A BENCHMARK STUDY OF CLUSTERING BASED RECORD LINKAGE METHODS

by
Kerem UĞURLU

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabanci University
Spring 2009

A BENCHMARK STUDY OF CLUSTERING BASED RECORD LINKAGE METHODS

APPROVED BY:

Yrd. Doç. Dr. Yücel Saygın (Dissertation Supervisor)	……………………..
Doç. Dr. Albert Levi	……………………..
Doç. Dr. Erkay Savaş	……………………..
Doç. Dr. Uğur Sezerman	……………………..
Yrd. Doç. Dr. Cemal Yılmaz	……………………..

DATE OF APPROVAL: ……………………..

Abstract

Record linkage (or record matching) tries to identify the records in datasets which represent the same entity. These entities could be people or any other entity of interest. In this study, there has been processed a benchmark of clustering algorithms used in record linkage was conducted. The reason for the interest was that with the rise of the machine learning, record linkage has been considered as a classification problem with two classes of matched and unmatched pairs. The pairs to be compared are the entries in the dataset with a possible reduction of comparisons to avoid the quadratic complexity. The reason for the need for the clustering benchmark is that the experiments are processed by assuming that the experimenter has substantial training data for the classification procedure so that he can proceed in a supervised fashion. However, this is usually not the case in real life scenarios. For that reason, in this benchmarking study, the main three clustering algorithms are applied on three different datasets which are selected with different characteristics on purpose.

Özet

Kayıt bağlama (ya da kayıt eşleştirme) veri setlerindeki aynı nesneyi kasteden kayıtları belirlemeye çalışır. Bu nesneler kişi veya ilgilenilen her hangi bir nesne olabilir. Bu çalışmada, kayıt eşleştirmelerinde kullanılan öbekleştirme algoritmalarının bir performans kıyaslaması yerine getirildi. Bu ilginin sebebi şuydu, makine öğrenmesinin yükselmesi ile kayıt eşleştirme uyan ve uymayan diye iki sınıflı bir sınıflandırma olarak düşünülmeye başladı. Karşılaştırılacak çiftler, ikinci dereceden zorluğu önlemek için olası bir karşılaştırmaların azaltılması ile veri setindeki kayıtlardır. Performans kıyaslama ihtiyacı sebebi deneylerin sınıflandırma işlemi için elde yeterince eğitme verisinin bulunması nedeniyle deneycinin denetlenen şekilde ilerleyebildiği varsayımıdır. Ancak, gerçek hayat senaryolarında durum genelde bu değildir. Bu sebeple, bu kıyaslama çalışmasında, üç ana öbekleştirme algoritması üç kasten farklı karakteristikte seçilmiş veri seti üzerinde uygulanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Record linkage is the process of identifying records belonging to the same entity from one or more data sources with possible different representations [1]. In various contexts, the problem of record linkage was described as *entity heterogeneity* [2], *entity identification* [3], *object isomerism* [4], *instance identification* [5], *merge/purge* [6], *entity reconciliation* [7], *list washing* and *data cleaning* [8]. Entities of interest may be individuals, families, households, geographic regions (can be seen in administrative census data), companies and customers (in customer relationship management). In real life, there are usually no identification numbers (like Social Security Numbers, or citizenship number) available at both data sources. This is why there is a high degree of uncertainty as to which records represent the same entity. As a result, record linkage can not be done using simple SQL join operations. In addition to that, "real world data is dirty" [9] and values in identifying fields of the corresponding entities lack a uniform format [10].

Historically, most record linkage procedure was done by human clerks in which these experts reviewed lists, obtained additional information when the missing data was the case and came up with linkage rules, i.e. linking the two entries in the dataset as two representations of the same object. The key point here is that when two records have considerable amount of information to come up with a decision whether they represent the same entity, the human can almost naturally make up for the typographical errors, abbreviations and missing data and match the two entries correctly as match or unmatch. The drawback on the other hand is that people compared to an automated process are very slow. Furthermore, if the files were large as in a census data case, the files have to be separated by several pages of print outs, consequently those matches on different printouts might not be reviewed. All work required extensive review in the past and each update of data available required a new set of training clerks [11].

On the other hand, the computer aided process does not have any of these deficits. When there are good identifiers available, computer algorithms are fast, accurate and come up with reproducible results [11]. As an example, the computer algorithms are searching the key identifiers with spelling variations or they can account for the relative frequency of combinations of identifiers much better than human beings.

