Census Optimization Using Machine Learning Techniques

Fernando Liça
fernando.lica@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

Abstract

The objective of this dissertation is to make use of administrative data scattered between several databases and use it to improve the Portuguese Census. Using such data will reduce the time and cost necessary to perform a census possible, which may in turn allow it to happen more often and in a more reliable way. To achieve this goal a prototype was developed consisting of three components: data cleaning and normalization, indexing using standard blocking, and classification using machine learning techniques. I study several optimizations using different algorithms to increase the amount of solved conflicts and the reliability of matched pairs. The obtained results support the feasibility of the methodology and software developed for the pairing of administrative data that are now available at Statistics Portugal and shall, consequently, provide an increase in the coverage of BPR (Base da População Residente).
Keywords: Census, String Matching, Classification, Machine Learning, Blocking, Conflict Solving.

1. Introduction

A National census is one of the most important sources for statistical and socio-economic purposes. A census provides a basis for the official statistics of a country, including the statistical information about its population, ranging from degree of literacy to the number of families. This work expands and optimizes a previous solution that uses machine learning techniques to handle this information in an acceptable computational time window.

2. Background

A census is a complex task that requires meticulous planning of every aspect, both methodological and technological, which will without a doubt mean high expenses. As an example, let's take a look at Table 1, where the costs of the population census in the United States in the last century are represented Gauthier [2002]:

As we can see, these costs grow along side the population and can amount to considerably high sums of money. It's worth noting that the average cost per person is not adjusted to the inflation, in reality they should be lower, but this still gives us an idea about the cost growth, which brings us to one of the main reasons to why population census don't happen more frequently. Due to that fact, several European countries are committed to migrate from the traditional census model to a new one based on citizen's administrative registers Baffour et al. [2013], Valente [2010].

The idea behind administrative based census is that every modernized country should have at least one digital database where it's citizens are registered, so instead of sending an agent to every home in order to count how many people live there, like in the traditional model, the register-base model looks instead in these databases for data about their citizens, such as their age and current living address, for example. Ultimately the population characteristics considered are limited to those available in the registers, therefore it's necessary to combine data across different databases in order for it to become high quality data Valente [2010].

This new type of census is starting to gain followers especially amongst the northern countries. As we can see in Figure 1, in 2010, countries like Norway, Sweden, Finland and Denmark already used register based census, while countries like France, Germany, Poland and Czech Republic use an hybrid combination between registers and more traditional means like forms and interviews.

[1] http://www.genealogybranches.com/censuscosts.html

Census Year	Total Population	Census Cost	Average Cost Per Person
1910	91,972,266	$15,968,000	17.07 cents
1920	105,710,620	$25,117,000	23.76 cents
1930	122,775,046	$40,156,000	32.71 cents
1940	131,669,275	$67,527,000	51.29 cents
1950	151,325,798	$91,462,000	60.44 cents
1960	179,323,175	$127,934,000	71.34 cents
1970	203,302,031	$247,653,000	1.22 dollars
1980	226,542,199	$1,078,488,000	4.76 dollars
1990	248,718,301	$2,492,830,000	10.02 dollars
2000	281,421,906	$4.5 Billion	15.99 dollars
2010	308,745,538	$13 Billion	42.11 dollars

Table 1: United States Population Census Costs In The Last Century[1]

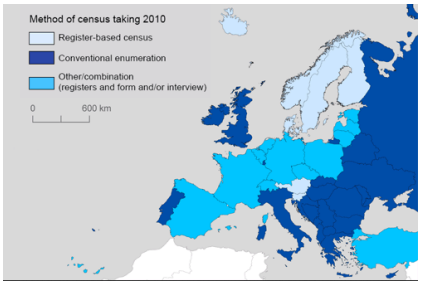

Figure 1: European Countries Census Type in 2010.[2]

3. Methodology

The applied methodology started by trying to understand the problem and the work environment with the INE staff at its headquarters in Alameda. I signed a confidential agreement due to the access of sensitive data which meant I could not work remotely and was limited to work directly at INE. I was provided a special laptop with all the tools used in the previous project. The pre-installed tools where: Oracle, Python language with libraries such as Numpy and Scikit Learn, X-Ming to give me a graphical interface to the server, which allow me to navigate the project folders like I would in the windows explorer and Putty to make the connection between the server, where all the databases where stored, and my local work space. I began by accessing the files to learn how the previous project was organized and performing a few queries to create sample tables with approximately a hundred records, to check if I had the necessary privileges. While everything was set up, I started planning what tools and algorithms I should test, and decided that I would experiment with Levenshtein, Jaccard, Jaro-Winkler, Hamming and Dice for string matching and Logistic Regression, Support Vector Machines, Decision Trees, Naive Bayes and Random Forest to classify the match candidates. Like it was previously discussed with INE, I would begin by preforming tests with the 2015 data in order to replicate the results achieved by Silva [2017] and Velho [2017], and only after finishing these tests would I then move to the 2016 data. During this period, I also concluded that I should re-test the generated model with a different amount of folds to determine if the current 2-fold validation is a good choice. Several alternatives to the blocking key were also tested, yet the results where negative in both cases, increasing the time required to run the process (the time difference was not measured because these changes meant a jump from a few minutes to at least 3 hours in execution time, at which point I terminated the experience). Afterwards I calculated the accuracy using different combinations of the algorithms that I previously mentioned, and the pair Jaccard - Decision Trees managed to score the highest accuracy percentage.

