Father First Name	Willebrordus
Father Last Name	Giimans
Mother First Name	Maria
Mother Last Name	Verhoeven
Witness First Name	Maria Jan
Witness Last Name	-
Witness First Name	Hendrick
Witness Last Name	van Bree

Table 1.2: An example of data from a birth certificate

Place	Helvoirt
Date	1794-05-04
Date of Notice	1794-04-19
Groom First Name	Willebrordus
Groom Last Name	Giimans
Groom Place of Birth	Diessen
Groom Place of Residence	Helvoirt
Bride First Name	Maria
Bride Last Name	Verhoeven
Bride Place of Birth	Cromvoirt
Bride Place of Residence	Helvoirt
Previous Wife First Name	Willemijn
Previous Wife Last Name	van den Hoef
Previous Husband First Name	-
Previous Husband Last Name	-
Witness First Name	-
Witness Last Name	-

Table 1.3: An example of data from a marriage certificate

For all records, some witnesses are also filled in.

To match those certificates together, we will need to apply entity resolution techniques. In chapter 2 we make a state of the art in which some of the most important are discussed. We will see that most state of the art entity resolution schema are based on machine learning: to function, they need to learn how to recognize a pair of matching records based on training examples. Such examples, usually drawn from human input or from the results of prior entity resolution projects, are not available in our case. To overcome this issue, we devise in chapter 3 a workaround entity resolution technique, based on a Bayesian statistical model, which uses probability distributions obtained in a data driven fashion instead of training data. In chapter 4 we discuss the implementation and the results of our data matching application. We will see that to overcome the lack of training examples, our technique needs the consistent availability of several certificate attributes. Results are encouraging nonetheless and in chapter 5 we conclude by summarizing our main results.

First Name	Arnoldus
Last Name	Giimans
Gender	-
Place of Death	Asten
Date of Death	-
Burial Death	1853-11-19

Table 1.4: An example of data from a death certificate

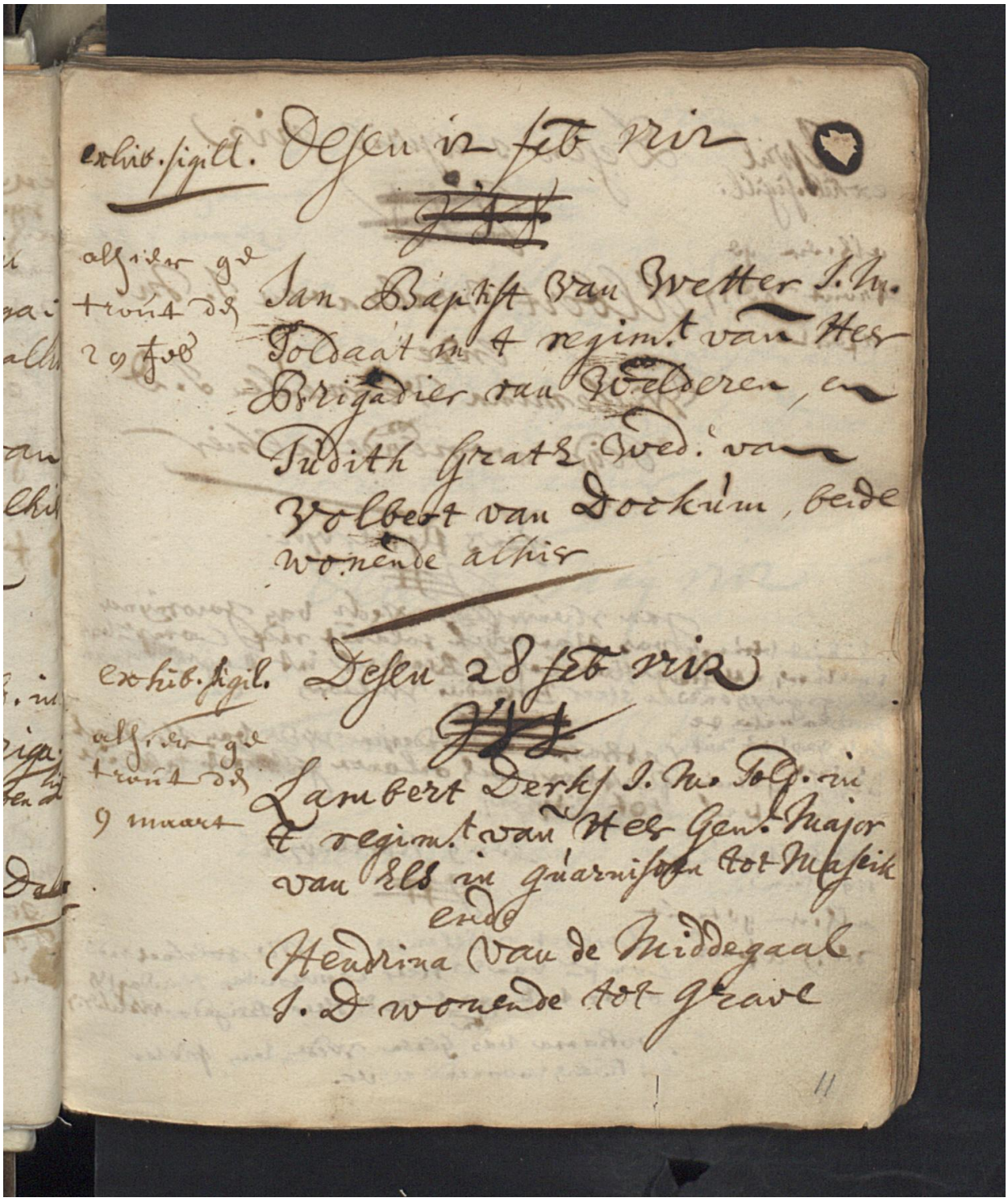


Figure 1.1: Picture of a death church certificate

Chapter 2

Data Matching: a State of the Art

This chapter is mainly based on Christen's book about Data matching, as it offers a comprehensive description of every step of entity resolution and several state of the art techniques [3]. It defines: "Data matching, also called record or data linkage, entity resolution, object identification or field matching is the task of identifying, matching and merging records corresponding to the same entities from several databases. When all records are from a single database, the task is called duplicate detection or deduplication." Figure 2.1, taken from Christen's book, shows the typical steps of a data matching process.

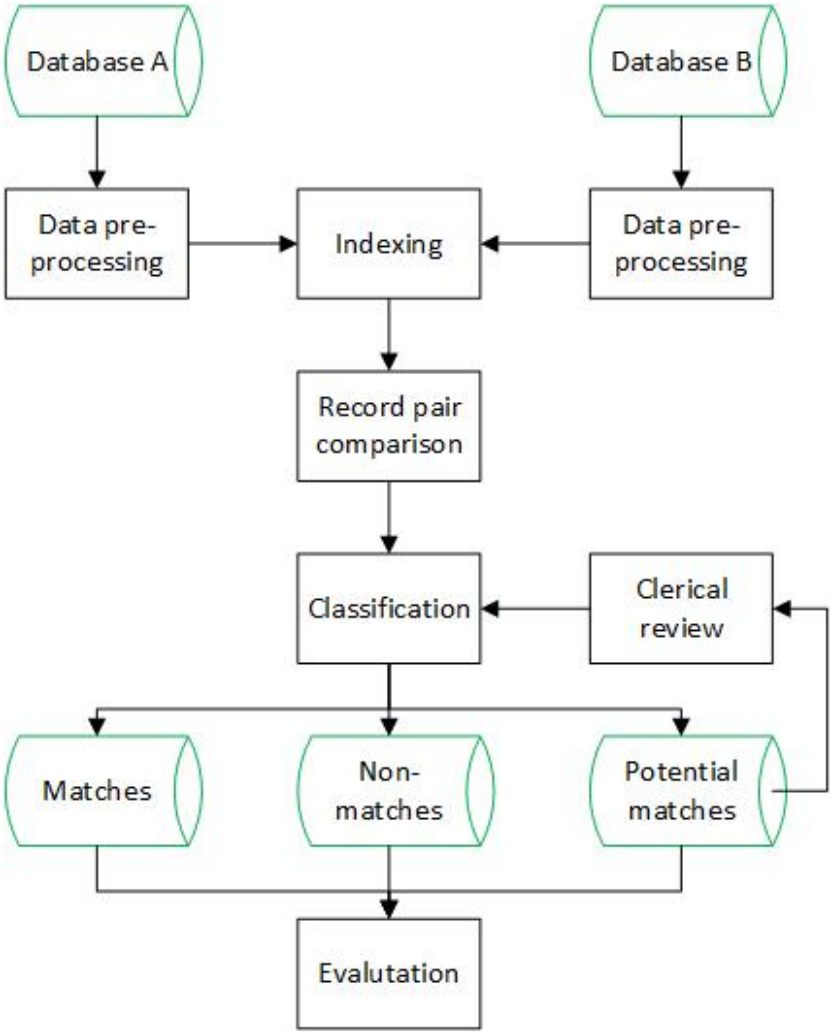


Figure 2.1: Data Matching Process Steps

Due to the lack of common entity identifiers in the databases to be matched, partially identifying attributes have to be used, such as names, addresses or dates of birth. Those data are however often of low quality due to typographical variations, errors, partial availability and change over time.
The poor quality of the data makes a first **cleaning** step necessary for most data matching procedures. During this step missing values are handled, noisy values are smoothed, inconsistent values are identified and corrected. The goal is to convert raw input data from the databases to be matched or deduplicated into a format that allows efficient and accurate matching.

Comparing two records will lead to the computation of similarity scores between several fields and takes some time. Moreover the comparison of two records will lead most of the time to non-matches. Also, the databases to be matched may simply be too big for a full comparison of every possible pair of records to be feasible. It is thus necessary to reduce as much as possible the amount of record pairs to be fully compared by removing the ones that unlikely correspond to true matches.
This second step is called **indexing** and will consist most of the time in dividing the data into groups according to some criteria so that a given record is only compared with the records from the same group, called candidate matches. In short: indexing filters out pairs which are very unlikely to match.

Once the indexing done, the **pairwise comparison** step can start. During this step a similarity score will be computed for each candidate pair to provide evidence of their similarity.

The next step is called "**classification**". During this step, the similarities computed during the comparison are used to assign candidate pairs to three sets:

- **Matches**. Matching pairs.
- **Potential pairs**. An optional set in which pairs for which it is unsure whether they are matching or not are placed. A human classification is usually required for the members of that set.
- **Non matches**. Non matching pairs.

Finally, an **evaluation** step can take place. This last step has to assess the quality of the data matching by computing several measures. The complexity of the algorithm but also more importantly accuracy measures such as precision and recall can be computed.
Figure 2.2 shows the expected result of data matching for the three certificates shown in tables 1.2, 1.3, and 1.4. Linking certificates, each containing fragmentary data about a person's life, gives us access to the whole picture. We discuss the usual steps of entity resolution in the following sections.

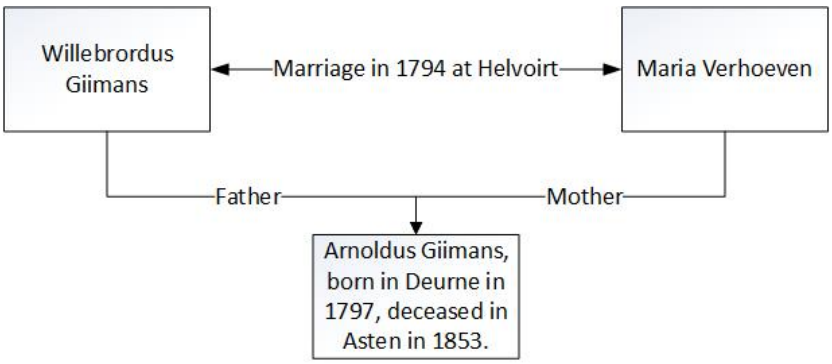


Figure 2.2: Output of data matching of a single birth, marriage and death certificate

2.1 Data Pre-processing

The goal of data pre-processing is to convert the raw input data from the database to be matched or deduplicated into a more standardized format allowing efficient and accurate matching.
The usual steps of data pre-processing are discussed below.

2.1.1 Remove Unwanted Characters and Words

This step corresponds to an initial cleaning. The goal is to remove individual characters, words, terms or abbreviations that do not contain information of use for data matching or deduplication.
Whenever a token that is to be removed or converted is found, it is treated using either hard-coded rules or look-up tables.

Lastly, during this first step, all letters may be converted into either lower or upper case, uniqueness of format (Unicode or ASCII) should be ensured and all multiple occurences of whitespace characters should be replaced by a single whitespace.

2.1.2 Standardization and Tokenization

This second step of data pre-processing is crucial to improve the quality of the data to be matched. It consists in correcting known errors or variations in attribute fields, replacing names or nicknames by their standard form, expanding abbreviations into their standard form. To do so, look-up tables containing standard place names or person's first and last names can be used[1]. Table 2.1 shows a sample of rows of such look-up table for men's first names: we see for instance that both Jorden and Gordanus are standardized to Jordanus.

Standard first name	Variations
Joosten	Joesten, Joestens, Joosten, Joostens
Jordanus	Gordanus, Jarden, Jodanus, Jordanes, Jordanis, Jordanmus, Jorde, Jurden, Jordanus, Jorden

Table 2.1: Men's first names look-up table

2.1.3 Segment Attributes into Well-defined and Consistent Output Attributes

This step deals with the common situation of database attributes that contain several pieces of information, such as the "Address" attribute. The goal is to make sure each output field contains a single piece of information. To do so, we need to parse input data, which is often semi-structured or in free-text format.

In our case this step will not be necessary since DTB's are structured data.

2.1.4 Verify the Correctness of Attribute Values

This last step consists in verifying the correctness of attribute values. This is done most of the time by using a reference database containing correct values. This step can, for example, be employed for addresses if an external database is available that contains all known and valid addresses in a country or region.

2.2 Record Pair Comparison

Comparison has to provide evidence of the similarity between each candidate pair found during indexing. This is done by computing one or several similarity scores for each pair. These scores can be based on names, places, dates, gender and so on. Thus, the output of the pairwise comparison step is a similarity vector for every candidate pair, in which each element corresponds to the similarity between a pair of features. Those vectors can be used as they are or aggregated into a single global similarity score. Such similarity measures allow later on to decide if whether or not two records refer to the same entity.

Because the data matched can be of low quality, it is important that the comparison functions return some indication of how similar two attribute values are, rather than a binary value. For instance, the

comparison of the names "Stefan Peter" and "Stephanus Petrus" should yield higher similarity than "Stefan Peter" and "Maria Janssen". In the following sections we briefly describe several comparison techniques.

2.2.1 Edit Distance String Comparison

Edit distance string comparison functions count the smallest number of edit operations required to convert one string into another.
The basic one, the **Levenshtein edit distance**, counts the smallest total number of single character insertions, deletions and substitutions, with no distinction between these three operations. The distance can be normalized to compute the Levenshtein similarity score between two strings s_1 and s_2, using the maximum of the lengths of the two strings:

$$sim_{levenshtein}(s_1, s_2) = 1 - \frac{dist_{levenshtein}(s_1, s_2)}{max(|s_1|, |s_2|)}$$

This similarity score is useful when a normalized similarity score for strings is needed. For instance, the Levenshtein distance between *Arnoldus* and *Aarnoltus* is 2, since the insertion of a letter *a* and the substitution of *d* to a *t* separate the two strings. The associated normalized Levenshtein similarity score is:

$$sim_{levenshtein}(Arnoldus, Aarnoltus) = 1 - \frac{2}{9} = 0.78$$

The basic Levenshtein edit distance dynamic programming algorithm has a $|s_1| \times |s_2|$ complexity[3].

2.2.2 Q-gram Based String Comparison

Q-gram based string comparison splits the two strings into sub-strings of length q (q-grams) using a sliding window and counts the q-grams occurring in both input strings. Its complexity is only $O(|s_1| + |s_2|)$[3]. The Jaccard coefficient is defined as:

$$sim_{jaccard}(s_1, s_2) = \frac{Q_1 \cap Q_2}{Q_1 \cup Q_2}$$

where Q_1 and Q_2 are the sets of q-grams of each string.
Using bigrams ($q = 2$) on our example of *Arnoldus* and *Aarnoltus*, we have (table 2.2):

String	Bigrams
Arnoldus	**ar**, **rn**, **no**, **ol**, ld, du, **us**
Aarnoltus	aa, **ar**, **rn**, **no**, **ol**, lt, tu, **us**

Table 2.2: An example of bigrams

$$sim_{q-gram}(s_1, s_2) = sim_{jaccard}(s_1, s_2) = \frac{5}{7 + 8 - 5} = 0.5$$

2.2.3 Jaro and Winkler String Comparison

The Jaro and Winkler string comparison functions combine edit distance and q-grams and were specifically designed for names comparison. The simple Jaro similarity between two strings s_1 and s_2 is[9]:

$$sim_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c - t}{c}\right)$$

where c is the number of common characters with positions within half the length of the longest string and t is the number of transpositions (adjacent characters that are swapped).

For instance, consider the strings s_1 = DIXON and s_2 = IDCKSONX, we have $|s_1| = 5$, $|s_2| = 8$. There is one transposition: DI to ID, therefore $t = 1$. Characters D, I, O, N, X are common and at distance respectively of 1, 1, 2, 2 and 5. X does not count as matching character since it's in position 3 in s_1 and 8 in s_2 which leads to a distance of $5 > \frac{max(|s_1|,|s_2|)}{2} = 4$. Therefore $c = 4$ (example taken from [18]). This yields a Jaro score of:

$$sim_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{4}{5} + \frac{4}{8} + \frac{4-1}{4}\right) = 0.683$$

Empirical studies have led to several modifications to the Jaro algorithm [3]. The basic Winkler similarity, shown below, gives more importance to the start of the strings. Indeed, experience has shown that in general less errors occur in the start of a word:

$$sim_{winkler}(s_1, s_2) = sim_{jaro}(s_1, s_2) + \left(1.0 - sim_{jaro}(s_1, s_2)\right)\frac{p}{10}$$

where $p \quad (0 \leq p \leq 4)$ is the length of the two strings' common prefix.

A second modification is applied for strings both at least 5 characters long. It is applied if the following three conditions are verified:

1. Both strings are at least 5 characters long:
$$min(|s_1|, |s_2|) \geq 5$$
2. There are at least two common characters (c) which are not in the prefix (p):
$$c - p \geq 2$$
3. Besides the common prefix both strings have at least half of the remaining characters of the shorter string in common.
$$c - p \geq \frac{min(|s_1|, |s_2|) - p}{2}$$

If all three conditions are met, the similarity is adjusted as follows:

$$sim_{winkler\_long}(s_1, s_2) = sim_{winkler}(s_1, s_2) + (1.0 - sim_{winkler}(s_1, s_2))\frac{c - (p+1)}{|s_1| + |s_2| - 2(p-1)}$$

It increases the similarity of long strings which have a lot of characters in common besides the prefix.
A third modification is to simply consider characters that can easily be substituted as similar: for instance "a" and "e", "w" and "v" or "5" and "s". It makes the similarity measure more forgiving with respect to spelling mistakes. This leads to a new definition of the basic Jaro similarity:

$$sim_{winkler\_sim}(s_1, s_2) = \frac{1}{3}\left(\frac{(c + c_{sim})}{|s_1|} + \frac{(c + c_{sim})}{|s_2|} + \frac{c - t}{c}\right)$$

This change in the definition of c can be passed to the two previously shown modifications. The complexity of Jaro and Jaro-Winkler algorithms is $O(|s_1| + |s_2|)$ [3].

2.2.4 Date comparison

For date comparison, it is common to calculate the difference between them, sometimes in an absolute fashion. Once the time interval between two records is available, it has to hint the (di)similarity of the records at hand. This can be done based on domain knowledge: we expect for instance a birth certificate and the associated death certificate to be separated by no more than 100 years. Impact on the similarity can also be drawn from statistics learned from the data, for instance the figure 3.2c from subsection 3.3.3 shows that in most cases 2 years separate birth certificates in which parents are the same. It also shows that a year difference above 11 tends to hint that two birth records do not belong to the same parents.

2.3 Indexing

Potentially, each record from one data source must be compared with every records from the other one: the complexity of such process would be $O(n \times m)$, where n and m are the sizes of both sets.
Therefore, before undergoing the comparison of records, whose computation complexity can also be $O(|s_1| \times |s_2|)$ where $|s_1|$ and $|s_2|$ are the lengths of the compared strings (see subsection 2.2.1), it is necessary to reduce the number of record pairs to be compared, by preselecting only those that possibly correspond to true matches. This filtering step is called indexing. Several indexing techniques are described in the following subsections.