The main problem with record linkage is the lack of training data for building an accurate model for classification. This is partially resolved through clustering the records into two regions Matched and Unmatched with an optional rejection region of Possibly Matched. The state of the art record linkage work uses the standard k-means algorithm which is implemented in almost any data mining toolbox. However, k-means has its drawbacks such as producing spherical clusters which may not be the best choice for generating training data for record linkage. A benchmark study on different clustering methods such as hierarchical or model based clustering on real data sets with various properties is missing. In the context of this work a benchmark study has been conducted. This study highlights the effectiveness of different clustering algorithms on datasets with different characteristics.

The following historical example [11] shows the drastic effect of computer use in record linkage. Before 1982, U.S. Census of Agriculture data were reviewed manually, and an unknown amount of duplicates remained in the datasets. In 1987, an *ad hoc* computer algorithm for classifying the pairs of entries in the dataset as match and unmatch and creating subsets for further clerical review found out that 6.6 % (396.000) of the records as duplicates and 28.9 % as possible duplicates that had to be clerically reviewed. 14.000 person hours corresponding to 75 clerks' work for three months were spent during these reviews to find out that there are an additional 450.000 duplicates which is 7.5 % of the entire census data. In 1992, algorithms based on probabilistic models were used in the census data. The software designated 12.8 % of the file as duplicates and left 19.7 % to clerical review. As to be seen from these figures, the 1992 computer procedures identified as many duplicates as in the 1987 census data with combination of clerical and computer procedures. The rates of duplicates identified by computer plus clerical reviews were 14.1 % in 1987 and 20.9 % percent in 1992. The sizes of the two data sets are comparable, since the 1987 data are multiplied by the base of 1992 data of 6 million. The 1992 procedures lasted 22 days; in contrast the 1987 procedures lasted three months.

The modern record linkage procedure has several steps. These steps of record linkage can be summarized as:

(1) *Data Preparation*

(2) Blocking

(3) Comparison

(4) Decision

(5) Evaluation

Data Preparation is an essential step in every record linkage process. The main purposes of this step are first to replace spelling variations of commonly occurring words with standard spellings using a fixed set of abbreviations or spellings and second to use certain key words found in standardization to further process the substrings [11]. After standardization, frequently appearing words implying the same object in a document with different representations are converted into one common item, so that the process does not perceive the same entity as different entities. Hence, without standardization many record pairs would be misclassified as unmatched whereas their actual status is matched.

The second step in the record linkage is the *Blocking* phase. The main aim of blocking is to reduce the number of comparisons. That is to say, blocking tries to separate the database into a set of blocks and compare the corresponding pairs in each block, separately. Naturally, it is assumed that there are no matches between different blocks. This way, we get rid of the naive nested loop approach for every record in the database. That is to say, given a dataset of N entries, the total number of comparisons would be $O(N^2)$ if we compare all entries with the remaining entries of the whole dataset. A generic blocking scheme would be to combine the first few initial letters of the attributes, like the first two letters of the surname, the initial of name and the year of birth date. Combination of the initial letters create a "blocking key", and compare only those record pairs that correspond to the same blocking key value. Details of various blocking schemes will be given in subsequent sections.

The third step is called the *Comparison* phase. For the comparison of the records we need a specific function, which has several values for the possible cases of errors and differences. The comparison function γ is defined for each attribute separately, that are common to the data sources. These attributes represent the identifying information at the records. The state of the art algorithms based on these functions are described in the subsequent sections.

The fourth step is the *Decision* phase. Based on the results of record pairs which is the output of the corresponding comparison function; the Decision phase produces two decision regions which are labeled as *Matched* and *Unmatched.* There is one more available region in the model which is so called the *rejection region*, where one cannot decide whether the pair is a match or an unmatch. This decision class is called *Possible Match*, denoted by P, where the pairs are left to clerical review for an expert. It is assumed that the expert is always able to identify the label of the pair as M or as U correctly.

The last step is *Evaluation* where the performance of the record linkage application is measured based on various metrics. Figure 1.1 [1] shows a general framework of a record linkage procedure.