4. Objective

The objective of this dissertation is to improve upon the work of Silva [2017] and Velho [2017] by testing new algorithms and optimizations in a system previously developed, to allow a census based on administrative data provided by the following entities:

- Statistics Portugal (INE) [3].
- Civil Population Register (BDIC)[4].
- Tax Authority (AT) (IRS)[5].
- Informatics of Social Security Institute (IISS)[6].
- General Statistics of Education and Science (IISS)[7].

[3] www.ine.pt
[4] www.irn.mj.pt
[5] www.portaldasfinancas.gov.pt
[6] www.seg-social.pt
[7] www.dgeec.mec.pt

- General Retirement Fund (CGA)[8].
- Unemployment and Vocational Training Institute (IEFP)[9].
- Immigration and Borders Service (SEF)[10].

These optimizations consist in testing several new algorithms for the main phases of the process, these being: different string matching algorithms, while measuring their performance and new machine learning algorithms, to improve the classification phase. I also expand the previous work by testing new ways to match records, for example testting new blocking key. This improves INE's record quality by reducing the amount of inconsistencies and errors present in BPR enabling them to use these records for an administrative based census. This, in turn, will enable Portugal to perform population census more often and have a positive socio-economical impact nation wide.

5. Basic Concepts

In order to have a better understanding of the process I use to tackle this complex problem, let's first take a look at the algorithms that lie at its core. The previous system is composed of a combination of the most well-known algorithms in record linkageI. P. Fellegi [1969], the challenge is to understand which is the combination of these methodologies that will achieve the best results, in the most efficient way. But before we get to that, it is important to have a comprehensive understanding of all the basic concepts like Duplicate Records, Blocking, String Similarity, Classification and Data, Evaluation Metrics, Conflict Solving and Record Linkage itself, that serves as the basis for this dissertation.

5.1. Record Linkage

Record Linkage, as the name indicates, is the task of finding two records in two different databases that represent the same real world entity. It may be the case that the records in each database are different yet they represent the same entity or, conversely, they may be equal but actually represent different entities. This can happen due to errors or due to the fact that there aren't sufficient characteristics included in the records I. P. Fellegi [1969].

When two different representations (records) of a real world entity exist due to these records containing errors, these different records are called Duplicates. When joining data provided by several different sources, duplicates are bound to happen. Cleaning and normalizing data might help to avoid this problem to a certain extent, but some errors will persist, hence the necessity of having a matching generator to do this automatically.

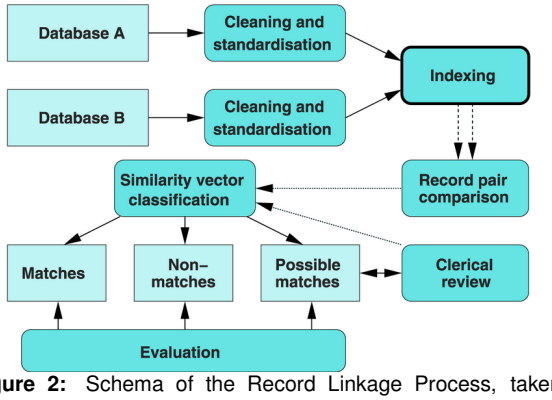

Figure 2: Schema of the Record Linkage Process, taken from Christen [2012b].

In Figure 2 we can see the scheme of the record linkage process. It begins by taking the data from information sources, in this case Database A and Database B, which goes trough the *Cleaning and Standardization* phase where the raw data is modified, replaced or even deleted if it's inconsistent or incorrect, turning it into reliable data. In addition, the data will also be normalized, to ensure that it has the same consistent format across all databases.

Next we have the *Indexing* phase where the records, that are going to be paired and compared are selected. This step is necessary due to the high quantity of records. Let's say we have Relation A and Relation B each with 1.000 records. To obtain the matching candidates of these two relations we would need to preform 1.000 * 1.000 = 1.000.000 comparisons. If we consider the fact that the task I'm trying to accomplish has millions of records, it would be computationally impracticable to preform every single comparison. To speed up this process, Blocking techniques are used to reduce the number of comparisons required. This will be further explained in Section 5.2.

In the *Record Pair Comparison* phase we take the records previously selected in the indexing phase and compare them using string matching techniques further explained in Section 5.3.

The *Similarity Vector Classification* phase classifies the records as Matches, Non-matches or Possible matches using similarity scores, and in the *Clerical Review* phase the possible matches can be labeled as Matches or Non-Matches by an expert.

Finally in the *Evaluation* phase the retrieved results will go through an evaluation so that the parameters can be further adjusted to obtain the best

[8] www.cga.pt
[9] www.iefp.pt
[10] www.sef.pt

possible result.

5.2. Blocking

The *Traditional Standard Blocking* technique is a widely used way to speed up record matching since the 60's I. P. Fellegi [1969]. It consists in generating a blocking key for each record, grouping together similar records in blocks and comparing only the records that are inside these blocks. Let's us take a look at an example in Table 2.

As we can see there are a few inconsistencies. The first in the address field where St. António is a abbreviation of Santo António and the second in the last name field where there are two different values, possibly due to an error while inserting the record in the database.

Let's now assume our blocking criteria, which essentially is the rule that allows the algorithm to determine how the blocking key is formed, is a combination of the last name and birth date fields. This will originate the following blocking keys represented in Table 3.

For the first two records, the difference is in the address field. Due to the fact that this particular field is not part of our blocking criteria both records will end up having the same blocking key. The same cannot be said for the two last two records, since the last name field is part of our blocking criteria therefore it will create two different blocking keys for each one, as we can see in Table 4.