2.3.1 Blocking

This is the simplest indexing technique: the records are clustered in mutually exclusive blocks to be matched, according to some criteria called "**blocking key**". Only records with the same blocking key are compared to each other. The final quality of the matching process depends on the quality of the definition of the blocking key since it determines which records are compared together. Obviously there is a trade-off between quality and cost of processing: enlarging the size of the blocks of records increases the completeness of the data matching, at the expense of complexity.

To alleviate the results of misspelling of names, **phonetic encoding** is often applied: records yielding the same phonetic code are placed in the same block. Phonetic encoding, however, is mostly language dependent. **Soundex** is one of the oldest and most widely used encoding algorithms. It encodes names by keeping the first letter and converting the remaining characters into numbers according to a transformation table (figure 2.3) based on English pronunciation and removing vowels (unless it's the first letter) and repetitions and capping the result.
Its major drawbacks are that a different first letter results in different codes and that differences at the end of the names are pruned if the codes are too long. One solution is to generate also codes of the reversed name strings [3].
An error in an attribute value used as a blocking key may lead to insert the record into another block; in

a, *e*, *h*, *i*, *o*, *u*, *w*, *y*	→ 0
b, *f*, *p*, *v*	→ 1
c, *g*, *j*, *k*, *q*, *s*, *x*, *z*	→ 2
d, *t*	→ 3
l	→ 4
m, *n*	→ 5
r	→ 6

Figure 2.3: Soundex encoding transformation table

particular, for this project, the omission of a field such as the first name (7% of the death certificates: see section 4.2) discards the record from the right block. A way to overcome that is to use different blocking keys, based on different attributes, and to merge all candidate record pairs.
An inherent issue of blocking is the coverage problem [7]: similar records are clustered into different blocks because the blocking key is too discriminative. This issue can be alleviated though by using overlapping blocks.

As example, we illustrate shortly one of the simplest blocking techniques: it uses the first letter of first and last names of records' stakeholders as blocking key. It is based on the ascertainment that in general no mistakes are made in a name's first letters. Consider the four names displayed in table 2.3. They lead to the two blocking keys **EP** and **LP**. It seems clear that Elizabeth Petri, Lisbeth Petrus and Liz Peter belong together as they are very similar names and may all refer to the same person. Likewise, Egidius Pooter and Eegidii Potter belong together. Tables 2.4 and 2.5 show the resulting blocks. Most of the records worth comparing end up in the same block, as usually similar names start with the same letter. However, Elizabeth Petri and Lisbeth Petrus end up in different blocks, this is an illustration of the coverage problem inherent to blocking algorithms.

Elizabeth Petri
Lisbeth Petrus
Egidius Pooter
Liz Peter
Eegidii Potter

Table 2.3: A bag of names

Elizabeth Petri
Egidius Pooter
Eegidii Potter

Table 2.4: Block with key **EP**

Lisbeth Petrus Liz Peter

Table 2.5: Block with key **LP**

2.3.2 Locality Sensitive Hashing

This section is based on [10]. The basic idea behind locality-sensitive hashing is to use a hash function that will yield the same result (called bucket) for similar records. We want similar strings to be hashed to the same bucket and dissimilar strings to be hashed to different buckets. Full pairwise comparison will then be performed on strings within the same bucket.

The indexing technique and the similarity measure used during the pairwise comparison have to be aligned, the goals being that records from the same bucket are more likely to yield high similarity and all record pairs worth comparing are end up in the same bucket. For instance, an indexing technique tailored to the use of Levenshtein edit distance may not be adapted to Jaro comparison, as discussed in section 3.2 and Appendix I. Therefore, the choice of hashing function will depend on the similarity measure between records, which itself of course depends on the type of document to be matched.

Locality sensitive hashing variants are described for the Jaccard distance, Hamming distance, cosine distance and for the normal Euclidean distance in [10]. In our case, records are mainly described by stakeholders' names. Those strings can easily be translated to a list of numbers by counting the occurrences of q-grams (see subsection 2.2.2) in those strings. Those numbers describe a vector or a dot in a n-dimensional space, where n is the number of different q-grams. Therefore, hashing functions based on the cosine distance (a distance based on the angle between vectors) or on the Euclidean distance can be used to compare them. As an example of such function, we describe below a family **F** of locality-sensitive hashing functions based on the Euclidean distance.

For the sake of clarity we show a 2-dimensional Euclidean space. Each function f in our family **F** will project all points on a single random line in the space. The line is then divided into segments each having the same constant length a. Points projected on the same segment are considered as candidate pairs (recall that each point represents a record). Figure 2.4, taken from [10], shows the randomly chosen line (oriented to be horizontal) as well as two records at an Euclidean distance d from each other.

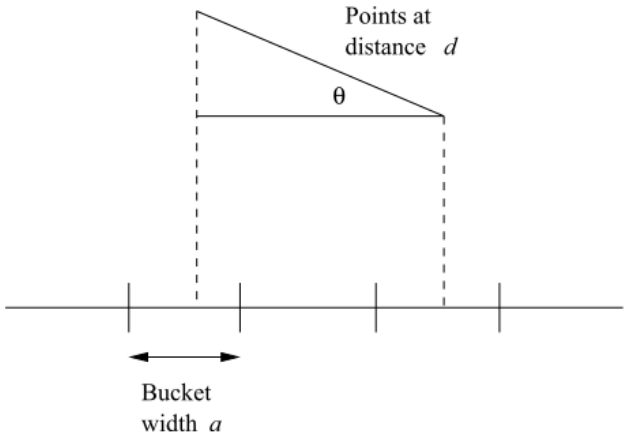


Figure 2.4: Two points at distance $d \gg a$ have a small chance of being hashed to the same bucket

The buckets obtained using this technique behave like the blocks used in blocking indexing. Using several projection functions instead of one, each one with its randomly chosen vector, allows to build overlapping buckets: the same record will end up in as many buckets as there are projection functions in **F**. This reduces the risk of similar records not being compared because stored in different buckets.

2.3.3 Levenshtein Automata Indexing

This section is based on [8] and [13]. Levenshtein automata indexing was used in [13] to match together users from Facebook and users from VKontakte, the most popular social network with Russian-speaking users.

Automata can be used as indexing tool: if we want to match two sets of records A and B, we can define L_d as being the language of all records within an edit distance d from any record in A. Records from B can be submitted to the automaton built on L_d. For each of the accepted records we can fetch candidate pairs from the automaton. The complexity of the automata indexing is in $O(n)$ where n is the length of the string being tested. This technique allows to avoid the coverage issue inherent to blocking techniques.

2.4 Classification

Once the pairwise comparison phase completed, a similarity vector (each element in the vector corresponds to the similarity score of an attribute like date, place or name...) is available for each candidate pair and we need to decide based on that vector if whether or not the pair is a match. This process is called classification. Candidate pairs are to be classified into two classes according to their similarities: matches and non-matches. A third class, called "potential matches", can be used if there are many records for which automatic classification is not sufficient and clerical review is required.

The availability of ground truth data giving a set of true matches and true non-matches allows to devise supervised classification ([11]) approaches based on machine learning: an algorithm learns the parameters of the classification. However, gathering such data is often a tedious and costly manual process [3]:

- Without indexing, the matching of two tables that contain m and n records will generate $m \times n$ candidate pairs to classify; without duplicates, a maximum of $min(m, n)$ matches will occur: the sizes of the match and non-match classes are very imbalanced. Thus, the probability that two randomly picked records are matching is very low, lesser than $\frac{min(m,n)}{m \times n}$. Therefore, finding serious candidate pairs requires the availability of a tool allowing to search and browse effectively trough records.
- Assessing a record pair may require to explore a lot of other similar records and/or to access external data sources.
- The results may differ from reviewer to reviewer and depend on the mood and concentration level of the reviewer.

If no ground truth data is available, unsupervised classification approaches such as threshold-based or rule-based classification are to be used: thresholds and rules can be set based on domain knowledge.

In the following subsections we briefly report upon several classification techniques.

2.4.1 Threshold-Based Classification

This is the simplest classification technique: all elements from the similarity vector are combined with a fixed formula and the resulting global similarity score is compared to a threshold (or two if potential matches are used), which can be set manually or learned based on training data:

- If we only use two classes (matches and non-matches), a single threshold value is sufficient. For a record pair (r_i, r_j) and a threshold t ([3]):

$$SimSum[r_i, r_j] \geq t \Rightarrow (r_i, r_j) \leftarrow Match$$

$$SimSum[r_i, r_j] < t \Rightarrow (r_i, r_j) \leftarrow NonMatch$$

- If we use three classes (matches, potential matches and non-matches), this becomes:

$$SimSum[r_i, r_j] \geq t_{upper} \Rightarrow (r_i, r_j) \leftarrow Match$$

$$t_{lower} < SimSum[r_i, r_j] < t_{upper} \Rightarrow (r_i, r_j) \leftarrow PotentialMatch$$

$$SimSum[r_i, r_j] \leq t_{lower} \Rightarrow (r_i, r_j) \leftarrow NonMatch$$

Of course this technique, as simple as it is, is very crude: the same threshold is used for all pairs and the quality of the matching is highly dependent on its value, as a very high threshold may filter out too many matching pairs while a very low threshold may accept too many non-matching pairs. Also, by combining the scores of all attributes, the information contained in individual scores is lost.

2.4.2 Probabilistic Classification

If the comparison is made based on multiple attributes, special attention should be devoted to the calculation of the weights assigned to the similarity values of the different attributes. Indeed, some attributes may have less discriminative power than others. The place of living for instance has less power than the first name when matching people together (many people live in the same place, and if the dates of two records are not the same, chances are that the person has moved). Also, some attribute values have less discriminative power than others, usually according to their popularity: for example, lesser weight should be associated to a high similarity on first names for "Maria" than for "Lutgarda": the probability that two randomly picked records have the same first name "Maria" is much higher than the probability that they both have the first name "Lutgarda", as "Maria" is one of the most popular first names.

In practice, for each attributes configuration, the probabilities that this configuration corresponds to a match and that this configuration corresponds to a non-match has to be computed. Thus, when comparing two records **a** and **b** with attributes configurations A and B, the global similarity score is a ratio of conditional probabilities, which are usually learned based on a set of labeled data:

$$SimScore[a,b] = \frac{P\left(A,B|\left(a,b\right) \in Match\right)}{P\left(A,B|\left(a,b\right) \in NonMatch\right)}$$

This is a more holistic way of computing the score: the resulting score is not only function of the absolute similarity between attributes, it is also function of the discriminative power of attributes and their values. The decision is finally made by comparing that score to a threshold, in a similar fashion than described in previous subsection.

2.4.3 Cost-Based Classification

There are two kinds of errors we can run into when performing classification: classifying matching records in the non-match category (false negatives) and classifying non-matching records in the match category (false positives). Each threshold value yields a number of false positives $|FP|$ and false negatives $|FN|$. Thresholds are usually set so that the overall number of misclassified record pairs $|FP| + |FN|$ is minimized. However, the cost of a false negative and the cost of a false positive may not be the same depending on the application.

For instance consider the application of matching together two databases containing data about the clients of an online store. The goal is to build a better profile for each client so that adds can better be targeted. False negatives lead to clients not receiving a targeted add, which translates into potential loss of profit, while false positives lead to clients receiving adds for products they are not interested in. In that case, the cost of a false negative is much higher than the cost of a false positive.

Now consider that instead of clients of an online store, the databases contain patients of different hospitals. In that case, false negatives lead to incomplete profile for some patients, who may not receive proper medication. False positives yield incorrect patient profiles, leading to a dangerous situation in which patients may receive unneeded medication. In that case, the cost of a false positive is much higher ([3]).

In our case, a false positive will make an entire part of a family tree incorrect while a false negative will erase an entire part of the tree.
If the class *Match* is a list of supposed matches that have to be validated by domain experts, the cost of false negatives is higher: if a pair of true positives is not shown to the expert, it will never be found. On the other hand, if the purpose of the application is to have a completely automatized entity resolution schema, then false positives have higher cost as they make the whole resulting genealogical trees incorrect.

The basic idea behind cost-based classification is to set, based on domain knowledge, the cost of a false negative C_{FN} and the cost of a false positive C_{FP} and to learn the classifier so that the overall classification cost is minimized:

$$TotalCost = C_{FP}|FP| + C_{FN}|FN|$$

where $|FP|$ and $|FN|$ are the number of false positives and false negatives yielded by the classification. This technique allows to better adapt the classification to the application at hand, by giving better control over the proportion of false positives and false negatives returned by the classification step.

2.4.4 Rule-Based Classification

During rule-based classification, candidate pairs are evaluated against a set of rules, testing combinations of similarity values, and classified accordingly. To function properly, a rule-based classifier must achieve complete coverage: all possible attributes configurations available to a pair of records must be covered by rules. There are two ways to build a rule-based classifier:

- The **supervised approach**: rules are learned based on training data. The quality of the classifier depends on the quality of the training data, as they must cover all possible cases.
- If no sufficient training data is available, the rules can be handwritten based on domain knowledge. This is the **unsupervised approach**. However, to devise by hand a set of rules achieving complete coverage is a tedious task, prone to error.

If complete coverage cannot be attained, a default classification outcome must be set. A rule usually has the form $P \Rightarrow C$ where C is the outcome of he classification and P is a predicate applied on similarity values [12]:

$$P = (term_{1,1} \vee term_{1,2} \vee ...) \wedge ... \wedge (term_{n,1} \vee term_{n,2} \vee ...)$$

Each term tests the value of a similarity feature, as for example: $Date(r_1) \leq Date(r_2)$. For instance, a very simplistic rule set would classify a person X and a person Y with similar first and last names or same father and similar first names or same mother and similar first names as being the same, with $NonMatch(X, Y)$ as default outcome [5]:

$$SimilarFirstName(X, Y) \wedge SimilarLastName(X, Y) \Rightarrow Match(X, Y)$$

$$areEqual(Father(X), Father(Y)) \wedge SimilarFirstName(X, Y) \Rightarrow Match(X, Y)$$

$$areEqual(Mother(X), Mother(Y)) \wedge SimilarFirstName(X, Y) \Rightarrow Match(X, Y)$$

This technique effectively allows the classification outcome to respect patterns learned from training data or/and human domain knowledge.

2.4.5 Active Learning Approaches

Supervised classification techniques require a significantly large training set, which is in many practical cases very difficult to obtain. Indeed, to be relevant, training data must be as unbiased as possible and will often have to be collected by hand, which requires effective browsing tools and manpower. On the other hand, unsupervised techniques rely heavily on human domain knowledge, which can be lacking.

To make up for the lack of high quality training data, several classification techniques have been built on techniques of active learning [3]. They initially only require a small training set based on which a first crude classification model is learned. This model is then enriched on the go by interactively collecting new training data. Those are obtained by regularly asking for user input. This leads to an iterative process, repeated until satisfying matching quality is achieved:

1. A classification model is trained based on available training data.
2. Candidate pairs are classified using that model.
3. The pairs that were most difficult to classify are submitted to a user for manual classification.
4. Those manually labeled pairs are added to the training set and the whole process is repeated.

This technique is useful when complete training data is not available and unsupervised techniques are not sufficient.

2.4.6 Collective Classification

Previously discussed techniques are said to be "local": they decide whether or not a given candidate pair of records is matching only based on the pair's attributes. This can often be sub-optimal as those techniques don't take the characteristics of all other candidate pairs into account.

Collective classification techniques work in a global fashion, they are often graph-based approaches [3]: a relationship graph can be built from a set of already known matches and potential matches. Records are nodes in the graph. A link between two nodes represents a relationship between the records. The weight of a link is initially set using the similarity score between the two records. In our case the links are family relationships such as husband, mother or brother. If two nodes are known without doubt as being matching, the link between them is strong. If they are potential matches, the link is soft. The whole idea is to iteratively recompute the weights of those ambiguous links based on the strong connections in the graph until a stopping criteria is met. They can then be classified as match and become strong or be classified as non-match and be erased. This global approach allows to find the best match for each record, instead of finding the first satisfying match.

2.5 Evaluation

The purpose of this last step is to assess the quality of the entity resolution schema. Usually, to do so, the record pairs previously classified are reviewed and assigned to one of the following categories:

- **True positives (TP):** Record pairs classified as matches and that are indeed true matches (they refer to the same entity).
- **False positives (FP):** Record pairs classified as matches but they are not true matches (they don't refer to the same entity).
- **True negatives (TN):** Record pairs classified as non-matches and that are indeed true non-matches.
- **False negatives (FN):** Record pairs classified as non-matches but they are true matches.

Table 2.6, taken from [3], shows the different outcomes of a data matching classification.

	Predicted matches	Predicted non-matches
Actual matches	True Positives (true matches)	False Negatives (false non-matches)
Actual non-matches	False Positives (false matches)	True Negatives (true non-matches)

Table 2.6: Outcomes of a data matching classification

A good entity resolution schema will maximize the number of true positives while minimizing both the number of false positives and false negatives. Hence, there are two main sizes to measure the quality of an entity resolution process:

- **Precision** is the proportion of matches correctly classified:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall** is the proportion of true matches correctly classified:

$$Recall = \frac{TP}{TP + FN}$$

Obviously **there is a trade-off between precision and recall** as a very lenient classifier will achieve high recall and low precision while a very restrictive classifier will achieve high precision and low recall. Thus, the entity resolution schema must be built in such a way that precision and recall are jointly maximized. A popular tool to do so are the precision-recall curves: each point on the curve represents precision and recall for a given matching setting. Figure 2.5, taken from [3], shows the typical shape of such PR curve.

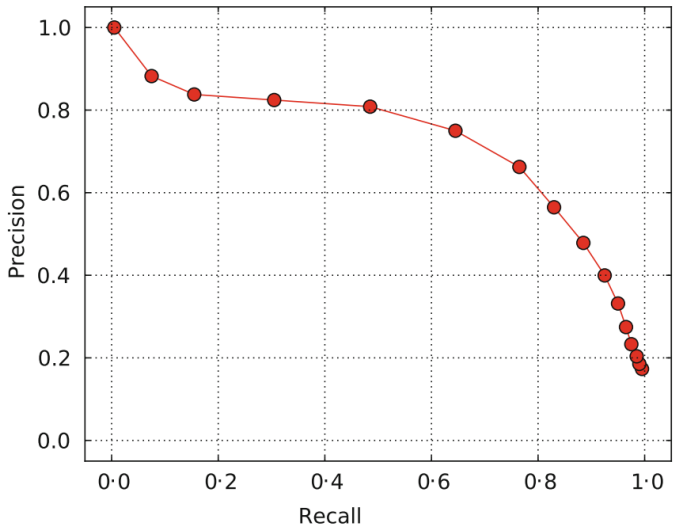


Figure 2.5: Example of Precision-Recall curve

To compute these measures, labeled data are required: in general, precision and recall yielded by an entity resolution schema are computed by applying it to a previously labeled set of records. By doing so it is possible to recognize true positives, false positives, true negatives and false negatives.

During evaluation, the computational speed of the entity resolution schema can also be assessed by computing the complexity of each step and by running the schema on a benchmark machine.

Chapter 3

Our Entity Resolution Schema

In this chapter, we devise an entity resolution schema that will allow us to match our certificates together. We saw in section 2.4 that most classification techniques are based on machine learning and require a training set. The evaluation step also requires training data to compute precision and recall, as discussed in section 2.5. However, such training set is not readily available for our data set and collecting it by ourselves is not in the scope of this project. Indeed, the best way to obtain an as unbiased as possible gold standard is to resort to manual labeling, which is a tedious and time intensive task as it requires users to find matching pairs by searching through the data. To do so, an interface allowing to browse, search and label records from our data set has to be implemented.

To overcome the lack of training data, we decided to devise workaround techniques for classification and evaluation. In the following sections we discuss the order in which certificate types should be matched together and we outline the techniques used in this work for indexing, pairwise comparison, classification and evaluation.

3.1 Devising a Global Matching Strategy

Before trying to match data together, we first discuss what data to match and in what order. We have birth, marriage and death certificates with data structure and availability discussed in sections 1.1 and 4.2. Different types of actors are consistently available:

- **Newborn** children from birth certificates.
- **Newlywed** brides and grooms from marriage certificates.
- **Parent couples** fathers and mothers from birth certificates.
- **Deceased people** from death certificates.