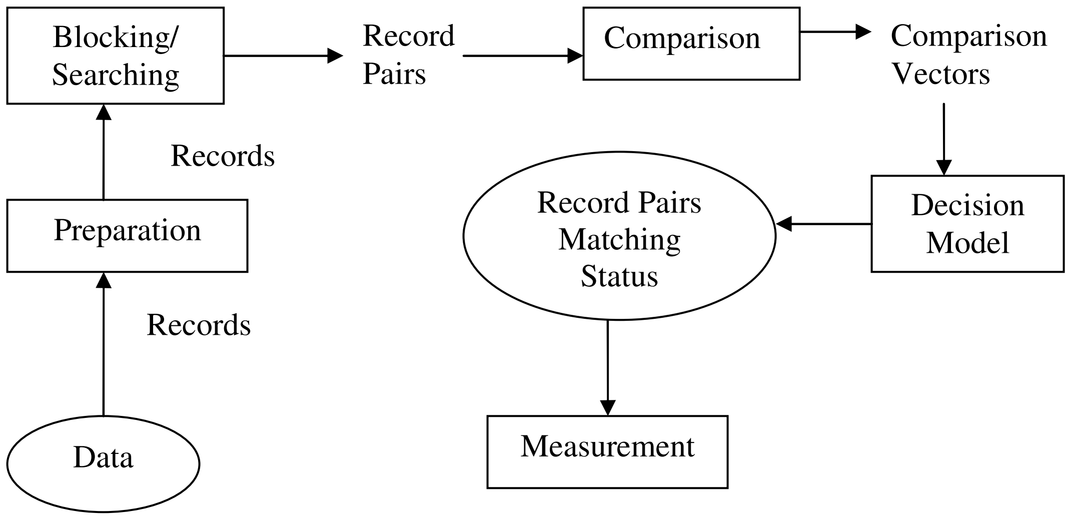

Figure 1.1

General Framework of a Record Linkage Process

Rest of the paper is organized as follows. In Section 2 we provide a formulation of the problems. In Section 3, we describe the data preparation phase. In Section 4, the idea of blocking has been introduced. In Section 5, the comparison stage with the main comparison functions are given. In Section 6, the classical supervised record linkage techniques are given. In Section 7, the clustering algorithms used in our benchmarking study are described. In Section 8, the implementation and the results are provided. Finally, in Section 9 we conclude our paper.

2. PROBLEM FORMULATION and NOTATION

We denote with $(a,b) \in A \times B$ an ordered pair of elements of the two populations A and B. The cross-product $A \times B = \{ (a,b) \mid a \in A, b \in B \}$ is the disjoint union of two sets:

$A \times B = M \cup U \wedge M \cap U = \varnothing$,

where

$M = \{ (a,b) \mid a \in A \wedge b \in B \wedge a = b \}$

$U = \{ (a,b) \mid a \in A \wedge b \in B \wedge a \neq b \}$

M denotes the matched set, and it includes all the elements which are common to ***A*** and ***B*,** respectively. The set of nonmatched pairs are denoted by *U* which consists of all pairs of combinations of elements in ***A*** and ***B***, which are definitely not representing the same entity. Obviously, *U* is much bigger compared to *M*, because the size (cardinality) of *U* is comparable to ***A x B***, where the cardinality of *M* could be at most equal to the cardinality of ***A*** or ***B*** (whichever is smaller). Although ideally we should be able to partition the data source into these two U and M, we can classify a record pair as a possible match than to falsely decide on its matching status with insufficient information. Therefore, a third set P, called possible matched can also be introduced to the process. If we add this third set also to our model, to create these three sets (U, M and P), we compare the common attributes (or characteristics, e.g. the same attribute column in two data sources) of the two sources, and evaluate for each pair of items of a *comparison vector* γ. All possible realizations of γ define the *comparison space*.

Example 2.1: Suppose that we have two different files about individuals, where there are three attributes in common. These are SSN, first name, and second name. Then we define a comparison function $\gamma = (\gamma_1, \gamma_2, \gamma_3)$ as follows:

$$\gamma_1 (SSN_1, SSN_2) = \begin{cases} 0, & \text{if SSN is missing on either of the two records} \\ 1, & \text{if SSN's agree} \\ 2, & \text{if SSN's disagree} \end{cases}$$

$$\gamma_2(\text{name}_1, \text{name}_2) = \begin{cases} 0, & \text{if name is missing on either of the two records} \\ 1, & \text{if names agree exactly} \\ 2, & \text{if names disagree, but initial 4 letters agree} \\ 3, & \text{if names disagree completely} \end{cases}$$

$$\gamma_3(\text{surname}_1, \text{surname}_2) = \begin{cases} 0, & \text{if surname is missing on either of the two records} \\ 1, & \text{if surnames agree exactly} \\ 2, & \text{if surnames disagree, but initial 4 letters agree} \\ 3, & \text{if surnames disagree completely} \end{cases}$$

Assuming two records cannot have the same SSN and be different identities and vice versa, if $\gamma = (1, \gamma_2, \gamma_3)$ then we denote the pair as a *Match,* and if $\gamma = (2, \gamma_2, \gamma_3)$ then we denote the pair as a *Unmatch.* For the remaining 16 cases $(0, \gamma_2, \gamma_3)$ we need a similarity check indicating the necessity of the record linkage.