The *Sorted Neighbor Blocking* technique starts by sorting the records by their blocking key values Christen [2012b]. A window of a fixed size $w > 1$ is then moved over the sorted records and the candidate record pairs are generated from the records inside the window. This technique can be implemented in two different ways:

Trough the *Sorted Array-Based Approach*, that inserts the blocking key values into an array of size n_A + n_B with n_A and n_B representing the number of records in Database A and Database B. The window is then moved over this sorted array generating candidate record pairs from all records in the current window that should have (n_A + n_B + w + 1) positions, the number of candidate record pairs generated in the window's first position is $\frac{n_A * n_B}{(n_A+n_B)^2} * w^2$ while the number of candidate record pair generated by the remaining positions is $\frac{n_A * n_B}{(n_A+n_B)^2} * 2(n_A + n_B - w)(w-1)$. The total number of unique candidate record pairs is given by the following Equation 1

$$CP = \frac{n_A * n_B}{(n_A + n_B)^2}(w^2 + 2(n_A + n_B - w)(w-1)) \quad (1)$$

Its important to notice that due to the fact that the windows size is fixed in this approach the number of candidate record pairs generated is independent of the frequency distribution of the BKV, and only depends upon the window size w and the size(s) of the database(s) Christen [2012b].

Sorted neighbor blocking can also be implemented trough the *Inverted Index-Based Approach* that, instead of inserting the BKV in a sorted array, it stores the blocking key values in an inverted index[11]. The window then moves over these values and candidate record pairs are formed from all records in the corresponding index lists. The number of windows positions is given by $(b - w + 1)$ where b is the number of different blocking key values. Considering these have a uniform distribution each inverted index list will contain $\frac{n_A}{b} + \frac{n_B}{b}$ record identifiers. The number of candidate record pairs is given by the following Equation:

$$CP = \frac{n_A * n_B}{b^2}(w^2 + (b - w)(2w - 1)) \quad (2)$$

5.3. String Similarity

Records are composed by several fields like name, age, sex, address etc, so in order to compare two of them we need to compare the strings in both records for each field. The String Similarity metric, as its name suggests, measures the similarity between two strings. The closest the strings are to each other, the higher their similarity is.

To compare these strings we'll use String Similarity algorithms. There are several algorithms to calculate this similarity but we're only going to talk about a few of the most well known amongst them.

5.3.1 Jaccard

The Jaccard similarity is one of the simplest algorithms we can apply to the problem. The idea behind it is the following:

Let's take a string $x = sunday$ and a string $y = monday$

First we split each string in q-grams, which are a contiguous sequence of q characters of a string. For example, a bigram (q=2) creates a window of two characters that will slide over the string, while a trigram (q=3) creates a window of three characters.

For this algorithm we will start by dividing the string x and y in bigrams to obtain the following

[11] www.dcs.bbk.ac.uk/~dell/teaching/cc/book/ditp/ditp_ch4.pdf

First Name	Last Name	Address	Birth Date
Pedro	Gonçalves	Santo António	02-04-78
Pedro	Gonçalves	St. António	02-04-78
José	Oliveira	Nazaré	18-06-84
José	Olivais	Nazaré	18-06-84

Table 2: Example of records.

Blocking Key
Gonçalves020478
Gonçalves020478
Oliveira180684
Olivais180684

Table 3: Blocking Keys for the records of Table 2.

results:

x = [su],[un],[nd],[da],[ay]
y = [mo],[on],[nd],[da],[ay]

The Jaccard similarity is defined by the equation 3:

$$Jaccard(x, y) = \frac{|x \cup y|}{|x \cap y|} \quad (3)$$

In equation 3, $|x \cup y|$ represents the number of common q-grams between x and y, while $|x \cap y|$ represents the size of the union between the q-grams of x and y.

So for this specific example, between string x and y we would obtain a similarity of:

$$Jaccard(x, y) = \frac{3}{7} = 0.428$$

5.4. Classification

I will make use of Supervised Learning techniques to classify if two records are either a Match or a Non-Match. Thus I will make a brief, conceptual explanation of what is Supervised Learning and what techniques we can use.

Let's start with something simple, we have two classes: Match and Non-Match and our features shall be First Name, Last Name and Address as shown in Tables 5 and 6.Then we are given two sets of record pairs, in the first one the pairs are already labeled so we already know its class, either a Match or Non-Match. This is going to be our Training Set. In the second set, the pairs are not labeled and we have to find a way to classify them, these will serve as Test Set. This is a classic supervised learning problem, given we already have a set of classified record pairs as our training set, we can now submit it to a supervised algorithm to obtain a model. We can then apply that model to our unlabeled record pair set (Test Set) in order to classify it.

There are several possible supervised learning algorithm but we are going to be focused on Decision TreesQuinlan [1986] of which I will give a more detailed explanation.

5.4.1 Decision Trees

Decision Trees are a supervised learning method used for classification and regression, that can help to support decisions and predict outcomes. The model is simple to understand by humans and performs well with large data sets, while remaining fast. In a decision tree structure there are three parts: non-leaf nodes, branches and leaf nodes (terminals nodes).

The root (the topmost node) and the others non-leaf nodes work as a test on an feature. From these nodes there are branches that represent the results of the test, which could follow to another non-leaf node or a terminal node. The terminal nodes are the class labels. To clarify, given a datapoint X, its feature values are used on the decision tree and a path is traced until reaching a class prediction.