Information is scarce, by matching records together we build an entity (a person) and we gain information about it. As there are four main types of actors, four matchings should be enough to aggregate those four actors into one person. The first matching will give useful data for further ones as information is gained at each step. Also, in general, priority should be given to couples matching: they are more precise since two pairs of names are available instead of one. To describe our matching strategy, we give below the sequence of four matchings we intent to perform:

1. **Parents to Parents matching**: will allow us to deduplicate parents. We will have a single entity for each couple and know when and where the couple had children as well as the names of the children.
2. **Parents to Newlyweds matching**: we will know when and where a parents couple got married. Also, 25% of marriage records contain place of birth of bride and groom. This can help us matching our entity with its birth record (as the entity's birthplace given in his or her marriage certificate should be the same as the place recorded in his or her birth certificate).
3. **Newborns to Parents matching**: find a newborn's children. This matching should allow to build a family tree. However, it is single person to single person matching: less features are available (one pair of person names instead of two).

4. **Newborns to Deceased people matching**: the last piece of information we need is date and place of death. It is a single person to single person matching as well.

In the next sections we describe our entity resolution schema, starting with the indexing technique.

3.2 Indexing: Bit Vector Tree Indexing for Levenshtein Distance

Our indexing technique is based on Marijn Schraagen's work [14] [15]. Levenshtein edit distance is described in subsection 2.2.1. The bit vectors indexing technique presented in this section addresses the coverage problem inherent to standard blocking techniques (see subsection 2.3.1) while offering practical computation time bounds by computing all pairs of records within a threshold of the Levenshtein edit distance. For each record, a bit vector is built. A position in the vector represents the presence or absence of certain characters of the alphabet. Order and frequency are discarded. The resulting bit vector is then used to store the record in a leaf of a binary tree. When looking for candidate matches for a given record, the associated bit vector is computed and a tree traversal is done to retrieve the candidate matches.

Records are mainly characterized by strings: first names and last names of people involved in an event. A bit vector can be constructed from each string, where each position in the vector represents the presence or absence of a letter. For instance, given an alphabet *{a, b, c}* and a record *cacca*, mapping characters to vector positions in alphabetical order would result in the vector [1, 0, 1] (example from [15]). Bit vectors can be used to construct a binary tree in the leaves which records are stored. For each record the corresponding bit vector represents a path in the tree to the leaf node containing the record. The maximum amount of nodes in the tree is bounded by $d * n$ where d is the depth of the tree and n the number of records (this corresponds to the worse case scenario in which each record would lead to the creation of a separate path in the tree). Shared paths make this number much lower in practice.
Figure 3.1, taken from [15] shows a simple vector tree built on the alphabet *{a, b, c}* for several records.

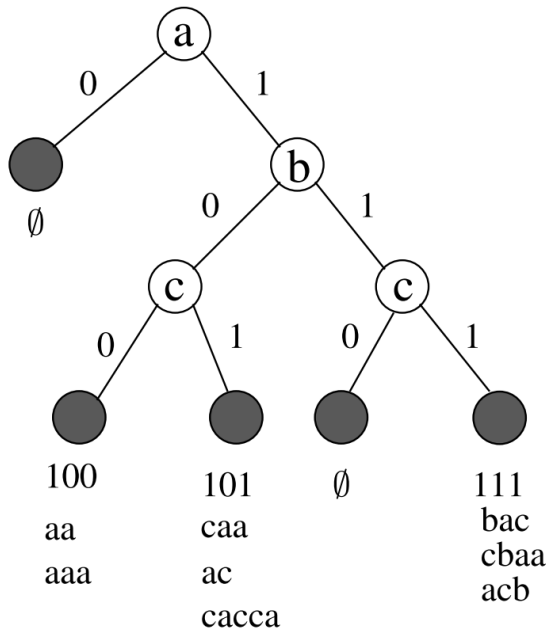


Figure 3.1: Example Vector Tree

Algorithm 1, taken from [15] shows the construction of the tree. A bit vector is built from each target record. The bit vector describes the path from root node to the leaf where the target record would be stored. We first check if there is an edge labeled with the value of the first bit from the root node to one of its child nodes. If there is, we follow it and do the same for the second bit and the child node. If there is not, we create a child node with an edge labeled with the value of the bit and follow the newly created edge. Once we get to the last bit of the vector, the traversal is over: we reached a leaf and we are to add the record to the current node.

Algorithm 1 Tree Construction on Target Records

for all *target records rec* **do**
  $[b_0, b_1..., b_n] \leftarrow BitVector(rec)$
  current node cur $\leftarrow$ *rootNode*
  for all bits b_i **do**
    if $EdgeExists\,(cur, b_i, next)$ **then**
      $cur \leftarrow next$
    else
      $AddEdge\,(cur, b_i, next_{new})$
      $cur \leftarrow next_{new}$
    end if
    if $LastBit(b_i)$ **then**
      $AddRecord(cur, rec)$
    end if
  end for
end for

When looking for potential matches, the tree is traversed for every source record using the recursive algorithm 2, taken from [15]. A bit vector is constructed for the source record and the tree is traversed, starting at the root node, by following the edges corresponding to the bit values of the vector. If the path exists, records stored in the leaf node are added to candidate matches. A limited amount of incorrect branch traversals is allowed so that approximate matches can be found. There are three cases:

1. We reached the last bit of the vector: we are in a leaf and need to return all records stored in it.
2. We did not reach the last bit of the vector and current error is below max error: we explore the incorrect path and we increment error and bit counter.
3. We did not reach the last bit of the vector: we explore the correct path and we increment the bit counter.

The traversal complexity is in $\binom{k}{e}$ with k the vector length (= the depth of the tree) and e the maximum permitted error.

We show that this indexing technique indeed returns records worth comparing: let r_1 and r_2 be two records, yielding respectively the bit vectors V_1 and V_2. Let e_1 be the number of vector positions with value 1 in V_1 and value 0 in V_2 and e_2 is the number of vector positions with value 1 in V_2 and value 0 in V_1. In other words e_1 is related to the number of letters which are in r_1 but not in r_2 and reciprocally. Thus, to transform r_1 into r_2, at least e_1 deletions and e_2 insertions are needed: $max(e_1, e_2)$ is a lower bound for the Levenshtein distance (described in subsection 2.2.1) between r_1 and r_2. $max(e_1, e_2) = error_{max}$ can be used as maximum allowed error during tree traversal, as shown in algorithm 2. By doing so we are assured that all leaves we don't visit only contain records which are at an edit distance of at least $error_{max}$ from the source record at hand. We just showed that records stored in far away leafs have high Levenshtein distance.

Algorithm 2 TreeTraversal(Node cur, bitVector V, int pos, int e_1, int e_2)

candidate set $C \leftarrow \oslash$
if *pos* = *last position* **then**
  return *records in node cur*
end if
$b \leftarrow V_{pos}$
incorrect node $next_{err} \leftarrow TreePath\,(cur, 1 - b)$
$e\prime_1 \leftarrow e_1 + b$
$e\prime_2 \leftarrow e_2 + (1 - b)$
if $e\prime_1 \leq error_{max}$ **and** $e\prime_2 \leq error_{max}$ **then**
  $C \leftarrow C \cup TreeTraversal\,(next_{err}, V, pos + 1, e\prime_1, e\prime_2)$
end if
correct node $next_{corr} \leftarrow TreePath\,(cur, b)$
$C \leftarrow C \cup TreeTraversal\,(next_{corr}, V, pos + 1, e_1, e_2)$
return C

Two parameters are also discussed in [15]:

- **Subvectors per record:** Since a bit vector is independent of characters position and characters frequency, many highly dissimilar records can be represented by the same vector. To alleviate this issue, a subvector can be computed on every string characterizing the record at hand. In our case a record contains the first and last names of a number of stakeholders (newborn, parents, bride and groom, deceased person). We thus build a bit vector for each of those strings then concatenate them to get the record's bits vector. By doing so, we increase the overall similarity of returned candidate matches.
- **Characters per vector positions assignment:** If multiple vectors are used to describe one record, and if every character in the alphabet is associated to a single vector position, we get longer vectors and thus larger trees and less traversal performances. To alleviate this, several characters are assigned to a single vector position. By doing so, we increase back the vector overlap for distinct strings. Nevertheless, an optimal letter vector position assignment can be found to make the disadvantage of overlapping vectors small compared to the efficiency gain of smaller vectors. Also, the assignment of letters to vector positions controls the distribution of records in the tree. Thus, assignment has a huge impact on the performance of the algorithm. However, finding an optimal assignment is difficult due to the complex interplay between assignment, tree size and tree traversal efficiency. Marijn Schraagen tested several character distributions in [15] and found an optimum using simulated annealing. We used that optimal character distribution in our implementation.

This technique's biggest limitation is inherent to its associated similarity measure, Levenshtein edit distance. Indeed, Levenshtein similarity does not yield satisfying results when it comes to synonym names. For instance the two synonym names "Stefan" and "Stephanus" yield a fairly low Levenshtein similarity of 0.555 compared to their basic Jaro similarity (see subsection 2.2.3) of 0.796. This issue can be alleviated though by using a synonyms dictionary to standardize names: for instance the names "Stefan" and "Stephanus" would be both standardized to "Steven".

3.3 Similarity Score Computation and Candidate Pairs Classification: Introducing a Pragmatic Approach Based on Bayes Theory

This section is mainly based on [16] (chap 5). The intended contribution of this work is to outline an effective data matching technique which does not require any kind of gold standard. Most state of the art matching techniques use a training set of some sort. In our case getting such gold standard would require human people to search and find by hand matches for a subset of the certificates. The high amount of stored certificates and the low amount of attributes available does not allow to do such matching by hand. In this section we describe a workaround matching procedure which yields good results without any need for training data.

3.3.1 Naive Bayesian Classifier

Our approach is based on the naive Bayesian classifier described in [16], which is an application of Bayes theorem:

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)}$$

where X and Y are random variables.
Let X denote a pair's attributes set: edit distance between stakeholders' names, delta years, geographical distance between two certificates, and let Y be the variable that represents the outcome of the classification phase (either match or non-match). $P(Y|X)$ is the probability distribution for a pair of records to be classified as match or non-match given a similarity configuration X. We need to learn it for every combination of X and Y so that any pair with similarity X' can be classified by finding the Y' maximizing it.

In a naive Bayes classifier we make the assumption that the attributes X_i are conditionally independent, which translates into:

$$P(X|Y = y) = \prod_{i=1}^{d} P(X_i|Y = y)$$

where each attribute set X consists of d attributes.
Two attributes are conditionally independent if, once every other attribute fixed, they are functionally independent. We will show empirically in section 4.4 that this hypothesis of conditional independency holds. To classify a record, the naive Bayes classifier computes:

$$P(Y|X) = \frac{P(Y)\prod_{i=1}^{d} P(X_i|Y)}{P(X)}$$

In standard approaches, the probabilities used in this formula are learned based on a training set. **However, adaptations have to be made in our case since such training data is not available**. In the next subsections we find a similar score function which minimizes the number of attributes we need to learn. We discuss then the learning of the conditional probabilities necessary to compute it.

3.3.2 Devising a Similarity Score Function

A record will be classified as $Match$ or $NonMatch$ based on a comparison of $P(Y = Match|X)$ and $P(Y = NonMatch|X)$. We have:

$$\alpha = P(Match|X) = \frac{P(X|Match)\,P(Match)}{P(X)}$$

$$\beta = P(NonMatch|X) = \frac{P(X|NonMatch)\,P(NonMatch)}{P(X)}$$

The simplest, most intuitive score function would be to use the probability α as a score: it is between 0 and 1 and easy to interpret. However, computing α would require us to know $P(X)$ and $P(Match)$:

- $P(X)$ is the probability distribution of a pair's attributes set. For any configuration $X_1 = x_1, X_2 = x2, ...X_d = x_d$ it would require us to compute the absolute probability $P(X_1 = x_1, X_2 = x_2, ...X_d = x_d)$ which is not practicable.
- $P(Match)$ is the absolute probability, when randomly picking a pair of records, that those two records are matching. If we denote $|Match|$ the number of matching pairs and N the total number of possible record pairs we have $P(Match) = \frac{|Match|}{N}$. N can be computed easily: it is the product of the sizes of the two sets we are matching. However, $|Match|$ is tricky to compute: without any training data available we can, at best, estimate it based on expected number of matching pairs. For instance when matching birth records with death records we expect $|Match| = min(|birth|, |death|)$ (where $|birth|$ and $|death|$ are the number of available birth and death records) since we only die once, there is a one to one relationship between birth and death records. In other cases though, such expected amount of matches is not available: when matching parents from birth certificates together to discover siblings it is much more difficult to set an expected amount of matching pairs. How many children did a couple have in average in North Brabant between the years 1580 and 1811 ?

Devising a score function that does not require $P(X)$ nor $P(Match)$ while still allowing to compare α and β would be handy. First we show that $P(X)$ does not need to be computed. This is easy to accept intuitively since it is a constant which appears in both α and β. Let's denote A and B the numerators, the hypothesis of conditional independency between attributes (cf Naive Bayes) allows to write:

$$A = P(Match)\prod_{i=1}^{d} P(X_i|Match)$$

$$B = P(NonMatch)\prod_{i=1}^{d} P(X_i|NonMatch)$$

$P(X)$ does not have to be computed, indeed: $\alpha + \beta = 1$ is equivalent to $A + B = P(X)$ so we can write $\alpha = \frac{A}{P(X)} = \frac{A}{A+B}$ therefore α and β can be computed without knowing $P(X)$. Let's denote a and b the conditional parts of A and B:

$$a = \prod_{i=1}^{d} P(X_i|Match)$$

$$b = \prod_{i=1}^{d} P\left(X_i | NonMatch\right)$$

A simple, easy to interpret score function is $\frac{\alpha}{\beta}$: if this quotient is larger than 1, then $P(Match|X)$ is larger than $P(NonMatch|X)$. To classify a candidate pair as match or non match, this score has to be compared with a threshold θ:

$$\frac{\alpha}{\beta} \geq \theta \Leftrightarrow \frac{A}{B} \geq \theta \Leftrightarrow \frac{a}{b} \geq \theta \times \frac{P(NonMatch)}{P(Match)} = \theta'$$

Since $\frac{P(NonMatch)}{P(Match)}$ is a constant, we can use $\frac{a}{b}$ as similarity score. The computation of the threshold θ' is empirical: it must be set in such a way that both precision and recall (see section 2.5) are maximized. In the next subsection we outline the calculation of the conditional probabilities used in the computation of the similarity score $\frac{a}{b}$ for a parents to parents matching (parent couples from different birth certificates are matched together). Then threshold setting and the impact of the lack of training data on evaluation will be discussed.

3.3.3 Learning the Conditional Probabilities

To compute the conditional probabilities $P(X_i|Match)$ we use the principle of information excess [14]: if only a subset of the information contained in a record pair is sufficient to label them as a match, then the remaining information can be used to derive domain knowledge. Several matchings are performed by using every attributes but the one we are seeking statistics for. For instance the distribution $P(DeltaYear|Match)$ is computed over a set of matching pairs obtained by forcing high similarity on all attributes but dates.

The probabilities $P\left(X_i|NonMatch\right)$ are computed over a set of non-matching record pairs. To obtain those, record pairs are generated randomly. Indeed, since the set of non-matching pairs is much larger than the set of matching pairs, the vast majority of randomly picked pairs are non-matches. For instance if matching birth certificates and death certificates together we expect $|match| = min(|birth|, |death|) = |death| = 363425$ while the total number of possible pairs is equal to $|birth| \times |death| = 1081532 \times 363425$ (see section 4.2). So in that case the probability of having a match when randomly selecting a pair of records is around 10^{-6}.

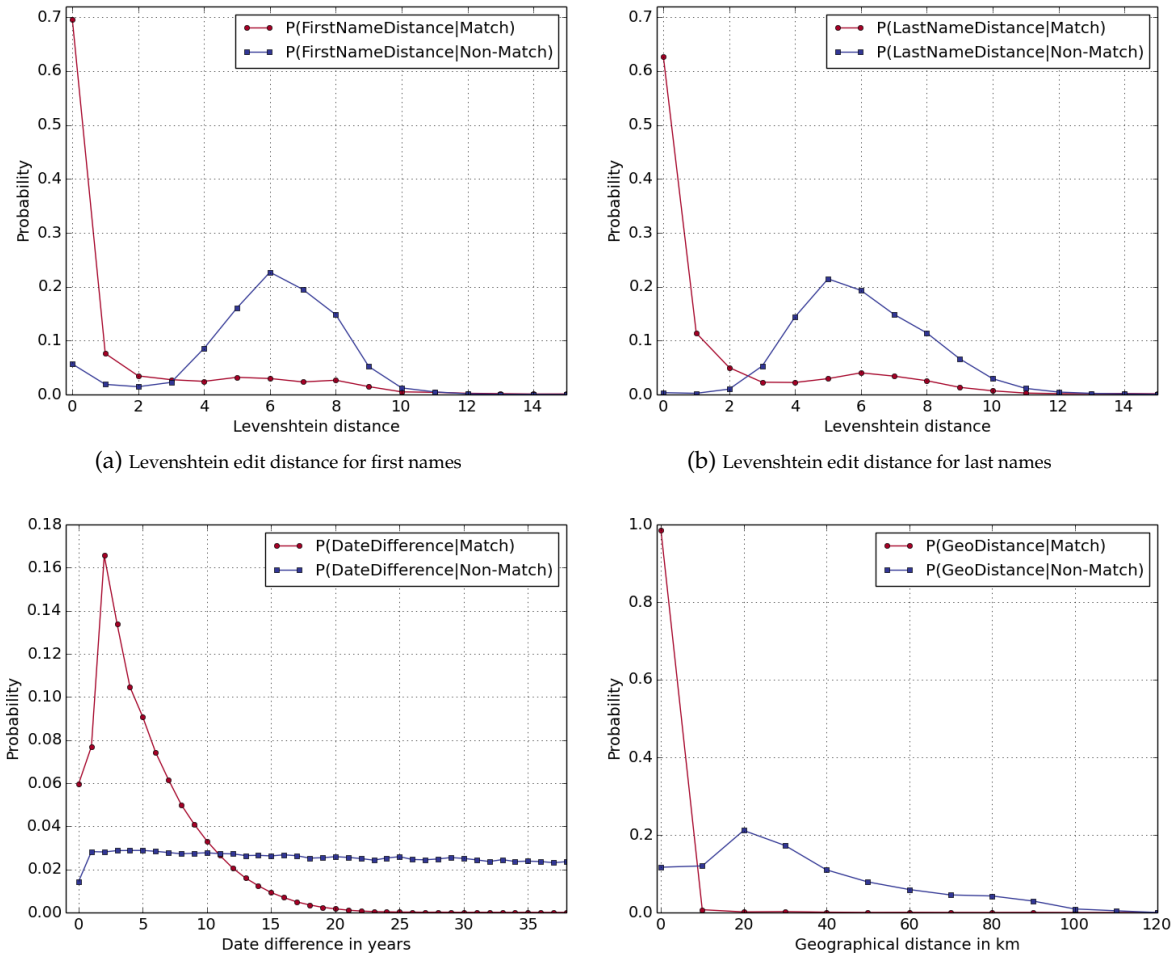
The probabilities are obtained by calculating the frequency of each attribute value over a set of matches obtained by applying the principle of information excess. Figure 3.2c shows the distributions $P(DeltaYear|Match)$ and $P(DeltaYear|NonMatch)$: the probability for a couple to have two children in the same year is pretty low, while a year difference of two has the highest probability. $P(DeltaYear|NonMatch)$ is more flat. We note that the quotient $\frac{P(DeltaYear|Match)}{P(DeltaYear|NonMatch)}$ starts to play a negative role in the score computation for $DeltaYear > 10$.

Figure 3.2d shows the distributions for geographical distance. The vast majority of matching certificates were recorded in the same place.

Figures 3.2a and 3.2b show similar distributions for first names and last names respectively. We note for instance that the quotient $\frac{P(FirstNameEditDistance|Match)}{P(FirstNameEditDistance|NonMatch)}$ starts to play a negative role in the score computation for $FirstNameEditDistance > 3$. This is consistent with our intuition that if two persons are the same they will have very similar names.

As the quality of the whole entity resolution schema depends on the quality of the probability distributions, which are computed over subsets of matching pairs, acquisition of those pairs must be done carefully. The only way to obtain them is to arbitrarily set very restrictive thresholds on their similarity values: we must be sure that the vast majority of returned pairs are indeed matching. These thresholds are based on human domain knowledge and the only ways to assess the correctness of the returned supposedly matching pairs are to:

1. Assess that the probability distributions $P(X_i|Match)$ computed over the pairs are conform to their expected shape. For instance, the distributions shown in fig 3.2 hint that birth certificates with identical parents most probably refer to parents with close to identical names, are separated by less than 10 years and occur in places less than 5 kilometers away; which is conform with our expectations.