3. DATA PREPARATION

The record linkage process begins with a data preparation stage. Appropriate parsing of entry components is the most crucial part of computerized record linkage [11]. In this stage data entries are stored in a uniform manner in the dataset to achieve the structural homogeneity of the dataset. It usually includes parsing and standardization step.

3.1 Parsing of Data Components

Parsing is the first part in the preparation phase. It locates, identifies, and isolates individual data elements in the data set. It makes it easier to compare the data entries with each other since it enables to compare the corresponding components item by item rather than the long complex strings of data. For instance, if the dataset is composed of name and address components, the appropriate parsing of these into separate blocks of information are an indispensable task in record linkage. Without this step, the record linkage procedure would classify many pairs of the same entity as nonlinks, because the corresponding blocks of the entries could not be compared [11] or these blocks are erroneously compared with other distinguishing parts of the data like name component of an entry compared with the combined name and surname component of another data entry.

3.2 Standardizing of Data Components

Data standardization means standardizing the values in certain fields of the entries to a predefined uniform content format [12]. For instance without standardization, there may be the case that ‘CORP’ and ‘Corporation’ occur in different places in the same data set which would lead to the confusion of the computerized record linkage that these entities represent different objects. Moreover, first name spelling variations such as “Rob” and “Bobbie” might be replaced with ‘Robert’ or with a word such as ‘Robt’ because ‘Bobbie’ might refer to a woman with her first name ‘Roberta’.

Table 3.1 and 3.2 of parsing and standardization from [11] exemplifies this phase. In the tables 3.1 and 3.2, following abbreviations are used: PRE stands for prefix, POST 1 and POST 2 stand for postfixes, BUS 1 and BUS 2 refer to commonly occurring words associated with businesses, Hsnm and Stnm refer to house and street numbers respectively, RR refers to railroad, BLDG refers to building.

Table 3.1
Examples of Name Parsing

STANDARDISED	PARSED							
	PRE	FIRST	MIDDLE	LAST	POST1	POST2	BUS1	BUS2
DR John J Smith MD	DR	John	J	Smith	MD			
Smith DRY FRM				Smith			DRY	FRM
Smith & Son ENTP				Smith		Son	ENTP	

Table 3.2
Examples of Address Parsing

STANDARDISED	PARSED									
	Pre 2	Hsnm	Stnm	RR	BOX	Post 1	Post 2	Unit 1	Unit 2	Bldg
16 W Main ST APT 16	W	16	Main			ST		16		
RR 2 BX 215				2	215					
Fuller BLDG SUITE 405									405	Fuller
14588 HWY 16 W		14588	HWY 16				W			

4. BLOCKING

To compare all available records even in moderate sized datasets is not feasible and should be avoided. The idea of blocking tries to solve this problem with various techniques. Basic definition and the primary algorithms are described below.

4.1 Definition and Motivation

The main goal of the blocking phase is to reduce the number of comparisons of the data entries with each other. Since these comparisons are expensive and imposes the most serious bottleneck of the process. As an example, even moderate sizes of 10 thousand records for each data set imposes in the naïve nested loop approach 100 million operations with in-depth similarity check. That is to say, it is tried to avoid the quadratic complexity due to "nested loop" approach.

Formally, blocking is defined as a partition of the file into blocks where the complex comparisons are limited to within these blocks. There are three main ways to reduce the number of record comparisons: these are blocking based on a blocking key [13], sorted neighborhood approach [9] and canopy clustering [14].

4.2 Blocking Based on a Blocking Key

In this approach, blocking can be implemented by sorting the entire dataset according to a blocking key. The blocking key is usually a combination of field entries with high discrimination power, and the entire data set can be sorted based on this key. The entries are then compared whenever they have the identical keys. A more efficient way to implement blocking is to use hash tables. The value of the hash function determines which bucket each data entry, and within each bucket the entries are processed.