The order of the features used along the way its decided by how much information we gain from each feature, the information gain can be obtained by the formula 4:

$$Gain(T, X) = Entropy(T) + Entropy(T, X) \quad (4)$$

with T representing our class and X representing a feature. Further explanation can be found in the work of Mitchell [1997] and here in this website [12]

6. Related Work

In this section we will talk about relevant previous works related to the optimization of census using machine learning techniques. To make it clearer I will split it into three sections: Duplicate Detection,

[11] www.chunml.github.io/ChunML.github.io/tutorial/Decision-Tree/

[12] www.saedsayad.com/decision_tree.htm

Gonçalves020478			
First Name	**Last Name**	**Address**	**Birth Date**
Pedro	Gonçalves	Santo António	02-04-78
Pedro	Gonçalves	St. António	02-04-78

Table 4: Example of records with the same blocking key.

Database	First Name	Last Name	Address
A	Miguel	Marques	Póvoa de Santa Iria
B	Miguel	Marques	Póvoa de Santa Iria

Table 5: Match Pair

Database	First Name	Last Name	Address
A	Miguel	Marques	Póvoa de Santa Iria
B	Pedro	Melo	Santarém

Table 6: Non-Match Pair

which will be further divided into Efficiency and Effectiveness, works related to Census using administrative data performed in other countries.

6.1. Duplicate Detection

The Duplicate Detection problem can be divided into actually finding the existent duplicates (Effectiveness) and into how quickly and cheaply, resource and time wise, we can find them (Efficiency).

6.1.1 Efficiency

Efficiency is obviously important when working with high quantities of data. The blocking step, explained in Section 5.2, is a good example of how reduce the amount of records that need to be paired up and drastically reducing the amount of time and resources required to run this project. Finding the optimal blocking key is, however, a big challenge. Hernández and Stolfo [1998] proposed a technique that consisted in combining the results of several executions of the sorted neighborhood blocking algorithm (Multipass approach) with a different blocking key in each execution.

Michelson and Knoblock [2006] state that the effectiveness of a multi-pass approach depends on the chosen attributes and used methods, according to them the fundamental issue of blocking research is to find what are the best attributes and what are the methods that should be applied to those attributes. To solve these issues they propose a machine learning approach. By automatically learning rules (a conjunction of attributes) using the SCA proposed by Mitchell [1997], the effective schema is then evaluated by learning rules that cover a sufficient number of True Positives while minimizing the number of False Positives, using the RR to quantify how well the current blocking scheme minimizes the number of candidates and the PC to measure the coverage of true positives:

$$RR = 1 - \frac{C}{N} \quad (5) \qquad PC = \frac{S_m}{N_m} \quad (6)$$

Where C represents the number of candidate matches, N the size of the cross product between both data sets, S_m the number of true matches in the candidate set and N_m the number of matches in the entire dataset.

Li et al. [2006] propose using a string map function, that will convert the blocking key values to a multidimensional Euclidean space, they then select a threshold and for each pair that is bellow that threshold, preform a multidimensional similarity join that will form clusters of records, the records on these cluster will then be compared with a similarity metric.

6.1.2 Effectiveness

Much like efficiency, we should always have effectiveness into consideration, and with that idea in mind TAILOR (**R**ecord **Li**nk**a**ge **T**oolbox) was developed by Elfeky et al. [2002]. With TAILOR users can build their own record linkage models by tuning system parameters and by plugging in in-house developed and public domain tools. These authors also proposed three models, one based on supervised learning (Induction Record Linkage Model), one based on unsupervised learning (Clustering Record Linkage Model) and the third one based on

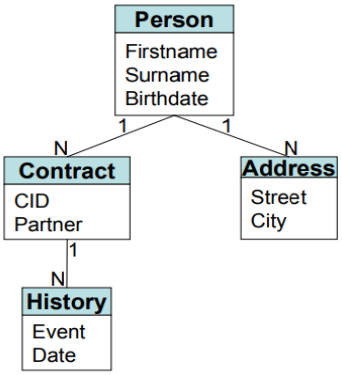

Figure 3: UML diagram of entities and relationships in Weis et al.(2008).

both supervised and unsupervised learning (Hybrid Record Linkage Model). Results show that all of these models obtain better accuracy and completeness than the probabilistic model.

According to Christen [2012a], the simplest way to classify two candidate record pairs into matches and non-matches is to sum all similarity scores of the previously compared fields of two different records into a single total similarity metric (the author calls it SimSum) and then compare this values with a similarity threshold to decide into which class each pair belongs. With two classes (Match and Non-Match) only a single threshold is required, if the SimSum values is above the threshold its a Match, if its bellow its a Non-Match. When we have three classes (Match, Non-Match and Possible Match) we need two thresholds (upper and lower), if the SimSum value is bellow the lower threshold is a non-match, if its above the upper threshold, its a match and if its between the thresholds its a potential match that needs to be reviewed by an specialist or a experienced user. Yet if we simply sum all of the fields they all have the same importance therefore if there are fields more important than others we can attribute different weights to each of them.

Weis et al. [2008] presented the DogmatiX prototype, which was designed to detect duplicates in hierarchical XML data. He also tells us how this prototype was applied on a large scale industrial relational database whose main business line is to store and retrieve credit histories of over 60 million individuals.

In short, descriptions for a given candidate type are semi-automatically selected using heuristics and conditions based on an XML Schema (Figure 3). To detect duplicates, only the elements that are closer to the candidate in the schema hierarchy like name or address are considered because they are more useful, while elements that are far from the element are less likely to clearly distinguish person candidates. Furthermore the heuristic selection is still refined by conditions like pruning XML elements based on data type, content model and cardinality.