(a) Levenshtein edit distance for first names

(b) Levenshtein edit distance for last names

(c) Date difference in years

(d) Geographical distance in km

Figure 3.2: Probability distributions between birth certificates in parents to parents matching

2. Assess the quality of the data matching's end result by manually labeling a set of matched pairs as true or false positives.

Both those techniques require domain knowledge. As mentioned in the start of this section, more attributes than strictly necessary to perform a match must be available. The more attributes the better. If not, we cannot learn the conditional probabilities necessary to compute the similarity scores. Also, the databases must contain enough records so that the restrictive thresholds comparisons yield large enough amounts of pairs to compute the probability distributions with precision.

In short, our entity resolution schema allows to match records without any training data, but it requires domain knowledge as well as the availability of many records with many attributes. We will see in sections 4.4 to 4.6 that the technique works fine with parents to parents and parents to newlyweds matching, as two pairs of names as well as dates and places are available, but it does not allow to effectively match newborns with parents as only a single pair of names as well as dates and places are available.

3.3.4 Learning a Threshold: Evaluation Step

During a threshold-based classification, the overall similarity score obtained for each candidate pair is compared to a threshold θ. The threshold must be chosen in such a way that both precision and recall are maximized (see section 2.5). θ is usually learned by performing a matching procedure over a training set then computing recall and precision for several values of the threshold. This leads to precision-recall curves allowing to easily find an optimal threshold. Figure 2.5 from section 2.5 gives an example of such PR curve.

Since there is no training set in our case, it is impossible to obtain such PR curve:

1. Thankfully, **precision** ($= \frac{TP}{TP+FP}$) can easily be computed for different thresholds based on the result of our matching procedure since number of positives are known and number of true positives can be found through manual labeling of a subset pairs classified as matches.

2. However, the exact **recall** ($= \frac{TP}{TP+FN}$) cannot be computed: since the exact number of true matches is unknown, the number of false negatives cannot be found.

There is however a workaround to compute the best threshold without knowing the exact recall yielded by each threshold value. Let R_1 and R_2 be the recall values for thresholds θ_1 and θ_2, we can find which threshold yields the better recall by knowing the precision P_1 and P_2 and the number of positives M_1 and M_2 obtained when using the thresholds at hand. Indeed:

$$P_1 \times M_1 > P_2 \times M_2 \Leftrightarrow R_1 > R_2$$

The left part of the implication simply tells that θ_1 yields more true positives than θ_2. Obviously, the threshold yielding the highest number of true positives yields the highest recall, as the total amount of true matches $TP + FN$ is a constant. So in our case instead of precision-recall curves we use precision-true positives curves. The best threshold is the one that jointly maximizes precision and amount of true positives. Figure 3.3 shows precision (vertical axis) and amount of true positives (horizontal axis) for different threshold values over a set of 1000 positive pairs labeled by hand. Each point on the curve corresponds to a value for the threshold. We chose to use the threshold associated to $(265, 0.98)$ as it yields good precision while keeping a large amount of true positives.

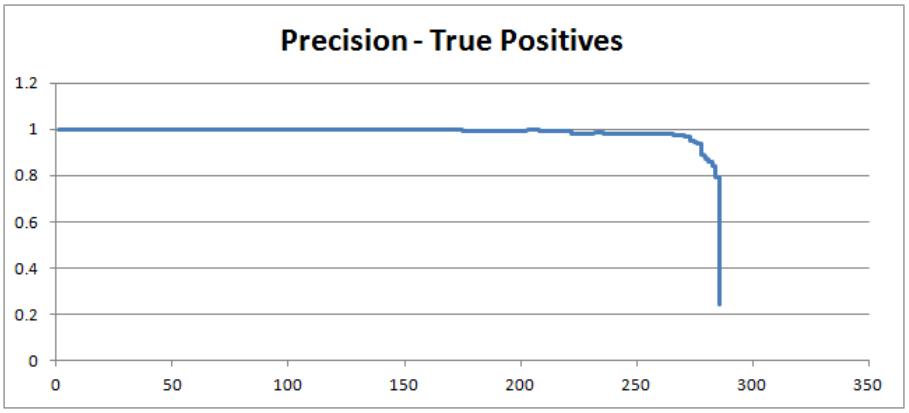


Figure 3.3: Precision to number of true positives computed over a subset of 1000 manually labeled pairs for parents to parents matching

In practice, this figure was obtained by manually labeling a set of 1000 candidate pairs. For each pair, the score value outlined in subsection 3.3.2 is available. The pairs are ordered decreasingly according to their score. Each pair is manually labeled: we learn for each pair if it is a true positive or a false positive. For each score value, the number of true positives and precision are computed as if that score value was used as a threshold by counting the number of true positives with a score above the value at hand and the number of pairs with a score above the value at hand.

Obviously, the quality of the threshold also depends on the quality of the labeling and on the size of the labeled set.

3.4 Overview of the Schema

Before jumping to the implementation, we conclude this chapter by presenting an overview of our global entity resolution schema: figure 3.4 and algorithm 3 show the complete matching process devised in this work. We summarize the steps of our entity resolution schema:

1. The bit vector indexing technique outlined in section 3.2 is used to find candidate pairs worth comparing.

2. During the pairwise comparison, attributes set X is computed for each pair. It consists in Levenshtein edit distance (see subsection 2.2.1) between stakeholders' names, year difference between certificates' dates and distance in kilometers between places.

3. The similarity score $\frac{\prod_{i=1}^{d} P(X_i|Match)}{\prod_{i=1}^{d} P(X_i|NonMatch)}$ outlined in subsection 3.3.2 is then computed based on the attributes set and its associated conditional probabilities (which were learned such as discussed in subsection 3.3.3).

4. The threshold used in classification is chosen (such as discussed in subsection 3.3.4) based on a manually labeled subset of candidate pairs using a precision-true positives curve (see figure 3.3).

5. The comparison of each pair's similarity score with the threshold finally yields the classification of the pair as a match or a non-match. Note that this last step can be skipped depending on the application. With no threshold-based filtering, the final output is a list of candidate pairs each with an associated similarity score. It can be used to help researchers explore the data as it gives a ranking of record pairs similarity. In our case we keep using a threshold-based classifier because:

   (a) We do not have enough manpower to do such exploration.

   (b) We do not have the time nor storage space to save all candidate pairs as several hundreds of candidate matches can be found for each record. So we save only the pairs classified as matches.

   (c) To assess the quality of our schema, we need to automatically classify candidate pairs as matches or non-matches. This allows us to then manually label a set of pairs classified as matches and compute precision and number of true positives over it.

Algorithm 3 Matching Records Together

Input: $R_1 \leftarrow$ *Fetch records from database* 1
Build bit vector tree $T \leftarrow R_1$
Input: $R_2 \leftarrow$ *Fetch records from database* 2
for all *Record* $r_2 \in R_2$ **do**
  Candidate pairs $CP \leftarrow T.treeTraversal(r_2)$
  for all *Candidate pair* $(r_1, r_2) \in CP$ **do**
    $X \leftarrow ComputeFeatures(r_1, r_2)$
    for all *Feature* $X_i \in X$ **do**
      Compute $f_i = \frac{P(X_i|Match)}{P(X_i|NonMatch)}$
    end for
    if $Score = \prod_i f_i \geq Threshold$ **then**
      Output: *Store matching pair* (r_1, r_2)
    end if
  end for
end for

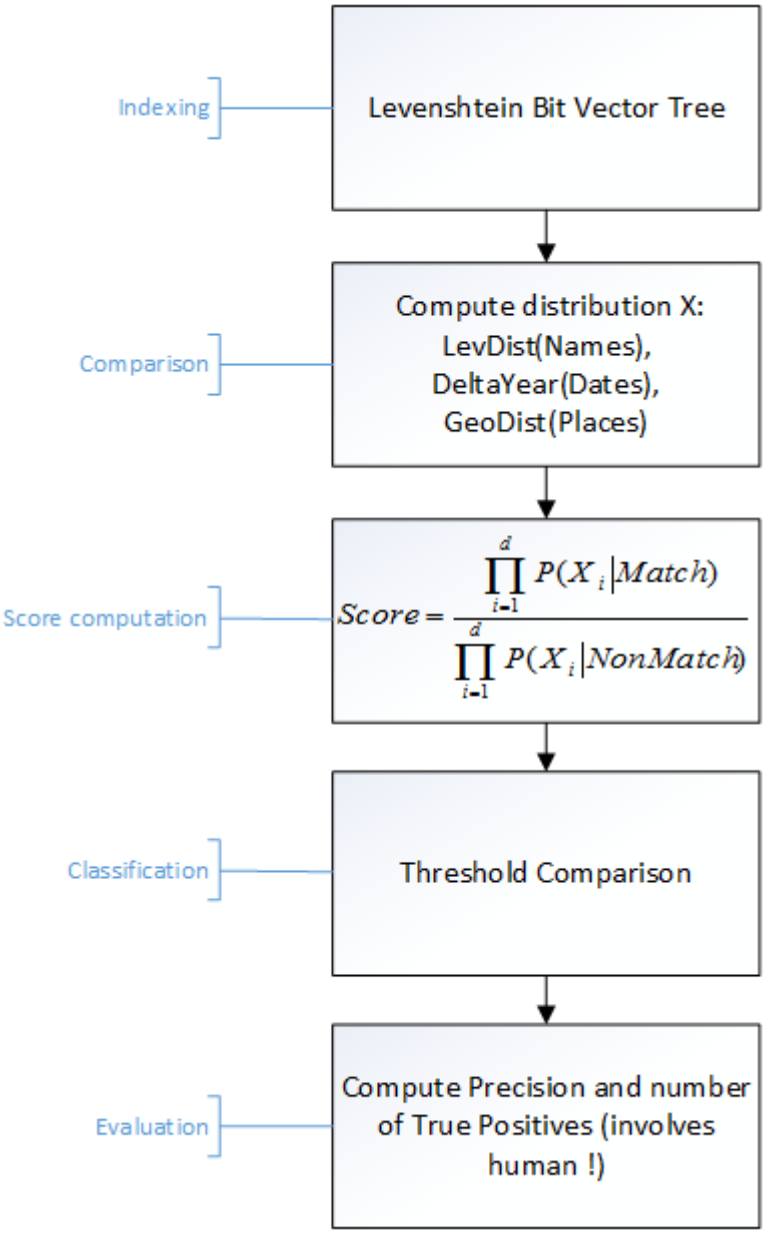


Figure 3.4: Our Complete Matching Process

Chapter 4

Project Implementation

In this chapter we implement, test and discuss our customized entity resolution schema. But first, we describe the parsing and the storing in a relational database of the certificates, which were initially available as XML files.

4.1 Building a SQL Database of DTB Records

The data is originally available in three XML files:

- DTB-D.xml contains birth church records. It is 7.86 GB and contains 1 081 532 records.
- DTB-B.xml contains death church records. It is 1.11 GB and contains 363 425 records.
- DTB-T.xml contains marriage church records. It is 1.73 GB and contains 306 701 records.

To allow us to explore, fetch and update data more easily, a XML parser was made using Java to load the data from the XML files into a relational SQL database. There are two main families of XML parser's:

1. **DOM parsers** load the entire file into memory and use the DOM structure of the file to find data of interest.
2. **SAX parsers** read the file line by line and use flags to find data of interest.

Due to the size of the files, it was not possible to load them in main memory, thus making the use of a DOM parser impossible.
The parser was written in Java using the **SAXParser** library. It reads the file line by line and uses three callback functions:

- **startElement:** called every time the parser reads an opening tag. It identifies which tag is being opened by assigning an open flag.
- **endElement:** called every time the parser reads a closing tag. It closes related opened flags.
- **characters:** called when the parser reads data between an open and a close tag. It caches the data being read.

Figure 4.1 shows the relational database schema.

Figure 4.1: The Database Schema

Each record is characterized by the place and the date of the event at hand, as well as the ID of the register containing the record. Records are linked to related stakeholders through foreign keys. For every occurrence of a reference to a person in a record, a row in the table "Person" is created. A person appears as many times in the table "Person" as there are certificates referring to him/her. Our goal is to match records together so that records referring to the same people can be linked.

4.2 Data Exploration

Tables 1.2, 1.3 and 1.4 from section 1.1 show the different types of certificates at hand and their fields. In this section we count the number of certificates available in the database and the availability of their attributes. To do so, some SQL queries were made to collect simple facts about the data:

- There are 1 081 532 birth records. 1 072 624 children with non-empty first name and last name are registered. 742 447 birth records have all data available: father, mother and child first and last name as well as place and date.
- There are 363 425 death records. 338 398 deceased people with non empty first name and last name are registered.
- There are 306 701 marriage records. 300 034 of them reference both a bride and a groom with non empty first name and last name. Furthermore, only 224 666 records have non empty bride, groom, place and date.
- For 155 817 people referenced as bride or groom in a marriage record there is a filled in place of birth in the marriage record.
- There are 7 477 070 references to a stakeholder across all records.
- 801 236 references to a father in birth records with non empty first name and last name.
- 752 020 references to a mother in birth records with non empty first name and last name.
- 1 231 777 references to a witness in birth records with non empty first name and last name.
- 13 references to the father of a deceased person in death records.
- 5 references to the mother of a deceased person in death records.
- 18 016 references to a relation of a deceased person in death records with non empty first name and last name. Recorded relation seem to be the wife most of the time.
- 17 301 references to the previous wife of a bridegroom in marriage records with non empty first name and last name.
- There are no people referenced as being the previous husband of a bride in marriage records.
- 301 references to the witness of a marriage with non empty first name and last name.
- 85 023 birth records, 39 164 marriage records and 133 590 death records don't have a date filled in.
- 88 156 birth records, 44 762 marriage records and 313 death records don't have a place filled in.
- 80 108 birth records, 5054 marriage records and 193 death records have both date and place missing.

4.2.1 Conclusions About the Quality of the Data

Those numbers allow us to draw several conclusions over the quality of the data:

- For most people, both parents are known through his or her birth certificate.
- The fields: relation, father and mother from death records is of limited use. Similar for the fields previous wife, previous husband and witness from marriage record.
- There are only a few fields we have on a consistent basis:
    1. For a **birth** record we know a place, a date as well as the newborn's names and his or her parents' names.
    2. For a **marriage** record we know a place, a date as well as the newlyweds' names.
    3. For a **death** record we know a place, a date as well as the deceased person's name.
    4. For each reference to a **person** only the first name and the last name are filled in consistently. We also note that references to newborns usually don't have any patronymic filled in.

The XML samples shown in Appendices C, D and E are consistent with those conclusions.

4.3 Data Cleaning

As stated in section 1.1 and verified in the previous section, the quality of the data at hand is quite poor. A first cleaning step is necessary, in particular to deal with the following issues:

- Valid variants of **person first names**[2, 17]. They can be standardized[1]. We used a dictionary of standard names from the Meertens Instituut [1] for men and for women to find those standard forms. When a first name has two standard forms (one for male and one for female) both are recorded.
- The **gender** of a person is not consistently filled in. It could be derived from the person's first name. However, some first names can refer to a man or a woman. For instance, "Jan" is used by both genders. We can at least fill in accordingly the gender field of people referenced as bride, groom, mother or father.
- The special case of **stillborn** children can be treated: when a child dies at birth, he or she often does not have a first name. Stillborn children are most of the time recorded with "Zoon", "Dochtertje van", "Kind van", ... as first name. All those different "names" used to describe a child who died at birth can be standardized: a standard first name "Stillborn" can be used instead of the different variations.

We now have to apply our entity resolution schema: in the next sections we outline the matchings discussed in section 3.1.

4.4 Matching Parents Together

We apply our schema to parents to parents matching: the goal is to link together birth certificates in which parents are the same. The features available for this matching are listed in table 4.1. The probability distributions used to match parents together are presented in figure 3.2 from subsection 3.3.3.

Birth record 1	Birth record 2	Availability
Father First Name	Father First Name	Always
Father Patronymic	Father Patronymic	Sometimes
Father Last Name	Father Last Name	Always
Mother First Name	Mother First Name	Always
Mother Patronymic	Mother Patronymic	Sometimes
Mother Last Name	Mother Last Name	Always
Place	Place	Most of the time
Date	Date	Most of the time

Table 4.1: Features available for parents to parents matching

Parents to parents matching is a many to many relationship exploration: a couple can be matched with several others since the same couple can have several children. We used our algorithm to match parents together. Once this was done we ran a small script to take transitive closure into account: if A and B are matching and if B and C are matching, A and C should be matching. The threshold was set using figure 3.3 from subsection 3.3.4 in such a way that both precision and number of true positives are maximized. For 560 949 out of the 742 447 birth certificates with non-empty parents names, place and date, at least one match was found. The 180 000 records for which no match was found were either only children or false negatives. To assess the quality of the matching algorithm, a subset of 1000 pairs classified as matches were randomly selected and labeled by hand. Only 10 false positives were found, which yields a confidence interval $P(0.984 < Precision < 0.996) \approx 0.95$.

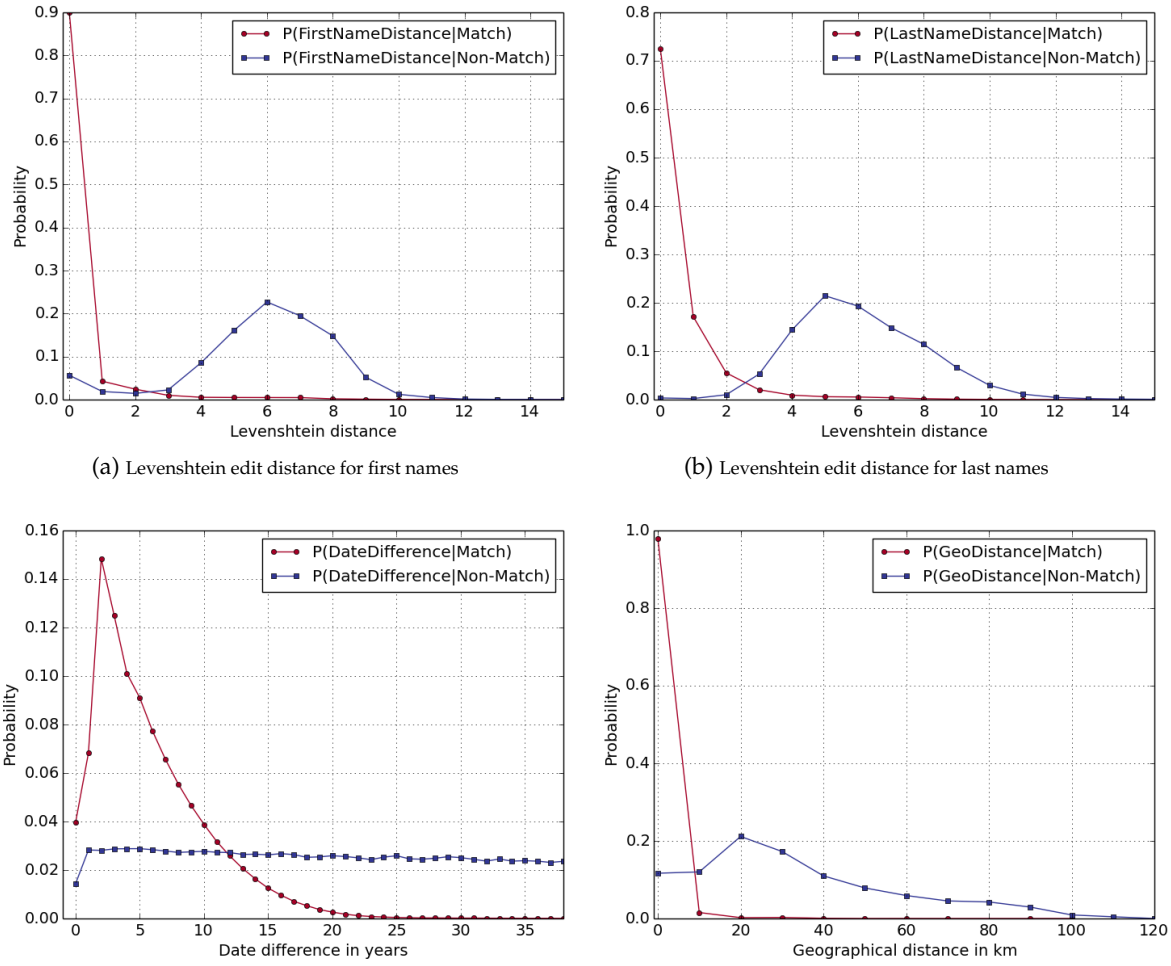
The use of standard names and probabilistic distributions allows to be relatively lenient for name variations. For instance, we detail the score computation and classification for the following sample records pair:

- R_1**:** Anthonii Geerits and Helenae Cornelissen have a child in Zeeland on the 1752-09-30.
- R_2**:** Antonii Gerits and Helenae Cornelii have a child in Zeeland on the 1750-01-14.