Although blocking increases the speed of the process considerably, it has two main drawbacks which are indirectly proportional to each other. First, the more discriminative the keys are, the less comparisons are processed within the entire dataset, but there is a danger that the matched pairs are not compared due to the reason that they do not belong to the same blocking group. On the other hand, the less discriminative the blocking keys are the more data entries are compared with each other, this leads to the increase of the run time.

For the complexity considerations, let *b* be the number of blocks, and assume that each block has *n/b* records. The number of record pairs will be $b \cdot O(n^2/b^2)$ i.e. $O(n^2/b)$. The time complexity of sorting is *O(nlog(n))* if hashing is not used. Hence, The total time complexity of blocking is $O(n^2/b)$.

4.3 Sorted Neighborhood Method (SNM)

In the naive sorted neighborhood approach two approaches are considered: partition the data to reduce the comparisons of large data sets and utilize parallel processing if a parallel processor is available. A methodology is required to effectively partition the data into blocks and consequently, the candidate sets are processed in parallel with a fixed sized window denoted by *w*. That is to say, the sorted neighborhood method can be summarized in the following three stages:

1. Key Creation: Crete a key for every entry of the dataset by combining relevant features of the entry
2. Data Sorting: Sort the records in the dataset based on the key retrieved in step 1.
3. Merging: Move a fixed size window through the list of records by comparing only those records that are within the range of the window. Namely, every record will be compared with the remaining *w*-1 records sequentially in top-down fashion.

Figure 4.1 [9] describes the process during the merge phase.

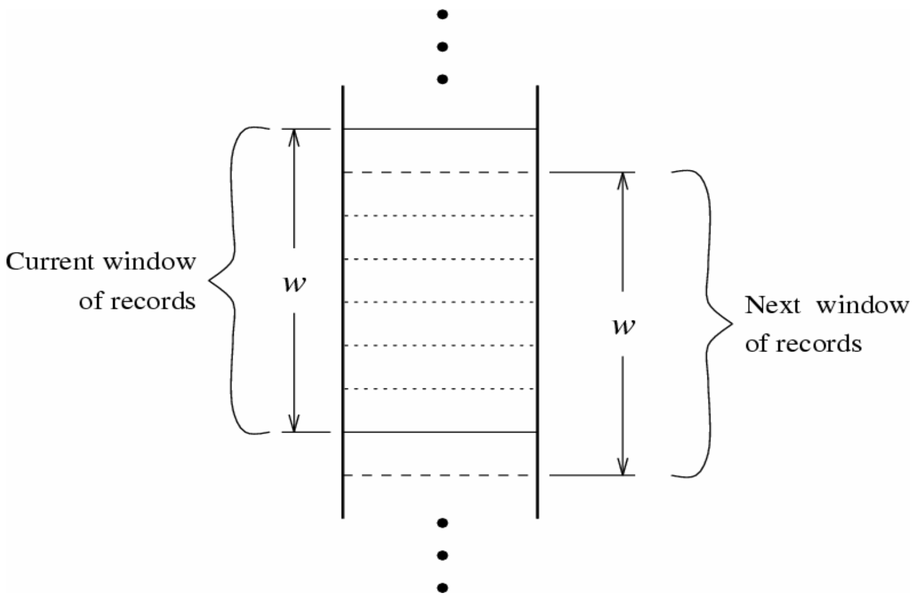

Figure 4.1
Window Scan during the Merge Phase

When this procedure is executed, the first step of creating the keys is an *O(N)* operation, the sorting phase is *O(N log N)*, and the merging phase is *O(wN)* where *N* is the number of records in the database. Thus, the complexity of the process is *O(N log N)* if $w < \log N$ otherwise it is *O(wN)*.

Furthermore, in very large databases, the dominant cost is usually disk I/O, that is to say the bottleneck will be the number of passes over the dataset. In this case, three passes will be necessary. One pass will be for preparing the keys, a second pass for the sorting algorithm and a final pass will be for window processing, this final pass may be processed through parallel processing if available.

The window size denoted by *w* is the parameter of the windows for scanning. The values of w range from 2 to *N* the whole of the dataset itself. The latter case means the nested loop approach itself. When w is taken as 2, then only the consecutive entries in the sorted dataset are compared. Hence, the *open* question of this approach is to determine the optimal settings for window size to maximize accuracy while minimizing computational cost [9].