7. Proposal

My objective was to develop on the project originally created by Silva [2017] and Velho [2017] represented in Figure 4, more specifically on the Blocking, Training and Classification Component .

I will begin by making a brief description of my colleagues' approach to the problem and then following up with my own proposal to improve upon their work.

To achieve an acceptable level of reliability, a cross-reference between databases is necessary to make sure that there are no mistakes and each record belongs exclusively to one person.

Then, to make the matching process computationally feasible Silva [2017] and Velho [2017] used the standard blocking technique to group similar records using a blocking key. Yet, by using this technique with a specific blocking key, some potential matches may be excluded if there are errors on the chosen fields (First Name, Birth Date).

To compare the records within these groups, I used the Edit Distance algorithm, given its simplicity. The feature extraction chosen method was also very simplistic: they calculate the string distance between the different fields, and use these distances as features to train the model that will classify the unmatched records using the Logistic Regression Classifier.

After obtaining a dataset with record pairs classified by the previously created model, this model takes into consideration all of the similarities between every field for each record pair, and calculates the probability of that pair being a match. If the probability is above 50%, it is considered a match, otherwise a non-match, and the pairs classified as non-matches are discarded, as we are not interested in them. Afterwards, the process checks if the obtained records exist in the BPR, because we only want the new ones.

But there is still an issue that needs to be addressed, this process is not flawless, as we may have pairs that the model considered to be a match and do not exist in the BPR, but in reality that is not the case. Let's look at Table 7 as an example.

As we can see, the model considers that the Record 1 from Database A matches the Records 2, 4 and 5 from Database B, but in reality only one of these matches can be considered true because each record can only have a single match between a specific pair of databases. To solve this problem,

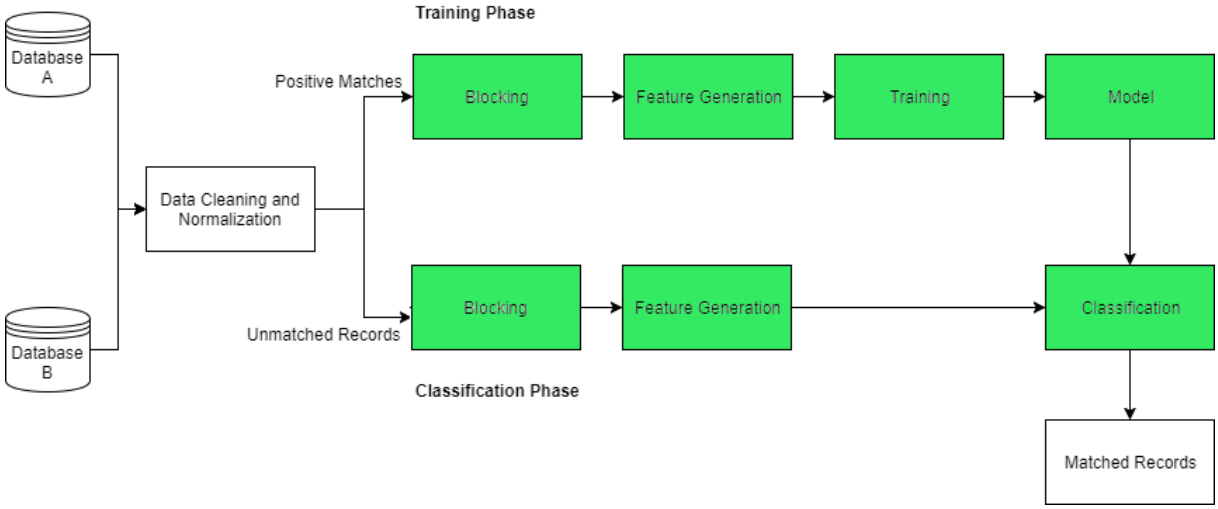

Figure 4: Record Linkage System Architecture.

Database A	Database B	Probability of Being a Match (%)
Record 1	Record 2	98%
Record 1	Record 4	67%
Record 1	Record 5	58%

Table 7: Possible Matches not yet present in BPR.

the process has a final step where it will only select the match with the highest probability.

Moving on to my own proposal, the following list presents what would be my main objectives to improve upon this work:

- Replicate the matching with the 2015 data to ensure that the previous code was working and to get familiarized with the project.
- Replicate the matching with the 2016 data to ensure it was compatible with the project and to create the necessary tables to move to the next stage.
- Attempt to reduce the process' running time by testing different blocking keys.
- Address the overfitting problem by performing cross-validation with a different number of folds.
- Test a new string matching algorithm apart from the previously used Edit Distance, like Jaccard or Jaro-Winkler.
- Test a new machine learning algorithm apart from the previously used Logistic Regression, like Support Vector Machines or Decision Trees.
- Measure the improvements obtained through the evaluation metrics mentioned in section 8.

To accomplish these goals, I would be intervening mainly on the colored components from Figure 4, but due to several issues with the previous code, I eventually work in all of them.

8. Evaluation

After cleaning the data and classifying it, we need to evaluate the quality of the returned matches. In a project that works with sensitive information, it's imperative that data coming from the sources has a minimum level of quality and reliability.

When declaring two records as either a Match or Non-Match, there are four different types of possible results: True Positives(TP), False Positives(FP), True Negatives(TN), False Negative(FN). In Table 8 we can see a schematic representation, making it easier to understand. P represents our prediction, C represent their real classification, M represents a Match and N represents a Non-Match.

	P = M	**P = N**
C = M	TP	FN
C = N	FP	TN

Table 8: Classification possible outcomes.