[1] `http://www.meertens.knaw.nl/nvb/`

Anthonii and Antonii are both standardized to Antonius, which yields an edit distance of 0 for fathers' first names and of 1 for last names. The edit distance is 0 for mothers' first names and 4 for mothers' last names. There is a geographical distance of 0 kilometers and a year difference of 2 between R_1 and R_2. Using the graphs presented in figure 3.2 we obtain the corresponding conditional probabilities and enter them into the score formula. We obtain a score of 392 810 which is above the threshold set to 300 000. It is a match.

To further assess the quality of our matching technique and to empirically verify the validity of the hypothesis of conditional independency between features, made in subsection 3.3.1, we recompute the probabilities $P(X|Match)$ based on the set of all matching pairs we just found. Figure 4.2 shows those distributions. They are very similar to the ones showed in figure 3.2 which were used to compute the similarity scores. This tends to intuitively show the good quality of our matching technique (very dissimilar distributions would have meant that the results are bad).



(a) Levenshtein edit distance for first names

(b) Levenshtein edit distance for last names

(c) Date difference in years

(d) Geographical distance in km

Figure 4.2: Probability distributions between matched birth certificates after parents to parents matching

We verify empirically the hypothesis of conditional independency between several attribute pairs X, Y by computing their correlation coefficient r_{XY}. A value near -1 for the coefficient indicates a strong negative linear dependency between X and Y, while a value near 1 indicates a strong positive linear dependency. If the value of the coefficient is 0, there is no linear dependency between X and Y:

$$-1 \leq r_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \leq 1$$

where σ_{XY} is the covariance of X and Y:

$$\sigma_{XY} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})$$

and σ_X, σ_Y are the standard deviations of X and Y:

$$\sigma_X = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \quad \sigma_Y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

Those values were computed for several pairs of attributes over the set of matched pairs, as shown in table 4.2. Note that we are only testing the existence of linear dependency between our features. This is because we expect monotonic relationships between our features, if any.

X	Y	σ_X	σ_Y	σ_{XY}	r_{XY}
Delta Year	Geographical Distance	4.50	2.57	0.85	0.074
Delta Year	First Names Edit Distance	4.50	1.04	0.43	0.091
First Names Edit Distance	Last Names Edit Distance	1.04	1.16	0.08	0.068
Geographical Distance	First Names Edit Distance	2.57	1.04	0.18	0.066

Table 4.2: Correlation Coefficients for Pairs of Features

As every $r_{XY} < 0.1$ we conclude that there is little to no conditional dependency between the features used to compute our similarity score: the hypothesis of conditional independency should be valid enough to yield good results.

Note that the correlation coefficient was not computed for all possible pair of attribute: we make the assumption that for instance if there is no correlation between geographical distance and first name there should not be any correlation between geographical distance and last name. We also suppose that this result can be generalized to the following newlyweds to parents matching and newborns to parents matching.

4.5 Matching Parents with Newlyweds

In the previous subsection we linked together birth certificates in which parents are the same. We can thus now consider newlyweds to parents as being a one to one relationship: each couple gets married once then has a number of children. To perform the matching we compare the names of the newlyweds, the place of marriage and the date of marriage with the names of parents, place of birth and date found in the birth certificate of their first child and with the birth date of their last child. Since it is a one to one relationship, each newlywed couple is matched with the parent couple for which the similarity score is the highest. To avoid getting too many false positives, a threshold below which pairs are considered as non matching and discarded was set based on a hand labeled set of 1000 candidate pairs. For the 300 034 marriage records in which both bride and groom names are filled in, we found 63 353 matches. This amount is surprisingly small considering that the set of birth certificate is the most complete one we have and that at that time people usually got married before getting children.

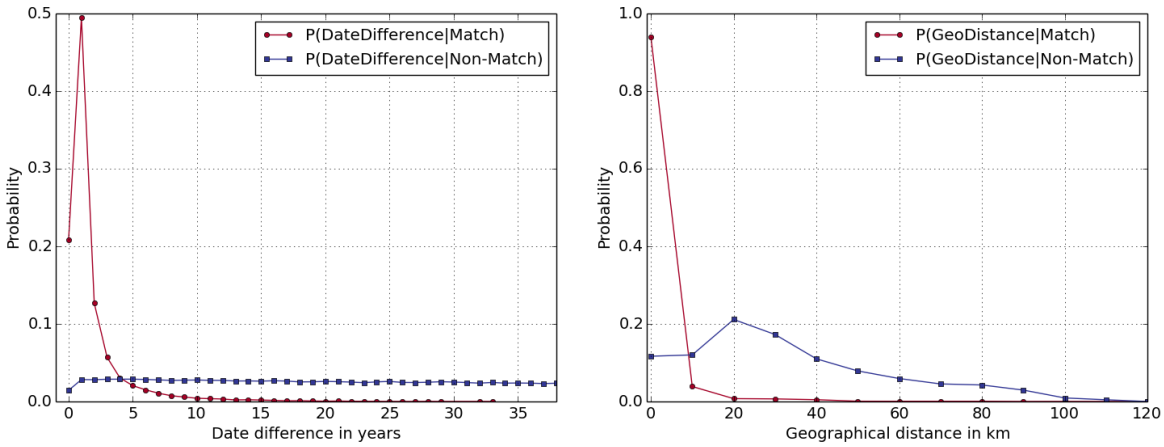
Table 4.3 shows the features available for newlyweds to parents matching.

Marriage record	Birth record	Availability
Groom First Name	Father First Name	Always
Groom Patronymic	Father Patronymic	Sometimes
Groom Last Name	Father Last Name	Always
Bride First Name	Mother First Name	Always
Bride Patronymic	Mother Patronymic	Sometimes
Bride Last Name	Mother Last Name	Always
Place	Place	Most of the time
Date	Date of first child's birth	Most of the time
Date	Date of last child's birth	Most of the time

Table 4.3: Features available for newlyweds to parents matching

Figure 4.3a shows the distribution for date differences in years between dates of marriage and parenting. They were obtained by forcing high similarity on stakeholders names, places and consistency on dates (no child before marriage, no child too many years after marriage). Figure 4.3b shows the probability distribution for distance between places for marriage to parenting. $P(GeoDistance|Match)$ was computed over a set of marriage-birth pairs obtained by forcing high similarity on parents and newlyweds names

and dates to be consistent. $P(GeoDistance|NonMatch)$ is the same than for parents to parents matching. Likewise, $P(DeltaYear|NonMatch)$ is the same than for parents to parents matching. Indeed, we consider that attribute distributions between completely unrelated certificates do not depend on the certificates' types. Distributions for first and last names found for parents to parents matching are also re-used here (see figures 3.2a and 3.2b from section 3.3.3).



(a) Date difference in years

(b) Geographical distance in km

Figure 4.3: Probability distributions between marriage and birth certificates in newlyweds to parents matching. Distributions on edit distance on first names and last names are the same as in figure 3.2

The threshold was set based on a thousand labeled pairs. The pairs were obtained by selecting a random subset of 1000 marriage records and for each searching the birth record yielding the highest similarity score. Figure 4.4 shows the precision-true positives curve yielded by that labeled subset. We chose a threshold value which corresponds to a precision of 0.99 and allows to find 280 our of the 282 true positives among our 1000 pairs.

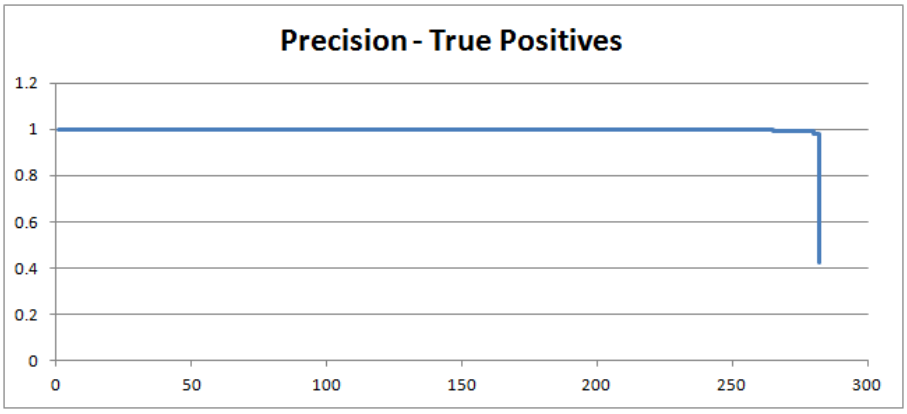
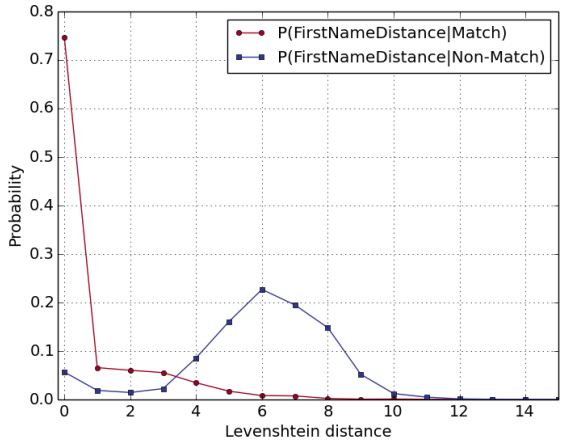
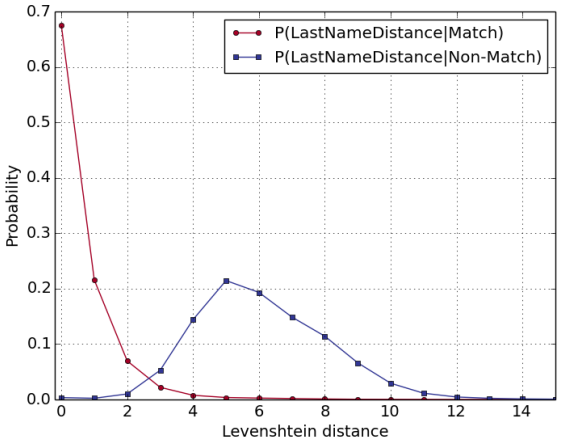


Figure 4.4: Precision to number of true positives computed over a subset of 1000 manually labeled pairs for newlyweds to parents matching

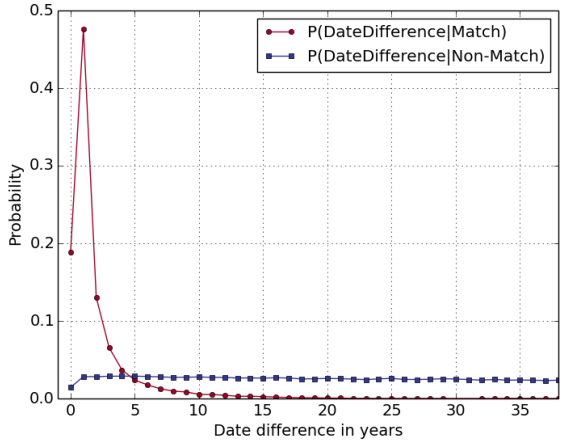
To assess the quality of our matching, we compute the precision over a set of 1000 labeled positive pairs: 36 false positives were found, which leads to a confidence interval: $P(0.952 < Precision < 0.975) \approx 0.95$. We also plot (see figure 4.5) the probability distributions computed over the set of all matched pairs. They look very similar to the ones used to compute the scores (see figures 4.3a and 4.3b from this section and figures 3.2a and 3.2b from section 3.3.3), which is reassuring.



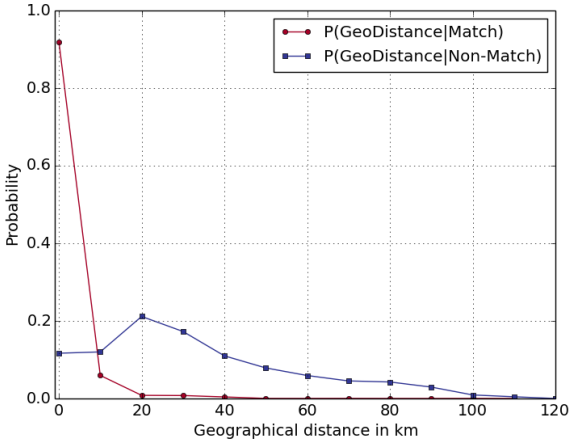
(a) Levenshtein edit distance for first names



(b) Levenshtein edit distance for last names



(c) Date difference in years



(d) Geographical distance in km

Figure 4.5: Probability distributions between matched marriage and birth certificates after newlyweds to parents matching

4.6 Matching Newborns with Parents

The goal of this matching is to build genealogical trees by finding for each person referenced as a newborn in a birth certificate, the matching birth certificate(s) in which the same person is referenced as father or mother. Recall that our technique is based on conditional probabilities which are computed thanks to information excess. In other words, for our technique to work properly we need to have more features available than strictly necessary to perform a match.

Person's birth record	Person's child's birth record	Availability
Newborn First Name	Parent First Name	Always
Newborn Patronymic	Parent Patronymic	Almost never
Newborn Last Name	Parent Last Name	Always
Place	Place	Most of the time
Date	Date of first child's birth	Most of the time

Table 4.4: Features available for newborns to parents matching, note that patronymic of a newborn is almost never mentioned

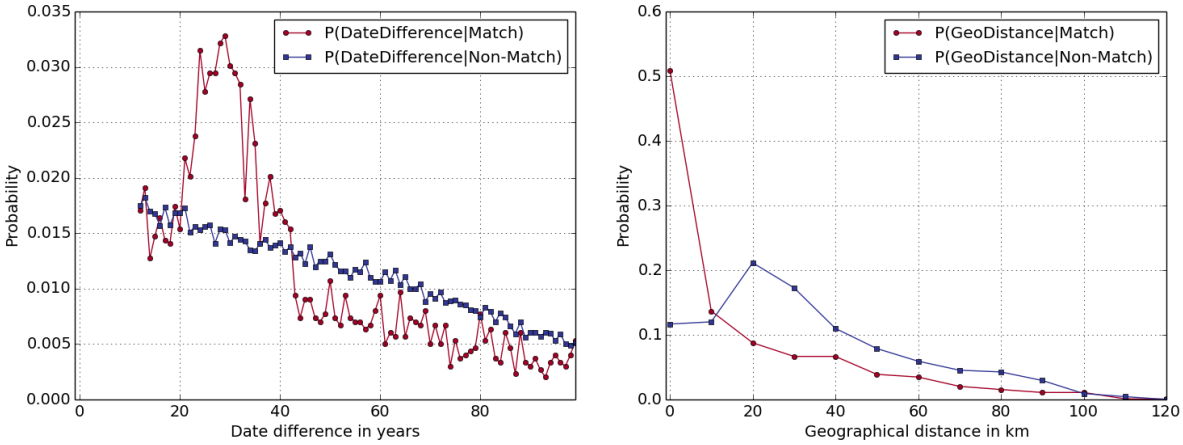
As shown in table 4.4, only a single person's first and last name as well as certificate place and date are consistently available for newborns to parents matching. This lack of attributes makes difficult the discovery of the conditional probability distributions necessary to compute similarity scores:

- Thankfully, distributions on edit distance for first names, patronymics and last names computed for parents to parents matching can be re-used: if two records refer to the same person, their names are

very similar most of the time.

- Distribution on distance between recorded places is obtained by forcing high similarity on stakeholders names, which are also forced to be unpopular (high similarity on unpopular names has more value than on popular names). Dates are simply forced to be consistent: one could not have children before nor after a certain age. Records for which more than one match are found are also discarded (it is a one to one relationship). This led to the distribution showed in figure 4.6b.
- Finally, distribution on year difference between recorded dates is obtained by forcing the same attribute similarities than for distances, but we also force the places filed in both certificates to be close (below 30 kilometers afar). This led to the distribution showed in figure 4.6a.

Due to the lack of attributes, we were forced to set very restrictive filters to obtain the sets used to compute the distributions $P(GeoDistance|Match)$ and $P(DeltaYear|Match)$. Because of this, those sets are very small and both contain only around 3000 records. The resulting curves look very uneven. They seem to show that usually people got children between their 20 and 40 and less than 10 kilometers away from their own birth place.



(a) Date difference in years

(b) Geographical distance in km

Figure 4.6: Probability distributions between birth certificates in newborn to parent matching. Distributions on edit distance on first names and last names are the same as in figure 3.2

To find a threshold for the classification, we proceed as usual with the labeling of candidate pairs for which similarity scores were computed using the above curves. However, it seems clear that the quality of the labeling will not be as good as for parents to parents and newlyweds to parents, as the lack of attributes makes this labeling hazardous. For instances:

- Are the Dionisius Corneliuszn born in 1613 at Loon op Zand and the Dionisius Jan Denis Cornelissen who had a child in 1666 at Oosterhout (10 km away) the same person ?
- Are the Joanna Catharina Herck born in 1785 at Zundert and the Anna Catharina Klerck mother in 1809 at Zundert the same person ?

Those are two occurrences among many. In doubt, we consider them as being non-matches. Thus, only the pairs for which first and last names are identical and places and dates plausible can be labeled as matches without any doubt. This nullifies the benefices of human labeling as classifying pairs for which names are identical as matches and pairs for which a difference small enough for a doubt to exist as non matches can be automated. The manual labeling of 600 candidate pairs led to the precision-true positives curve shown in figure 4.7. We chose to use the threshold corresponding to the point (64,0.95).

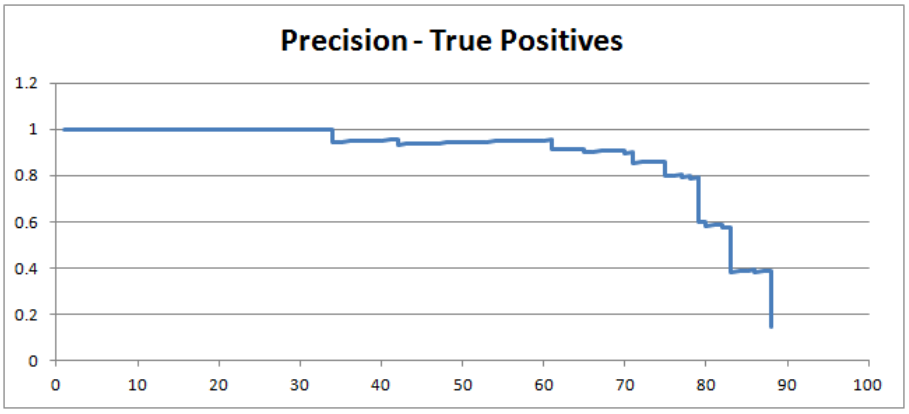
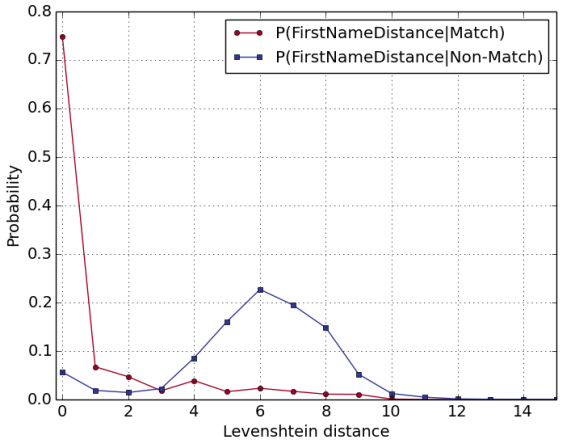


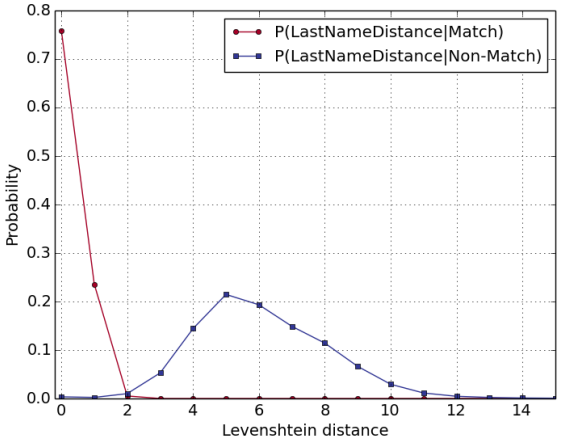
Figure 4.7: Precision to number of true positives computed over a subset of 1000 manually labeled pairs for newlyweds to parents matching

We found 39 252 matching pairs after applying our algorithm to a test sample of 250 000 randomly selected birth certificates. The manual labeling (still hazardous) of a thousand of those pairs led to a relatively mediocre confidence interval for the precision of $P(0.841 < Precision < 0.883) \approx 0.95$. Figure 4.8 shows the probability distributions computed over the matching pairs found by our algorithm. While the distributions for first and last name look similar to the ones used to compute the score (cf figure 3.2 from subsection 3.3.4) and the distribution for geographical distance is likewise similar to the one shown in figure 4.6b, the distribution for years differences however looks really uneven with a peak at 29 years.