True Positives represent the pairs that we predicted to be a Match and are actually a Match, False Positives the pairs we predicted to be

Matches but are in fact a Non-Match, False Negatives stand for the pairs that we predicted to be Non-Matches but are Matches and finally, True Positives represent the pairs that we predicted to be Non-Matches and are actually Non-Matches. Using these definitions, I will now present the most commonly used classification evaluation measures.

8.1. Evaluation Metrics

In essence, **Precision** represents the number of actual matches (True Matches) amongst all the returned matches. To capture the large number of negative examples on the algorithm's performance, the **Precision** will compare the false positives to true positives Davis and Goadrich [2006].

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

Recall represents the number of returned matches from all the actual matches and it is presented in Equation 8 .

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

Accuracy is the ratio between the True results (True Positives + True Negatives) and everything else returned (True Positive + True Negative + False Positive + False Negative). This ratio is given by in Equation 8.1

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

9. Result

In this section I will present the results obtained during the six moths I've worked with INE.

10. Comparing Results

In the following, Table 9 I will present the figures that represent the number of new matches, referenced in Section 7 for the year 2015.

As it is shown in Table 9, some of the results are very similar to the ones obtained by my colleagues, while others are considerably different. This happened due to the fact that I used SQL scripts created by myself, which produced different data and in turn, different results as expected.

I then tested the possibility of using different blocking keys as an alternative to the one currently in use (First Name, Year of Birth, Month of Birth, Day of Birth), with the following combinations:

	Previous Results	My Results
BDIC AT	244.903	284.985
BDIC IISS	47.836	55.631
BDIC EDUC	51.138	36.184
BDIC IEFP	11.974	21.311
BDIC CGA	60.545	60.160
IISS SEF	30.120	29.950
AT SEF	52.177	52.142
SEF EDUC	12.796	11.457

Table 9: Differences between the amount of new matches found by Silva [2017] and Velho [2017] and the amount of new matches I found for the year 2015.

- Last Name, Year of Birth, Month of Birth, Day of Birth.
- Last Name, Postal Code.
- First Name, Last Name.

Unfortunately, this would either generate very few candidates for classification, due to a very strict key or many null entries, for example the postal code field, meaning that only a few thousands of records could be used, or it would generate too many candidates due to a less strict key, which would make the process extremely time consuming and simply unfeasible.

Similarly, when I tested the training component with a different number of folds, it would simply increase the time necessary to finish the process, without any noteworthy impact on the results, so I decided I would continue to use the 2-fold approach.

Afterwards, I started creating the tables with the data from the year 2016. Given that I obtained different tables than Silva [2017] and Velho [2017] using the 2015 data, it wouldn't make sense to compare the tables created from the 2016 data, using my scripts, to the tables from 2015, that Silva [2017] and Velho [2017] used, and expect an improvement, because they are not equivalent. Therefore, from this point forward, I used my 2015 results as a baseline.

	2015 Results	2016 Results
BDIC AT	284.985	9.055
BDIC IISS	55.631	64.702
BDIC EDUC	36.184	21.644
BDIC IEFP	21.311	12.165
BDIC CGA	60.160	45.329
IISS SEF	29.950	27.442
AT SEF	52.142	58.604
SEF EDUC	11.457	8.802

Table 10: Differences between the amount of new matches found with data from 2015 and new matches found with data from 2016.

As expected, the amount of new matches found with data from 2016 are lower than the ones previously achieved. This is due to the fact that the new records found by Silva [2017] and Velho [2017] were incorporated into the BPR matrix, therefore, the majority of the new records found by me already exist in BPR and are no longer considered ”new”. Yet, there are some exceptions where I obtain higher results than my colleagues previously did. These exceptions happened because the SQL scripts used by me had some differences (as I was never able to fully replicate the code my colleagues used), leading to different results.

11. String Matching and Classification Algorithms

Originally, Silva [2017] and Velho [2017] used simple algorithms for the matching of strings in the existing records (ex: Name, Address) and for the classification of record pairs the chosen algorithms were Edit Distance and Logistic Regression, due to their simplicity. Yet, more complex and interesting algorithms exist, and one of the goals for this thesis was to test these alternatives and find out which work better.

For the string similarity calculation, I tested the following algorithms: Jaccard, Jaro-Winkler, Hamming and Dice. The results obtained are shown in Table 11 that contains the accuracy obtained by each algorithm.

It is important to note that these results tell us how accurate a certain classification algorithm is while using the features (in this case the similarities) provided by the string matching algorithms. The chosen classification algorithm was the Logistic Regression, again, for its simplicity, as what I was really trying to understand at this stage was what string matching algorithm provides the best features. Ultimately, the Jaccard algorithm offered slightly better results than the rest.

To perform the classification of records, the following algorithms were tested: Decision Trees, Support Vector Machines, Naive Bayes and Random Forest. The results are presented in Table 12.

As we can see in Table 12, there is no clear margin between the results, as they all have a pretty similar accuracy. However, the combination of Jaccard for string matching and Decision Trees for classification does have a slightly higher accuracy for the majority of the database pairings.

In Table 13, we can see the results from running the process with both the previously used algorithms and with the newly used ones, there is an increase in the number of new matches found (matches that did not previously exist in BPR) for the pairs BDIC-AT, BDIC-IISS, BDIC-CGA and SEF-EDUC yet, for BDIC-EDUC, BDIC-IEFP, IISS-SEF and AT-SEF, the amount of new matches found is lower. Given the fact that the new solution (using Jaccard and Decision Trees) obtained a slight increase in accuracy, a higher amount of new matches should be expected. Consequently, these results need to be further examined by INE to determine the problem. There are a few plausible causes that may explain such behaviour, for example: when I calculate the New Matches for Previous Solution, the new found matches are not added to BPR (adding new matches to BPR is out of this project scope). Therefore, many ”New” matches found by both solutions may overlap.

To put it simply, let’s take the pair BDIC AT as an example, the previous solution obtained 9.055 new matches and the new solution obtained 9.208, this could mean that 9.055 matches are common to both solutions which would result in a 153 matches difference. But the two solutions might not have any new match in common, which would mean an approximate 9.000 matches difference between them. Could this imply that one of the solutions is completely wrong ? And, if that is indeed the case, which solution is correct ?

12. Conclusions

The objective of this work was to obtain a model capable of taking records from several sources and preform a multi-database cross reference in order to better detect duplicates, errors and inconsistencies in a computational acceptable time. In this section, I will describe the decisions I’ve made along the way whilst providing my opinion about it and making recommendations for the future.

First, I needed to replicate the results obtained in the previous work with the 2015 data to make sure everything was set to move to the 2016 data. This was only partially possible, given that some critical elements from the previous work were missing. After achieving results as similar to the previous work as possible, it was time to test the 2016 data, which as expected, yielded lower results in terms of new matches found in BPR compared to 2015, given that many of them had already been found

	Levenshtein	Jaccard	Hamming	Dice	Jaro-Winkler
SEF EDUC	0.9129	0.9132	0.9105	**0.9145**	0.9105
AT SEF	0.9895	**0.9928**	0.9882	0.9853	0.9781
IISS SEF	0.9799	**0.9836**	0.9793	0.9772	0.9749
BDIC CGA	**0.9761**	0.9708	0.9757	0.9690	0.9710
BDIC IEFP	0.9858	**0.9881**	0.9857	0.9841	0.9841
BDIC EDUC	0.9813	**0.9839**	0.9815	0.9769	0.9776
BDIC IISS	0.9984	**0.9989**	0.9984	0.9759	0.9958
BDIC AT	**0.9991**	0.9990	**0.9991**	0.9851	0.9950

Table 11: Accuracy for the string matching algorithms (using Log. Regression for classification).

	Log. Regression	Dec. Trees	SVM	Naive Bayes	Random Forest
SEF EDUC	0.9132	**0.9251**	0.9153	0.9068	0.8988
AT SEF	**0.9928**	0.9927	0.9927	0.9904	0.9887
IISS SEF	0.9836	0.9850	0.9790	0.9808	**0.9851**
BDIC CGA	0.9708	0.9758	**0.9883**	0.9516	0.9516
BDIC IEFP	0.9881	0.9882	**0.9883**	0.9812	0.9815
BDIC EDUC	0.9839	**0.9856**	0.9831	0.9823	0.9788
BDIC IISS	0.9989	**0.9992**	0.9987	0.9940	0.9967
BDIC AT	0.9990	**0.9993**	0.9990	0.9979	0.9975

Table 12: Accuracy for the Classification Algorithms (using Jaccard distance for string matching).

Data Source	Unmatched Records (a)	Previous Solution			New Solution		
		PM (b)	MIM (c)	NM (d)	PM (b)	MIM (c)	NM (d)
BDIC	8.843.729	116.956	107901	9.055	117.339	108.131	9.208
AT	4.041.101						
BDIC	8.023.181	663.642	598.940	64.702	711.018	627.873	83.145
IISS	1.125.575						
BDIC	12.592.913	32.323	10.679	21.644	31.368	10.135	21.233
EDUC	43.307						
BDIC	13.296.719	31.368	19.203	12.165	18.522	12.079	6.443
IEFP	36.564						
BDIC	13.053.206	255.476	228.034	27.442	267.729	240.479	27.250
CGA	196.135						
IISS	553.981	34.582	25.780	8.802	33.952	18.350	15.602
SEF	266.241						
AT	8.909.159	130.114	71.510	58.604	77.923	12.323	55.155
SEF	235.945						
SEF	385.191	15.332	6.530	8 802	28.798	13.196	15.602
EDUC	18.839						

Table 13: Comparison between the results from previous solution (with Levenshtein - Logistic Regression) and the new solution (with Jaccard - Decisions Trees).
(a) Number of records to match per database.
(b) Positive Matches.
(c) Matches that already exist in the matrix.
(d) New Matches.

and integrated to BPR.

Once all the data from 2015 and 2016 had been classified, I moved to the next step: testing the same process with new algorithms, with the goal of possibly finding new matches and improving performance.

Four new string matching algorithms (Jaccard, Jaro-Winkler, Hamming and Dice) and four new classification algorithms (Support Vector Machines, Decision Trees, Naive Bayes and Random Forest) were implemented, along with a few interface improvements to make it easier for the user to select the intended options. The reason behind the choice of algorithms was straightforward: I wanted to use well known algorithms, as it made it easier for people with some experience in the field to know what was happening behind the classification phase, and because it was also easier to find single libraries that included all of the required algorithms.

After having every algorithm implemented, I decided that I should test the process with the 2016 data, as this is where new matches where more

likely to be found first, to calculate the accuracy of every algorithm implemented (both similarity and classification algorithms) to determine the best option.

The results showed that the Levenshtein algorithm and the Decision Trees algorithm for the string similarity and classification components respectively were the best choice for most table pairings. Jaccard distance (using Log. Regression for the classification) managed to achieve an higher accuracy in 75% of the database pairs, while the Decision Trees classifier (using Jaccard for string matching) achieved a higher accuracy in 50% of the database pairs.