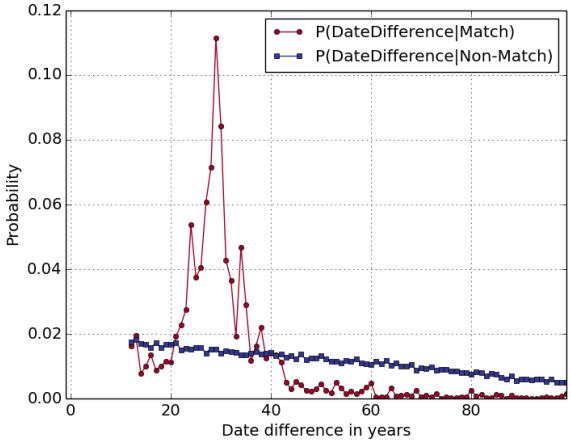
It seems clear now that our schema is somewhat not suitable for single person to single person matching, as the amount of available features is not sufficient. This lack of data makes impossible the collection of reliable conditional probability distributions necessary to compute similarity scores. The manual labeling of pairs, necessary to configure the classifier and to assess the quality of the matching schema, is hazardous as it is not humanly possible to decide whether or not two certificates refer to the same person only based on first name, last name, date and place. With that in mind, we drop the newborns to deceased people step.



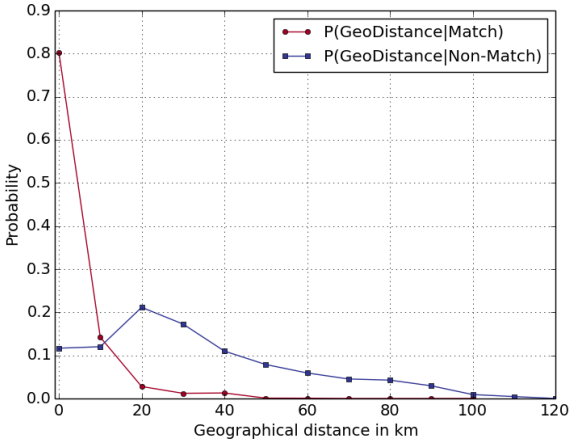
(a) Levenshtein edit distance for first names



(b) Levenshtein edit distance for last names



(c) Date difference in years



(d) Geographical distance in km

Figure 4.8: Probability distributions between matched birth certificates after newborns to parents matching

4.7 Comparison with an Arbitrary Technique

To assess the quality of a new entity resolution technique, it has to be compared with current state of the art techniques. Such techniques are usually based on a training set of some sort, which is not available in our case. In our custom entity resolution technique the similarity scores are computed based on conditional probabilities learned thanks to the principle of information excess (see subsection 3.3.3). For comparison purposes, we devise an arbitrary technique in which the scores are computed based on our domain knowledge. That is why we qualify this technique as being "arbitrary": instead of learning the scores in a data-driven fashion, we set them based on our own knowledge.

To compare the techniques, we apply both of them to newlyweds to parents matching. The application of our Bayes based technique is described in section 4.5. A surprisingly low amount of 63 353 matching pairs was found with a satisfying confidence interval for the precision of $P(0.952 < Precision < 0.975) \approx 0.95$.

Our arbitrary technique uses the same bit vector tree indexing as our Bayes-based entity resolution schema. The score computation, however, is different. The global similarity score is the normalized sum of similarity functions for each pair of features:

$$Score = \frac{menFNSim + menLNSim + womenFNSim + womenLNSim + dateSim + placeSim}{6}$$

where FN stands for First Name and LN stands for Last Name, and names' scores are the normalized Levenshtein similarity scores (as described in subsection 2.2.1) between stakeholders' names.

The arbitrariness appears when we have to define score functions for dates and places. If the duration between two dates and the distance between two places can easily be computed, the question is what distance and duration should yield high similarity ? Based on our intuition and domain knowledge, we decided that:

- A year difference of one between date of marriage and date of first child's birth should yield the highest score value while no more than a 38 years difference should be possible between marriage and child's birth. We filter out all record pairs with delta years above 38 and consider a linear function for similarity:

$$dateSim = 1 - \frac{|1 - deltaYear|}{38}$$

- Marriage and first child's birth should take place in very close locations: if the distance between the two certificates is below 5 kilometers, *placeSim* is set to 1. Otherwise it is set to 0.

The classification is performed in an identical way as previously: the similarity score computed for each candidate pair is compared to a threshold, which is learned based on a hand labeled set of candidate pairs. Figure 4.9 shows the precision to true positives curve resulting from the hand labeling of 500 candidate pairs. We chose the point (122, 0.96) on the curve as it corresponds to both high precision and number of true positives. It translates to a threshold of 0.83 on the global similarity score.

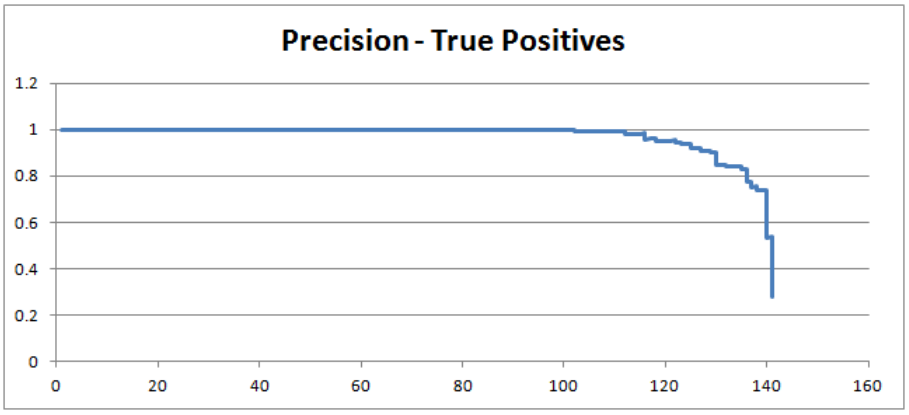
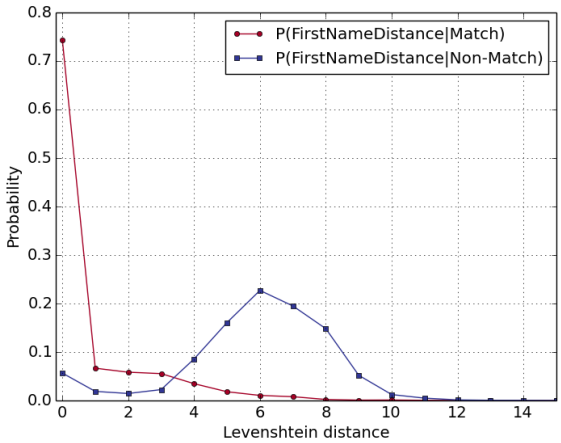


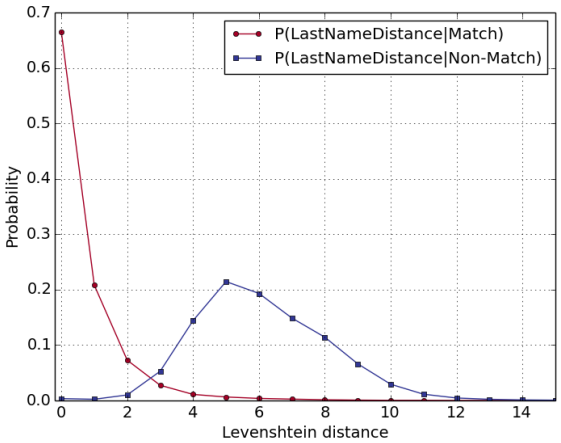
Figure 4.9: Precision to number of true positives computed over a subset of 500 manually labeled pairs for newlyweds to parents matching using arbitrary technique

Once the threshold chosen, the technique can run on the whole data set. 64 302 matching pairs were found. A thousand of those pairs were manually labeled to compute a confidence interval for the precision of $P(0.940 < Precision < 0.966) \approx 0.95$. Figure 4.10 shows the conditional probability distributions computed over the matched pairs using our arbitrary technique. Figures 4.10a and 4.10b look very similar to figures 3.2a and 3.2b from subsection 3.3.4. Figure 4.10d looks similar to figure 4.3b from section 4.5. The only notable difference is that figure 4.10c is less steep than figure 4.3a from section 4.5. This is surely due to the fact that a linear function in years difference was used to compute the similarity score for dates.

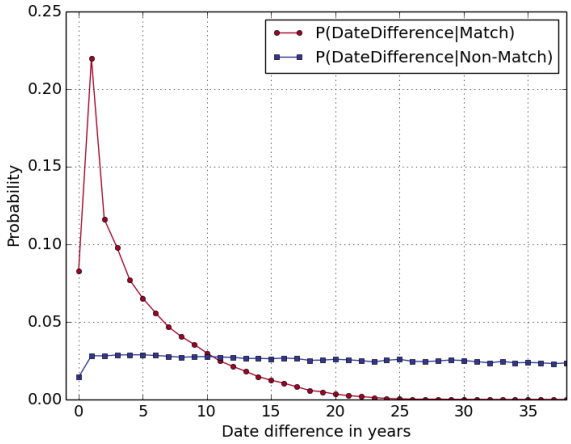
This comparison did not yield strong evidence that the Bayes-based technique leads to better results than the arbitrary one. However, the setting of score functions where it must be decided what year difference and geographical distance should yield the highest similarity between two records is non trivial. It requires to actually know what time interval and geographical distance usually separate matching records (and we learned that by computing the conditional probability distributions...). Such prior knowledge, usually not readily available, is not necessary with the Bayes-based technique, as it can be obtained through the computation of conditional probabilities using the principle of information excess.



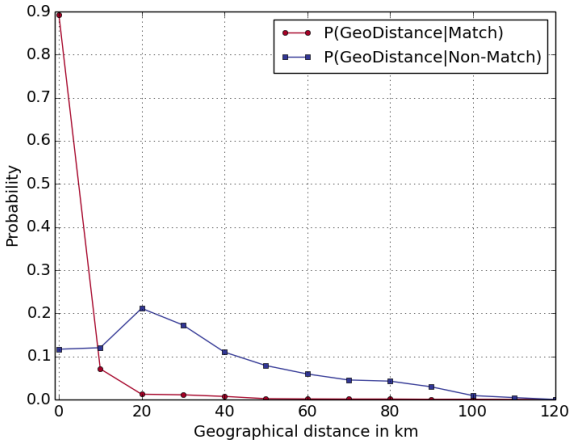
(a) Levenshtein edit distance for first names



(b) Levenshtein edit distance for last names



(c) Date difference in years



(d) Geographical distance in km

Figure 4.10: Probability distributions between matched marriage and birth certificates after newlyweds to parents matching using the arbitrary technique

Chapter 5

Conclusions

Our initial goal was to link together birth, marriage and death certificates issued in the province of North-Brabant in the Netherlands between the years 1580 and 1811. This project had a large research component, as it was the first attempt to apply entity resolution techniques to these certificates. As such, we first had to explore our data set and assess its quality, which proved to be low and uneven. Such low quality made predictions of the project's final outcome very unsound and it was decided to adopt a step by step approach: each phase of the entity resolution schema would be a step of our project.

For indexing, we used a bit vector tree technique for Levenshtein distance: it is effective, intuitive, easy to implement and to tweak, with the added benefit that it does not present the coverage issue inherent to blocking techniques. It is also computationally fast enough for the sake of our application. As such, it is a good choice for a first entity resolution approach. However, better results can be obtained by using an adaptation of this bit vector tree technique for Jaro distance (discussed in Appendix I), while better performances can be attained by instead using Levenshtein automata indexing (discussed in subsection 2.3.3). Both those techniques are strong contenders for future works.

As we explored classification and evaluation techniques, the issue of training data availability became more and more pressing. Indeed, most state of the art classification techniques are based on machine learning: they require training data to learn the parameters of classification. Such data are usually either based on the results of previous entity resolution projects on equivalent data sets or manually obtained by having volunteers finding pairs of matching records. Neither of those was available in our case and we had to devise a workaround entity resolution schema, based on Bayes theory and on the principle of information excess, to overcome the lack of training data.

Our entity resolution schema leads to encouraging results when matching couples together (see sections 4.4 and 4.5) as the relatively high amount of features available, two pairs of person names, a date and a place, allows to collect reliable probability distributions and to perform manual labeling of returned candidate pairs with confidence. However when it comes to single person to single person matching, our technique does not work properly, as the few available attributes, a single pair of person names, a date and a place, no longer allow for the collection of probability distributions nor to perform manual labeling of candidate pairs with confidence, as discussed in section 4.6.

As the vast majority of entity resolution techniques available in literature are based on a training set of some sort, we hope that our work contributes to the research of entity resolution without training data. Our work also allowed to find duplicate certificates in the data set (the same certificate indexed several times) and to match together parents couples and newlyweds with parents couples.

Appendices

Appendix A

Thesis Catalog Description

Automatic detection of name variations

Toon Calders (WIT)

For this project a large data collection consisting of historical birth, death, and marriage certificates of the province of North-Brabant in the Netherlands is available. This collection contains certificates for about 3 million people, from 1580 until 1955. This collection of paper documents has been indexed by volunteers. For many of the certificates (unfortunately the index is not complete yet), the names of the people involved in it, and their role have been recorded in a database. Consider for instance the following example of an index entry for a death certificate:

	Death certificate
Deceased	Johanna Louise Fredrika Frans
Relation of the deceased	Gerard Cornelius Reincke de Sitter
Father of the deceased	Carl Ludwig Frans
Mother of the deceased	Alida Philippina Zehender
Type of deed	death certificate
Number of deed	5
Place	Beers
Date of decease	26-02-1825
Period	1825
Contains	Overlijdensregister 1825
Number of inventory	50
Record number	456

There are, however, several problems with the data recorded by the volunteers:

1. Volunteers made mistakes when recording the names
2. Natural name variations occur; for instance, during the Napoleonic era, Willem preferred to be called Guillaume. After the French left the Netherlands, Willem became Willem again. Other, less spectacular variations: Fredrika versus Frederika.
3. Another source of variation is the granularity at which locations are reported. Sometimes locations have been reported at suburb or even neighborhood level, whereas in other records only the city is reported.
4. Also the original data contained errors. For instance, the order of names may have been swapped.

The goal of this graduation project is to automatically detect name variations for location and person names, using statistical and data mining methods. Because of the large size of the database it is very likely that most name variations occur frequently. In a pilot study, it was shown that name variations could be detected by finding pairs of full names sharing most surnames, but not all. The differences often were name variations. Your task will be to extend this approach to also include locations, and exploit additional background knowledge such as: for most birth certificates there is a matching death certificate, no one has more than one birth and death certificate, etc. This project has a large research component, so your creative input will be required as well. For this project it is absolutely not necessary to speak or understand Dutch.

Appendix B

Popularising Article

Entity Resolution in Historical Documents for Family Tree Discovery

Aurélien Plisnier[1], Julia Efremova[2], and Toon Calders[1,2]

[1]*Université Libre de Bruxelles, Belgium,*
[2]*Eindhoven University of Technology, The Netherlands*

1 Introduction

As a person's life goes on, certificates recording his birth, marriage, the birth of his children then finally his death are emitted. In Holland, from around 1580 to the appearance of Napoleonic era certificates in 1811, this task was carried on by priests: they recorded birth, marriage and death certificates in their churches registers. As time went on and people moved on, the life events of every single person were scattered across several certificates, often stored in different registers and churches. Now, those certificates are some of the last traces left by these people. Thankfully, they are being retrieved and manually digitized by volunteers of the BHIC [1].

The purpose of this work is to support genealogical and historical research by linking together those certificates. As no personal identification number was available at that time, partly identifying features such as person names, birth date or living place are to be compared. Table 1 shows the different types of certificates and the attributes they contain. The linking of those records must allow to trace back the lives of these ancestors and to build their genealogical tree.

Birth certificate	FirstName, LastName, Gender, BirthDate, BirthPlace, FatherFirstName, FatherLastName, MotherFirstName, MotherLastName
Death certificate	FirstName, LastName, Gender, DeathDate, DeathPlace
Marriage certificate	GroomFirstName, GroomLastName, BrideFirstName, BrideLastName, MarriageDate, MarriagePlace

Table 1: Available features for each certificate type

1. Brabants Historisch Informatie Centrum https://www.bhic.nl/het-geheugen-van-brabant

2 Problem Definition

Currently, there are around 1 000 000 birth, 350 000 marriage and 300 000 death certificates available in XML format. To search through such a large set for certificates referring to the same person is not humanly possible and must be automated. This requires the use of entity resolution techniques. Entity resolution, also called data matching or record linkage, is the process of linking together records referring to the same entities, initially stored in different data sources [1]. This allows to aggregate data which was scattered across several sources: a classical example is the matching of two commercial databases containing records of a firm's customers. As each database may contain different data about the customers, the merging of the two allows to build a more complete customer profile, enabling a better targeted marketing.

The application of entity resolution in our case consists in the linking of certificates in which stakeholders are the same. For instance, finding all birth certificates in which parents are the same allows to find siblings. Also, for each person mentioned as a newborn in a birth certificate, finding the birth certificates in which the same person appears as father or mother allows to build genealogical trees.

Most state of the art entity resolution techniques are based on machine learning [1]: to function they need to learn how to recognize a pair of matching records based on training examples. Such examples, usually drawn from human input or from the results of prior entity resolution projects, are not available in our case and would be too costly to obtain in sufficient amount. Thus, the contribution of this work is the definition of a workaround entity resolution technique which does not require the prior availability of training data.

3 Approach

Our technique is based on a statistical model: for a pair of certificates we compute the probability that those two certificates refer to the same person, given their features distribution. Let X be a random variable denoting the configuration of a records pair's features. we need to compute the following conditional probability:

$$P(Match|X) = ?$$

It represents the probability that two certificates refer to the same person, given their features configuration X. The Bayes theorem for statistics allows to write:

$$P(Match|X) = \frac{P(X|Match)\,P(Match)}{P(X)}$$

It reduces the problem to the computation of the conditional probability $P(X|Match)$, which represents the probability for two certificates to present the attributes configuration X, given that both certificates refer to the same person. Such probabilities can be computed using the principle of information excess outlined in [2]: if it is possible to make correspond two certificates using only a subset of their features, the remaining features can be used to derive domain knowledge.

In practice, we compute the contribution of each feature to the probability $P(X|Match)$ over a subset of matching records pairs obtained by forcing high similarity on all other features. Consider for instance the linking of birth certificates in which parents are the same: to obtain the contribution of dates on the similarity we compute the distribution of years differences between dates over a subset of matching records pairs obtained by forcing places, mother first and last names as well as father first and last names to be identical. The result is presented in figure 1: it shows that two birth certificates in which parents are the same are usually separated by 0 to 10 years and that a difference of more than 11 years probably corresponds to non-matching certificates.

By repeating the operation for every feature, it is possible to compute $P(Match|X)$, which allows to decide whether or not two certificates are matching. All of this without training data.

4 Discussion and Conclusions

The application of this technique led to encouraging results for the linking of couples, such as parents to parents or newlyweds to parents, as they provide ample amount of features. However, it has showed its limitation when applied to the matching of single persons, such as newborn to parent or newborn to deceased person, as the few amount of features offered by those, a single pair of names, a date and a place, is not sufficient to compute the probability distributions.

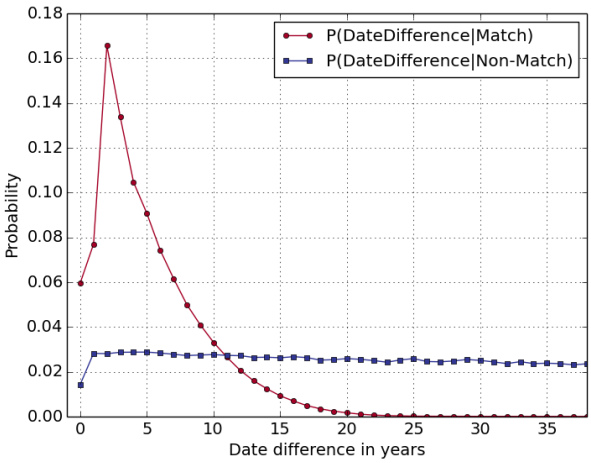


Figure 1: Probability distributions for year differences between birth certificates in parents to parents matching

In short, we have developed an entity resolution schema allowing to overcome the lack of any training data. However, it requires the records to present more features than strictly necessary for the matching. As the vast majority of entity resolution techniques available in literature are based on a training set of some sort, we hope that our work contributes to the research of entity resolution without training data.