Despite having achieved better accuracy with the new algorithms, this doesn't always translate to more records found. Some of the smaller discrepancies need to be further examined by INE to determine the causes.

By the end of the project, I documented everything I did and created a read-me (see Annex) to ensure the next person to work in this project makes the transition as smoothly as possible.

13. Future Work

I will dedicate this section to talk about further improvements and the future possibilities for this project. One way to optimize this process is by having more than a single blocking method. This will allow the reduction in the exclusion of records from the matching process due to inconsistencies, for example, for this work I use the Last Name and the Birth Date (day, month, year) for the blocking process however, when I try to match two records, even if they are a positive match and in fact represent the same person but one of the records has a error in one of this fields, this match will be excluded.

There are also plenty of alternative algorithms for both the string matching and classification phase that could provide better results. This process is not restricted to existing ones, given that new algorithms might be developed even as I write this thesis. With this issue in mind, I made the implementation of new algorithms as modular and simple as possible, and the fact that this project is written using Python is also helpful, given the amount of exiting libraries.

The interface could also be further developed to present the information in a more user friendly format, generating graphics and statistics. Another aspect that could be improved is the ability to take the names of the databases in any order (BDIC_AT or AT_BDIC) because, currently, the project will only recognize one of them, and this would be particularly useful for the INE staff, given the amount of data they have to work with, making the work faster and less prone to errors.

INE also has plans to adapt the process to match addresses in the future, given that the address field is usually where more errors occur. By using this process, INE hopes to find duplicates and correct mistakes.

References

B. Baffour, T. King, and P. Valente. The modern census: Evolution, examples and evaluation. *International Statistical Review*, 81(3):407–425, 2013.

P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012a.

P. Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. 24 (9):1537–1555, 2012b.

J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid. TAILOR: A record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 17–28. IEEE, 2002.

J. C. Gauthier. *Measuring America: The Decennial Censuses From 1790-2000*, volume 5. 2002.

M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, 2(1):9–37, 1998.

A. B. S. I. P. Fellegi. A Theory for Record Linkage. 64(328):1183, 1969.

C. Li, L. Jin, and S. Mehrotra. Supporting efficient record linkage for large data sets using mapping techniques. *World Wide Web*, 9(4):557–584, 2006.

M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

T. M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

R. Silva. *Matching Census Data Records*. PhD thesis, Instituto Superior Técnico, 2017.

P. Valente. Census taking in Europe: how are populations counted in 2010? *Population & Societies*, (467):1, 2010.

L. Velho. *Emparelhamento de dados Censitarios*. PhD thesis, Instituto Superior Técnico, 2017.

M. Weis, F. Naumann, U. Jehle, J. Lufter, and H. Schuster. Industry-scale duplicate detection. *Proceedings of the VLDB Endowment*, 1 (2):1253–1264, 2008.

Appendices

Census Optimization Using Machine Learning

Required Libraries

To make the project work the following Libraries are required:

- cx_Oracle
- nltk
- sklearn
- numpy
- textdistance
- pandas

Instructions

To run the project take the following steps:

1. Create normalized tables:
   - Go to the queries folder, then select a new folder that corresponds to the tables you want to pair up.
   - Run the script 1_"name_of_table".sql for each of the tables you want to normalize (for example: 1_AT).
   - As an alternative you can go to the prototype folder ans run the script_main_test.py and select the option normalization (2 times, one for each table).

2. Training:
   - In the same folder (corresponding to the tables you want to pair up) run the script 3_"name_of_pair".sql (ex:3_cand_pos_bdic_at) (script 2 is included in script 3) or alternatively go to the prototype folder and run the file script_main_test.py and select training.

3. Classification:
   - In the same folder run the script 4_"table1"_norm_nm_"table2".sql, and then the script 5_cand_nm_"table1"_"table2"_"year".sql.
   - In Oracle extract the tables cand_nm_"table1_table2"_"year" and cand_pos_"table1_table2"_"year", while extracting the table cand_nm_"table1_table2"_"year", extract only the columns that come after nome_3pri excluding rec_id, id_null, identificador_tabela1 e identificador_tabela2),
     table cand_pos_"table1_table2"_"year", extract only the columns that come after nome_3pri excluding rec_id.
   - Go to training folder and run the file similarities.py (2 times, one for each table) choose the option non-classified for the cand_nm_"table1_table2"_"year" table and the option classified for the cand_pos_"table1_table2"_"year" table.
   - Run the file train_model.py and type the year and the names for the pair for which you want to generate a model.
   - Run the file match_records.py and type the year and the names for the pair you want to train.

- Use Oracle to import the resulting matches_"tabela1_tabela2"_"ano".csv file (you will need to change the size of the size/precision field corresponding to the column rec_id to 9.
- Return to the folder "queries" and run the script 6_class_"table1_table2"_"year".sql.
- Run the script 7_class_"table1"_"table2"_n_matriz_"year".sql.
- Run the script 8_class_prob_max_"table1"_"table2"_"year".sql.
- Run the scripts contradicao_"table1"_"table2"_"year".sql and incerteza_"table1"_"table2"_"year".sql (required to perform monitoring).

4. Monitoring:

- Go to the monitoring folder and run the file monitorização.py.
- To obtain information about table normalization select option 1.
- To obtain information about record percentage select option 2.
- To obtain information about record classification select option 3.
- To obtain information about contradiction and uncertainty percentage select option 4.