5 Acknowledgment

We thank Bijan Ranjbar-Sahraei, Marijn Paul Schraagen and Alexander Panchenko for their advice, guidance and support throughout this project.

References

[1] Peter Christen. *Data matching*. Springer, 2012.
[2] Marijn Schraagen. *Aspects of record linkage*. PhD thesis, Universiteit Leiden, 2014.

Appendix C

Sample Birth Certificate XML DTB Record

```
<records>
<record uuid="0005c577-4703-ffec-892f-b18190ffc4b8">
        <field name="uuid" label="uuid" formtype="Uuid">
            <value>0005c577-4703-ffec-892f-b18190ffc4b8</value>
        </field>
        <field name="ordering" label="ordering" formtype="System"/>
        <field name="guid" label="GUID" formtype="text"/>
        <field name="Register" label="Register" formtype="TextEntiteitRecord">
            <entity uuid="a2e1ffd7-9443-44e5-9288-11c9caff9036" name="doop-, trouw- en
                 begraafboeken">
                <record uuid="d6c1faa0-7288-2da5-ce99-afccaea84a21"/>
            </entity>
        </field>
        <field name="plaats" label="Plaats dopen" formtype="text">
            <value>Deurne</value>
        </field>
        <field name="datum" label="Datum dopen" formtype="Customdate">
            <value>1797-10-02</value>
        </field>
        <field name="Kind" label="Kind" formtype="ChildEntity">
            <entity uuid="a8f38f90-58af-427c-bae1-3258a5c342ef" name="kind">
                <record uuid="0d3dcaec-4641-11e3-a747-d206bceb4d38">
                    <field name="modified_by" label="modified_by" formtype="System">
                        <user/>
                    </field>
                    <field name="modified_time" label="modified_time" formtype="System
                        ">
                        <value>2013-11-05 10:46:52</value>
                    </field>
                    <field name="ordering" label="ordering" formtype="System"/>
                    <field name="voornaam" label="Voornaam" formtype="text">
                        <value>Arnoldus</value>
                    </field>
                    <field name="patroniem" label="Patroniem" formtype="text"/>
                    <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                        "/>
                    <field name="geslachtsnaam" label="Achternaam" formtype="text">
                        <value>Giimans</value>
                    </field>
                    <field name="alias" label="Alias" formtype="text"/>
                    <field name="plaats_geboorte" label="Plaats geboorte" formtype="
                        text"/>
                    <field name="datum_geboorte" label="Datum geboorte" formtype="
                        Customdate"/>
                    <field name="geslacht" label="Geslacht" formtype="radio">
                        <value>m</value>
```

```
            </field>
            <field name="leeftijd" label="Leeftijd" formtype="text"/>
            <field name="diversen" label="Diversen" formtype="textarea"/>
        </record>
    </entity>
</field>
<field name="Vader" label="Vader" formtype="ChildEntity">
    <entity uuid="34d88351-9fa6-4005-b17f-dc14d0c7c430" name="vader">
        <record uuid="0d3e0b74-4641-11e3-a747-d206bceb4d38">
            <field name="modified_by" label="modified_by" formtype="System">
                <user/>
            </field>
            <field name="modified_time" label="modified_time" formtype="System
                ">
                <value>2013-11-05 10:46:52</value>
            </field>
            <field name="ordering" label="ordering" formtype="System"/>
            <field name="voornaam" label="Voornaam" formtype="text">
                <value>Willebrordus</value>
            </field>
            <field name="patroniem" label="Patroniem" formtype="text"/>
            <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                "/>
            <field name="geslachtsnaam" label="Achternaam" formtype="text">
                <value>Giimans</value>
            </field>
            <field name="alias" label="Alias" formtype="text"/>
            <field name="diversen" label="Diversen" formtype="textarea"/>
        </record>
    </entity>
</field>
<field name="Moeder" label="Moeder" formtype="ChildEntity">
    <entity uuid="81c421ce-f3ea-49d7-bcff-9db0fc21de89" name="moeder">
        <record uuid="0d3e426a-4641-11e3-a747-d206bceb4d38">
            <field name="modified_by" label="modified_by" formtype="System">
                <user/>
            </field>
            <field name="modified_time" label="modified_time" formtype="System
                ">
                <value>2013-11-05 10:46:52</value>
            </field>
            <field name="ordering" label="ordering" formtype="System"/>
            <field name="voornaam" label="Voornaam" formtype="text">
                <value>Maria</value>
            </field>
            <field name="patroniem" label="Patroniem" formtype="text"/>
            <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                "/>
            <field name="geslachtsnaam" label="Achternaam" formtype="text">
                <value>Verhoeven</value>
            </field>
            <field name="alias" label="Alias" formtype="text"/>
            <field name="diversen" label="Diversen" formtype="textarea"/>
        </record>
    </entity>
</field>
<field name="Getuige" label="Getuige" formtype="ChildEntity">
    <entity uuid="1bf5b61f-e5c2-4275-8412-1cf23637ac78" name="getuige">

        <record uuid="0d3eb128-4641-11e3-a747-d206bceb4d38">
            <field name="modified_by" label="modified_by" formtype="System">
                <user/>
            </field>
            <field name="modified_time" label="modified_time" formtype="System
                ">
```

```
                    <value>2013-11-05 10:46:52</value>
                </field>
                <field name="ordering" label="ordering" formtype="System"/>
                <field name="guid" label="GUID" formtype="text"/>
                <field name="akte_guid" label="GUID akte" formtype="text"/>
                <field name="voornaam" label="Voornaam" formtype="text">
                    <value>Maria Jan</value>
                </field>
                <field name="patroniem" label="Patroniem" formtype="text"/>
                <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                    "/>
                <field name="geslachtsnaam" label="Achternaam" formtype="text">
                    <value>N.N.</value>
                </field>
                <field name="alias" label="Alias" formtype="text"/>
                <field name="diversen" label="Diversen" formtype="textarea"/>
            </record>

            <record uuid="0d3e7848-4641-11e3-a747-d206bceb4d38">
                <field name="modified_by" label="modified_by" formtype="System">
                    <user/>
                </field>
                <field name="modified_time" label="modified_time" formtype="System
                    ">
                    <value>2013-11-05 10:46:52</value>
                </field>
                <field name="ordering" label="ordering" formtype="System"/>
                <field name="guid" label="GUID" formtype="text"/>
                <field name="akte_guid" label="GUID akte" formtype="text"/>
                <field name="voornaam" label="Voornaam" formtype="text">
                    <value>Hendrick</value>
                </field>
                <field name="patroniem" label="Patroniem" formtype="text"/>
                <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text">
                    <value>van</value>
                </field>
                <field name="geslachtsnaam" label="Achternaam" formtype="text">
                    <value>Bree</value>
                </field>
                <field name="alias" label="Alias" formtype="text"/>
                <field name="diversen" label="Diversen" formtype="textarea"/>
            </record>
        </entity>
    </field>
</record>
</records>
```

Appendix D

Sample Marriage Certificate XML DTB Record

```
<records>
<record uuid="001e466f-b77b-ee5d-6b23-b28609952706">
        <field name="uuid" label="uuid" formtype="Uuid">
            <value>001e466f-b77b-ee5d-6b23-b28609952706</value>
        </field>
        <field name="Register" label="Register" formtype="TextEntiteitRecord">
            <entity uuid="a2e1ffd7-9443-44e5-9288-11c9caff9036" name="doop-, trouw- en
                 begraafboeken">
                <record uuid="2d040670-7d51-ff50-c4d1-6c7f6c03c9f2"/>
            </entity>
        </field>
        <field name="plaats" label="Plaats trouwen" formtype="text">
            <value>Helvoirt</value>
        </field>
        <field name="datum" label="Datum trouwen" formtype="Customdate">
            <value>1794-05-04</value>
        </field>
        <field name="plaats_ondertrouw" label="Plaats ondertrouw" formtype="text"/>
        <field name="datum_ondertrouw" label="Datum ondertrouw" formtype="Customdate">
            <value>1794-04-19</value>
        </field>
        <field name="plaats_proclamatie" label="Plaats proclamatie" formtype="text"/>
        <field name="datum_proclamatie" label="Datum proclamatie" formtype="Customdate
            "/>
        <field name="Bruidegom" label="Bruidegom" formtype="ChildEntity">
            <entity uuid="a477f7e6-b8c1-44d8-bddd-b5d35dc50c54" name="bruidegom">
                <record uuid="162fb73e-4649-11e3-a747-d206bceb4d38">
                    <field name="modified_by" label="modified_by" formtype="System">
                        <user/>
                    </field>
                    <field name="modified_time" label="modified_time" formtype="System
                        ">
                        <value>2013-11-05 10:46:52</value>
                    </field>
                    <field name="ordering" label="ordering" formtype="System"/>
                    <field name="voornaam" label="Voornaam" formtype="text">
                        <value>Willebrordus</value>
                    </field>
                    <field name="patroniem" label="Patroniem" formtype="text"/>
                    <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                        "/>
                    <field name="geslachtsnaam" label="Achternaam" formtype="text">
                        <value>Giimans</value>
                    </field>
                    <field name="alias" label="Alias" formtype="text"/>
                    <field name="weduwnaar" label="Weduwnaar" formtype="checkbox"/>
```

```
            <field name="jongeman" label="Jongeman" formtype="checkbox"/>
            <field name="leeftijd" label="Leeftijd" formtype="text"/>
            <field name="plaats_geboorte" label="Geboorteplaats" formtype="
                text">
                <value>Diessen</value>
            </field>
            <field name="plaats_wonen" label="Woonplaats" formtype="text">
                <value>Helvoirt</value>
            </field>
            <field name="diversen" label="Diversen" formtype="textarea">
                <value>weduwnaar</value>
            </field>
        </record>
    </entity>
</field>
<field name="Bruid" label="Bruid" formtype="ChildEntity">
    <entity uuid="d0d15972-9212-464e-9694-d39b0a472c03" name="bruid">
        <record uuid="1630483e-4649-11e3-a747-d206bceb4d38">
            <field name="modified_by" label="modified_by" formtype="System">
                <user/>
            </field>
            <field name="modified_time" label="modified_time" formtype="System
                ">
                <value>2013-11-05 10:46:52</value>
            </field>
            <field name="ordering" label="ordering" formtype="System"/>
            <field name="voornaam" label="Voornaam" formtype="text">
                <value>Maria</value>
            </field>
            <field name="patroniem" label="Patroniem" formtype="text"/>
            <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                "/>
            <field name="geslachtsnaam" label="Achternaam" formtype="text">
                <value>Verhoeven</value>
            </field>
            <field name="alias" label="Alias" formtype="text"/>
            <field name="weduwe" label="Weduwe" formtype="checkbox"/>
            <field name="jongedochter" label="Jongedochter" formtype="checkbox
                "/>
            <field name="leeftijd" label="Leeftijd" formtype="text"/>
            <field name="plaats_geboorte" label="Geboorteplaats" formtype="
                text">
                <value>Cromvoirt</value>
            </field>
            <field name="plaats_wonen" label="Woonplaats" formtype="text">
                <value>Helvoirt</value>
            </field>
            <field name="diversen" label="Diversen" formtype="textarea">
                <value>jongedochter</value>
            </field>
        </record>
    </entity>
</field>
<field name="Eerdere vrouw" label="Eerdere vrouw" formtype="ChildEntity">
    <entity uuid="938b960f-7f9f-44ea-9799-929531ed073d" name="eerdere vrouw">
        <record uuid="16300342-4649-11e3-a747-d206bceb4d38">
            <field name="modified_by" label="modified_by" formtype="System">
                <user/>
            </field>
            <field name="modified_time" label="modified_time" formtype="System
                ">
                <value>2013-11-05 10:46:52</value>
            </field>
            <field name="ordering" label="ordering" formtype="System"/>
            <field name="voornaam" label="Voornaam" formtype="text">
```

```
                    <value>Willemijn</value>
                </field>
                <field name="patroniem" label="Patroniem" formtype="text"/>
                <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text">
                    <value>van den</value>
                </field>
                <field name="geslachtsnaam" label="Achternaam" formtype="text">
                    <value>Hoef</value>
                </field>
                <field name="alias" label="Alias" formtype="text"/>
                <field name="diversen" label="Diversen" formtype="textarea"/>
            </record>
        </entity>
    </field>
    <field name="Eerdere man" label="Eerdere man" formtype="ChildEntity"/>
    <field name="Getuige" label="Getuige" formtype="ChildEntity"/>
</record>

</records>
```

Appendix E

Sample Death Certificate XML DTB Record

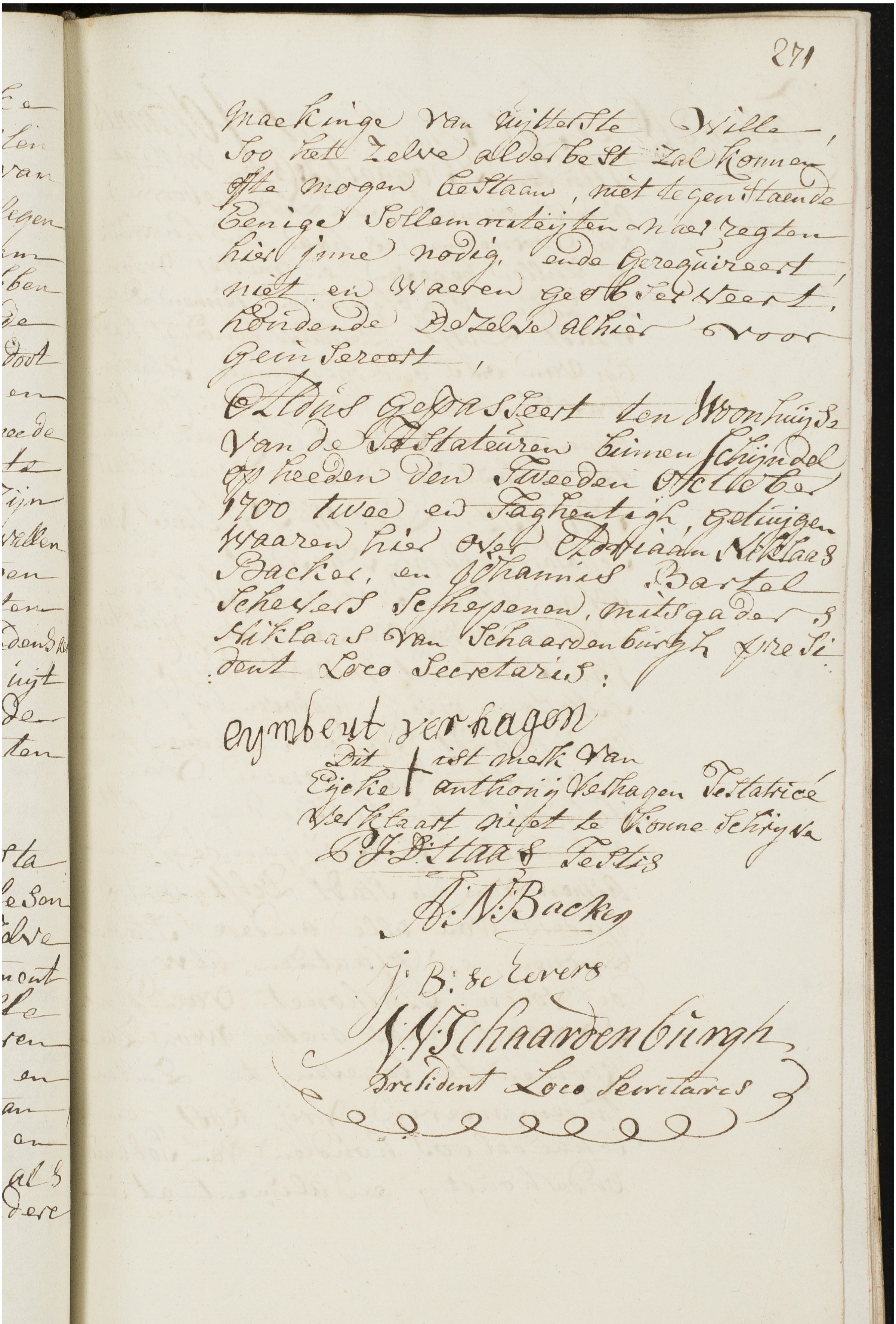
```
<records>
<record uuid="0023a693-20e6-8730-d84a-168ef58126a5">
        <field name="uuid" label="uuid" formtype="Uuid">
            <value>0023a693-20e6-8730-d84a-168ef58126a5</value>
        </field>
        <field name="Register" label="Register" formtype="TextEntiteitRecord">
            <entity uuid="a2e1ffd7-9443-44e5-9288-11c9caff9036" name="doop-, trouw- en
                 begraafboeken">
                <record uuid="a174224f-d40d-ff1c-5656-1b0d71055d29"/>
            </entity>
        </field>
        <field name="plaats" label="Plaats begraven" formtype="text"/>
        <field name="datum" label="Datum begraven" formtype="Customdate">
            <value>1853-11-19</value>
        </field>
        <field name="Overledene" label="Overledene" formtype="ChildEntity">
            <entity uuid="8493c3c2-86b1-4445-8bc4-708a8f57cc7e" name="overledene">
                <record uuid="7263c24e-4648-11e3-a747-d206bceb4d38">
                    <field name="voornaam" label="Voornaam" formtype="text"/>
                        <value>Arnoldus</value>
                    <field name="patroniem" label="Patroniem" formtype="text"/>
                    <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                        "/>
                    <field name="geslachtsnaam" label="Achternaam" formtype="text">
                        <value>Giimans</value>
                    </field>
                    <field name="alias" label="Alias" formtype="text"/>
                    <field name="plaats_overlijden" label="Plaats overlijden" formtype
                        ="text">
                        <value>Asten</value>
                    </field>
                    <field name="datum_overlijden" label="Datum overlijden" formtype="
                        Customdate"/>
                    <field name="geslacht" label="Geslacht" formtype="radio"/>
                    <field name="leeftijd" label="Leeftijd" formtype="text"/>
                    <field name="diversen" label="Diversen" formtype="textarea"/>
                </record>
            </entity>
        </field>
        <field name="Relatie" label="Relatie" formtype="ChildEntity">
            <entity uuid="293319ce-c658-4364-8bec-de846e09deaf" name="relatie">
                <record uuid="7263fa34-4648-11e3-a747-d206bceb4d38">
                    <field name="ordering" label="ordering" formtype="System"/>
                    <field name="relatietype" label="Relatietype" formtype="select"/>
                    <field name="voornaam" label="Voornaam" formtype="text">
                        <value>Ambrosius</value>
```

```
                    </field>
                    <field name="patroniem" label="Patroniem" formtype="text"/>
                    <field name="tussenvoegsel" label="Tussenvoegsel" formtype="text
                        "/>
                    <field name="geslachtsnaam" label="Achternaam" formtype="text"/>
                    <field name="alias" label="Alias" formtype="text"/>
                    <field name="diversen" label="Diversen" formtype="textarea"/>
                </record>
            </entity>
        </field>
        <field name="Vader" label="Vader" formtype="ChildEntity"/>
        <field name="Moeder" label="Moeder" formtype="ChildEntity"/>
    </record>

</records>
```

Appendix F

Picture of a Notary Act


271

maekinge van uijterste wille,
Soo hett zelve alderbest zal konnen
offte mogen bestaan, niet tegenstaende
eenige sollemniteijten naer regten
hier inne nodig, ende gerequireert
niet en waeren geobserveert,
houdende deselve alhier voor
geinsereert,

Aldus gepasseert ten woonhuijse
van de testateuren binnen Schijndel
op heeden den tweeden october
1700 twee en tagentigh, getuijgen
waaren hier over Adriaan Niklaas
Backer, en Johannes Bartel
Schevers schepenen, mitsgaders
Niklaas van Schaardenburgh Presi-
dent Loco Secretaris:

eijmbert verhagen
Dit + ist merk van
Eijcke anthonij verhagen testatricé
verklaart niet te konne schrijva
A.J.D. Klaas testis
A:N:Backer
J:B: Schevers
N:V:Schaardenburgh
President Loco Secretaris

Figure F.1: Picture of a testament notary act

Appendix G

Picture of a Napoleonic Era Certificate

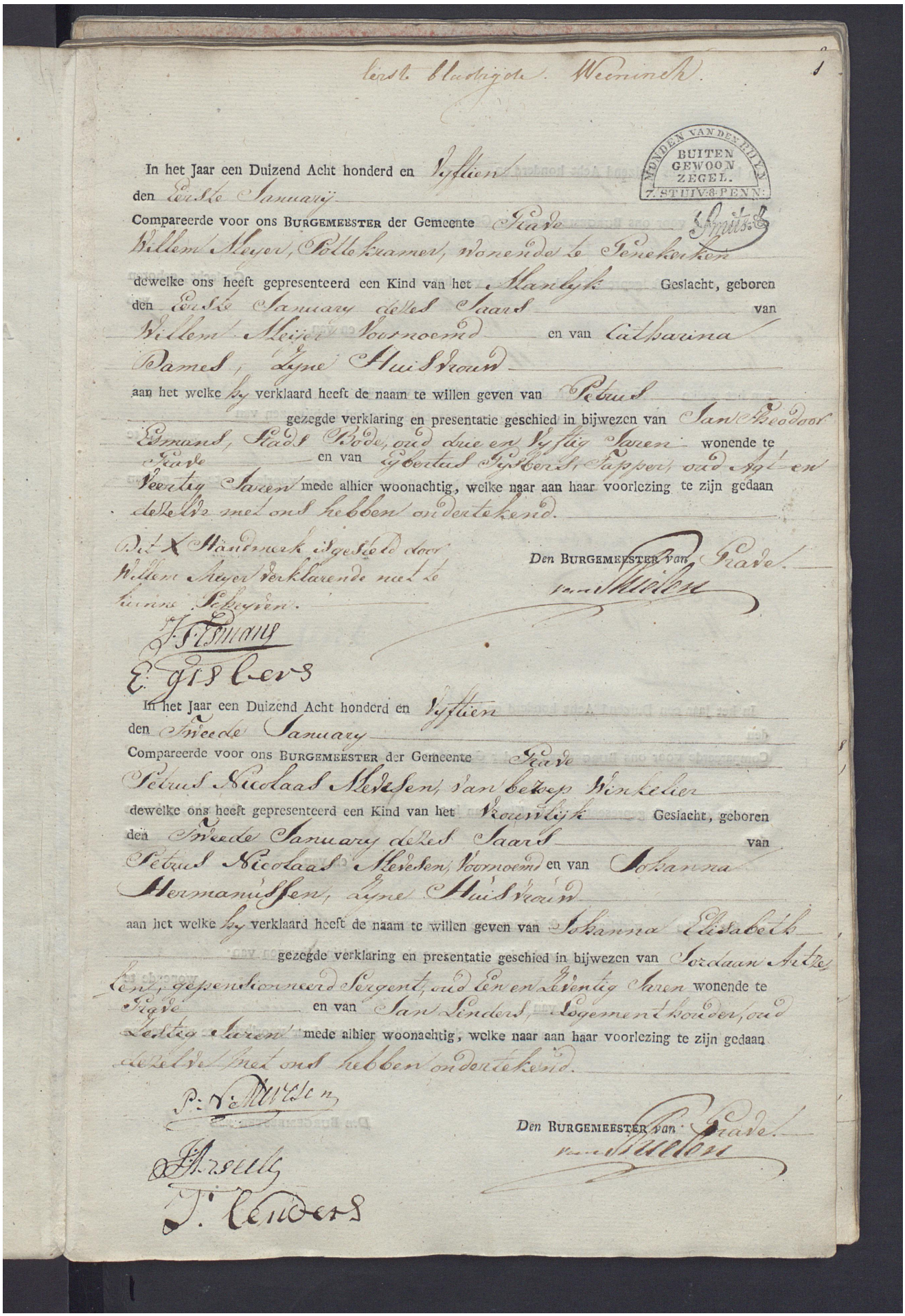


Figure G.1: Picture of a Napoleonic era birth certificate

Appendix H

Preliminary Data Exploration using QlikView

QlikView [1] is a business intelligence tool allowing to query databases and to build interactive dashboards presenting the data in a user friendly manner. We used it to plot some graphs in the early phases of the project, to gain some insight about the data.

Figure H.1 shows the total amount of certificates emitted in some of the most popular places. We see that not all types of certificates are available everywhere. For instance it seems that no one dies in Tilburg, while no one gets married nor dies in Bergen op Zoom and 's-Hertogenbosch...

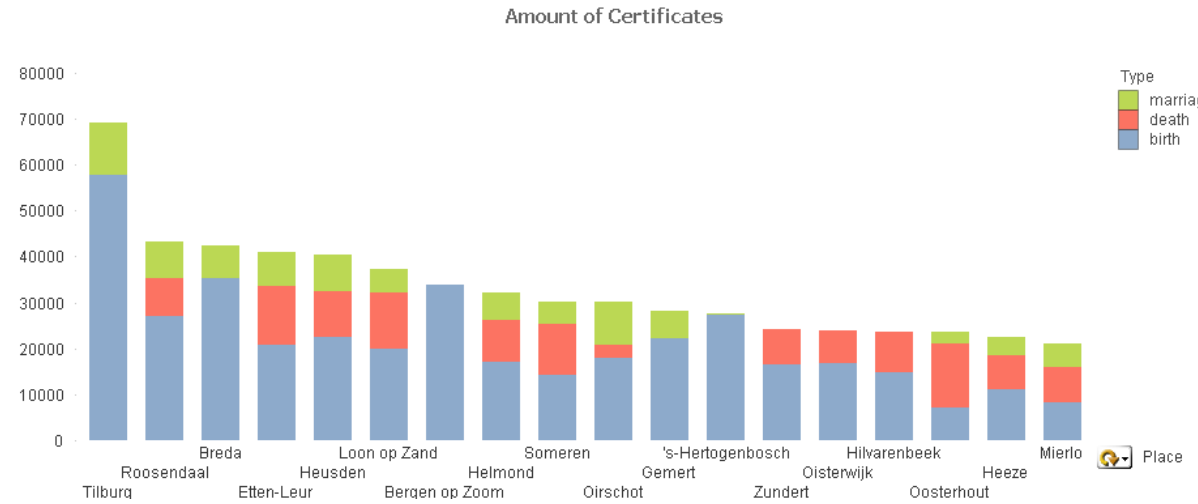


Figure H.1: Distribution of certificates among popular places

Figures H.3 and H.2 show the same kind of distribution for years: for each year the total amount of certificates emitted is plotted. Almost no certificates were recorded before year 1580 nor after year 1811. 1811 corresponds to the replacement of church records by Napoleonic era certificates. Between years 1580 and 1811, the amount of DTB certificates emitted each year increases steadily.

[1] http://www.qlik.com/

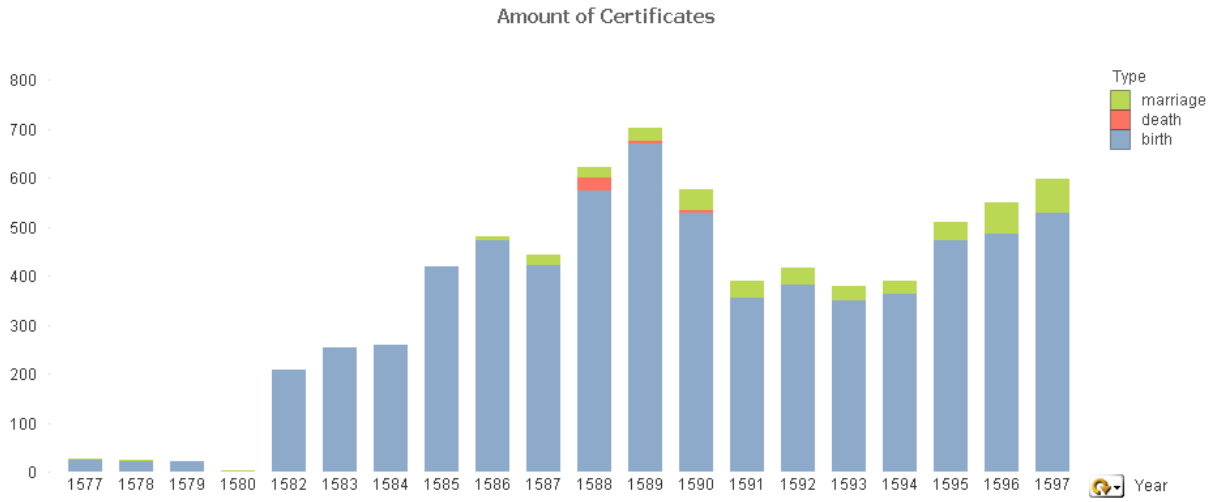


Figure H.2: Distribution of certificates among years 1577 to 1597

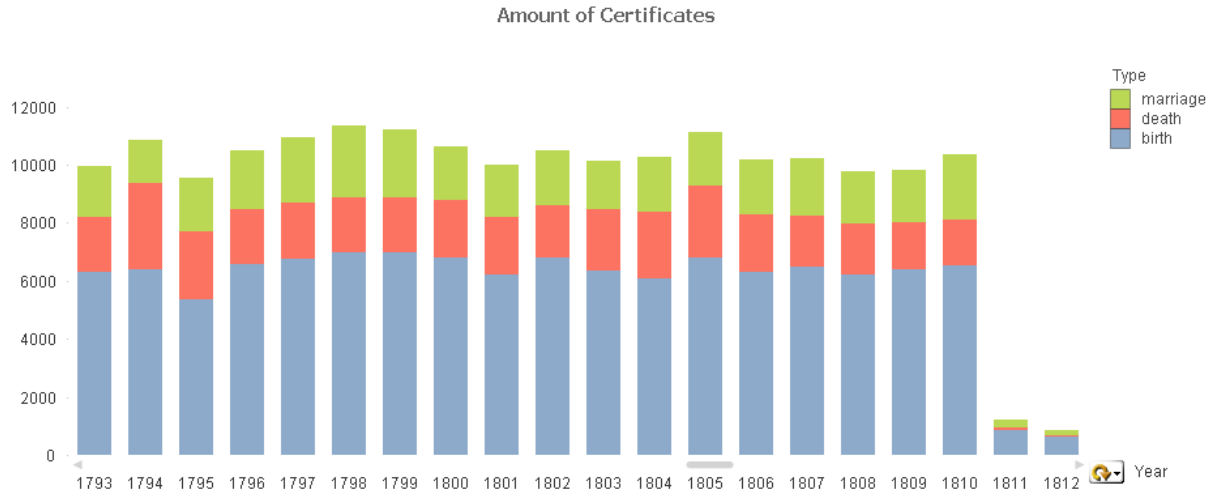


Figure H.3: Distribution of certificates among years 1793 to 1812

Figure H.4 shows the popularity distribution of first name lengths. It shows that the most popular length for a first name is 8 characters while usually first names contain 3 to 10 characters.

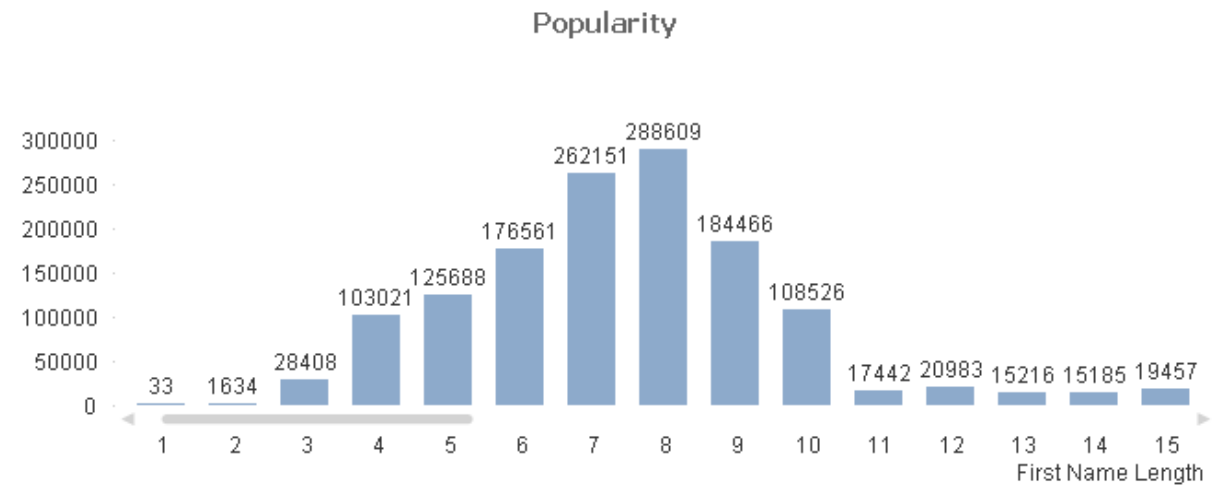


Figure H.4: Length popularity for first names

Figure H.5 shows the popularity distribution of last name lengths. It shows that the most popular length for a last name is 7 characters long while usually last names contain 3 to 11 characters.

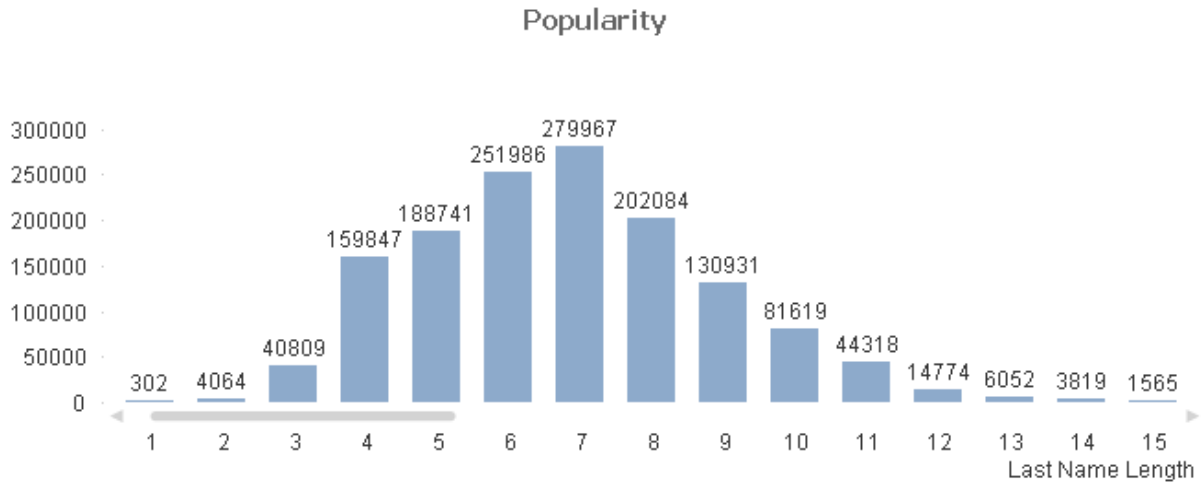


Figure H.5: Length popularity for last names

Appendix I

Bit Vector Tree Indexing for Jaro Distance

This Appendix is based on [14] and [15]. Jaro similarity (described in subsection 2.2.3) is much more permissive than Levenshtein edit distance when it comes to synonym names. For instance the two names "Stefan" and "Stephanus" yield a Jaro similarity of 0.796 and a Levenshtein similarity of 0.555. To use Jaro similarity in our pairwise comparison, we need to adapt our tree indexing (see section 3.1) to make sure that far away leafs will only contain records yielding poor Jaro similarity.

We start with the definition of the basic Jaro distance (see subsection 2.2.3):

$$sim_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}\right)$$

The goal of the indexing phase is to make sure that candidate pairs will yield a Jaro distance above a given threshold th:

$$sim_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}\right) \geq th$$

We isolate c; by doing so we obtain the amount of common characters necessary for the Jaro similarity to be above th:

$$c \geq \frac{3th - \frac{c-t}{c}}{\frac{1}{|s_1|} + \frac{1}{|s_2|}}$$

Let c' be the minimum number of shared characters for which the Jaro similarity is above th. c' is computed by considering a best case scenario in which all shared characters are common within half the length of the longest string and there is no transposition ($t = 0$):

$$c' = \frac{3th - 1}{\frac{1}{|s_1|} + \frac{1}{|s_2|}}$$

c' yields an upper bound on the Jaro similarity. Indeed, $c \leq c'$ and $t \geq 0$: shared characters outside of the window and transpositions have a negative effect on the Jaro similarity. We can thus redefine the error used in the tree traversal: it will be the maximum number of non shared characters. Given a threshold th on Jaro similarity and an input string s of length $|s|$, the error is $e = |s| - c'$. High error in tree traversal will lead to lower c' thus lower maximal Jaro distance: we just hinted that far away leafs will contain records leading to poor similarity score.

We compute e for two strings s_1 and s_2:

$$e = |s_1| - \frac{3th - 1}{\frac{1}{|s_1|} + \frac{1}{|s_2|}}$$

We first note that e grows linearly with $|s_1|$. Consider $|s_2| \approx |s_1|$ (which will be the case in practice, as discussed in the next paragraph), we have:

$$e = \frac{|s_1|}{2}(3 - 3th) \qquad 0 < th \leq 1$$

Our definition of traversal error depends on the lengths of the compared strings, the error being proportional to the strings' lengths. Thus, all strings cannot be stored in a single tree: the error while traversing the tree would be undefined. We will therefore have to break down our vectors tree into several subtrees each containing strings of the same length. A specific traversal error will be available for each subtree. Two things are to be noted:

1. The method is optimized for shorter strings because short strings yield small error thus faster tree traversal.

2. When looking for candidate pairs, no need to traverse all subtrees: only the tree containing corresponding length range and adjacent trees should be visited. Indeed, matching names should have relatively similar lengths.

This adaptation keeps the same traversal complexity of $\binom{k}{e}$, with k being the vector length (= the depth of the tree) and e the maximum permitted error.

This multiple subtrees structure is tricky to implement: for instance in the case of a matching between two couples, the big unique tree, used in our application, built from the concatenation of husband first name, husband last name, wife first name and wife last name bit vectors, would be turned into four collections of subtrees. Storing records in such structure and traversing it is really not intuitive and it was decided that the time investment needed to implement such thing was not worth the benefit of using Jaro distance over Levenshtein edit distance. Indeed, a better permissiveness toward name variations can be attained much more easily by using readily available dictionaries of standard names, thus allowing to replace several synonyms by the same standard form during a prior data cleaning step.

Bibliography

[1] Gerrit Bloothooft. Assessment of systems for nominal retrieval and historical record linkage. *Computers and the Humanities*, 32(1):39–56, 1998.

[2] Christine L. Borgman and Susan L. Siegfried. Getty's synoname and its cousins: A survey of applications of personal name-matching algorithms. *JASIS*, 43(7):459–476, 1992.

[3] Peter Christen. *Data matching*. Springer, 2012.

[4] Julia Efremova, Alejandro Montes Garcia, and Toon Calders. *Classification of historical notary acts with noisy labels*. Eindhoven University of Technology, The Netherlands, 2015.

[5] Julia Efremova and Bijan Ranjbar-Sahraei. *Genealogical Record Linkage based on Modified Markov Logic Networks*. Eindhoven University of Technology, The Netherlands, 2015.

[6] Julia Efremova, Bijan Ranjbar-Sahraei, Frans A. Oliehoek, Toon Calders, and Karl Tuyls. *Investigation of a Baseline Method for Genealogical Entity Resolution*. Maastricht University, The Netherlands, 2012.

[7] Julia Efremova, Bijan Ranjbar-Sahraei, Hossein Rahmaniand, Frans A. Oliehoek, Toon Calders, Karl Tuyls, and Gerhard Weiss. *Multi-Source Entity Resolution for Genealogical Data*. Eindhoven University of Technology, The Netherlands, 2015.

[8] Gilles Geeraerts. *INFO-F-403 - Language Theory and Compiling*. Université Libre de Bruxelles, Belgium, 2013.

[9] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data quality and record linkage techniques*. Springer, 2007.

[10] Jure Leskovec, Anand Rajaraman, and Jeff Ullman. *Mining of Massive Datasets*. Stanford University, USA, 2014.

[11] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *LingvisticæInvestigationes*, 30(1):3–26, 2007.

[12] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[13] Alexander Panchenko, Dmitry Babaev, and Sergei Obiedkov. *Large-Scale Parallel Matching of Social Network Profiles*. TU Darmstadt, FG Language Technology, Darmstadt, Germany, 2015.

[14] Marijn Schraagen. *Aspects of record linkage*. PhD thesis, Universiteit Leiden, 2014.

[15] Marijn Schraagen. *Complete Coverage for Approximate String Matching in Record Linkage Using Bit Vectors*. Universiteit Leiden, The Netherlands, 2015.

[16] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Addison Wesley, 2005.

[17] Richard Webber. Quality control in the recording and matching of personal names. *Journal of Direct, Data and Digital Marketing Practice*, 14(2):131–142, 2012.

[18] Wikipedia. Jaro–winkler distance, 2015.