The naïve sorted neighborhood method's second step, namely sorting the dataset, is usually preferred to be avoided due to time considerations when the dataset is considerably large. For that reason, Hernandez and Stolfo [6] propose an additional first stage. They consider first clustering the dataset into *n*-dimensional cluster space using the blocking key for each entry key. Then they apply the sorted neighborhood method to each individual cluster independently and in parallel if there is a parallel processor is available. Hernandez and Stolfo call this approach the clustering method. Given a group of two or more databases, these individual sets are combined with each other and turned in to one dataset of size *N*. This method can be summarized as follows:

1. Clustering Data: Traverse the records sequentially and for each record create a key of n-attributes key and map it into an n-dimensional cluster space based on the attributes of the key.
2. Sorted-Neighborhood Method: Apply the sorted-neighborhood method independently on each cluster as described above using the n-attributes key of step 1. We can use the key extracted above for sorting. Ideally, the whole cluster is assumed to be in the main memory during the operation.

The effectiveness of the sorted-neighborhood method highly depends on the key selected to sort the records. By the key of an entry we mean a subset of attributes in the database or substrings within the attributes chosen from the record with sufficient discriminating power.

As an example [9], the following key consists of the following substrings of each entry: the first three consonants of a last name, followed by the first three letters of the name field, followed by the address number field, and all of the consonants of the street name, followed by the first three digits of the social security number. These choices are due to the assumption that last names are typically misspelled and first names are less discriminative than the last names. The keys are used to sort the data in order to ensure that the matching data will be close to each other in the final sorted list. The example implies that the first and the last names are definitely the same entities, whereas the third one is also the same entity even though the last name attribute has been misspelled. The last one, on the other hand, is probably a different identity. Also note that our key construction made all entries take the same key value.

Table 4.1
Example Records and Keys

First	Last	Address	ID	Key
Sal	Stolfo	123 First Street	45678987	STLSAL123FRST456
Sal	Stolfo	123 First Street	45678987	STLSAL123FRST456
Sal	Stolpho	123 First Street	45678987	STLSAL123FRST456
Sal	Stiles	123 Forest Street	45654321	STLSAL123FRST456

After the blocking phase, i.e. in the merge phase, Hernandez and Stolfo [6,9] are using the knowledge intensive *equational theory*. Naturally, the more information there is available in the dataset, the better inferences can be made. As an example to the equational theory, consider the case where the two entries contain have the identical address and name values in their corresponding entry fields. It may be inferred that the two entries represent the same entity. A further example can be given as follows: Two social security numbers are the same but the names and addresses are totally different. These may infer that the two entries belong to the same person who moved or they could be two different people and there is an error in the social security number of the dataset. Hence, by checking the other relevant attribute fields in the database, if available, the better inferences can be made.

As an example to the equational theory [9] consider an employee database with the following rule:

```
Given two records, r1 and r2
IF the last name of r1 equals the last name of r2,
    AND the first names differ slightly,
    AND the address of r1 equals the address of r2
THEN
    r1 is equivalent to r2
```

The vague point in this rule is how to implement the expression “differ slightly”. For that purpose, a set of distance functions such as the ones in [15] are used to compare pieces of data which are usually string data. By applying the distance function to the corresponding record attributes we return a corresponding real number representing the similarity of the two pieces of information.

To reach sufficient accuracy, the inference process is divided into three stages. All records within a window are compared based on similar rules as above. In the second stage, the information gathered during the first stage is combined to see if we can come up with a decision. For those pairs of information that could not be *merged* due to lack of information gathered in the first stage, the algorithm checks the other subsequent relevant fields to get a result if available. Otherwise, in the first stage, the more precise and more time consuming distance functions are used as a final attempt to merge the two entries.

By selecting a threshold to capture obvious typographical errors we come up with a decision whether the pair represents the same entity. This point rises a new question of what the threshold should be and how to find this threshold. The answer is based on a significant amount of training data for the automated process or the data expert’s past experiences. Hence, Hernandez and Stolfo’s method is a knowledge intensive process which does heavily depend on the past experience. Furthermore, it is reasonable to expect that the rules proposed for a dataset will not give satisfactory results for another dataset.

A further consideration in this sorted neighborhood approach is that in general no single pass will be sufficient to catch all matching record pairs. The reason for that is by building the key an attribute that appears first in the key has higher discriminating power than those appearing after them. For a precise and explanatory example, assume that we have a database with SSN field. Assume further that, since SSN field has very high discriminating power the key begins with the first three numbers of this field. If we have two records with 283459783 and 823459783 corresponding SSN’s it is unlikely that they will fall under the same window.

For that reason, to increase the number of similar records merged, we have to execute several independent runs of the SNM by using a different key each time and a relatively small window. Hernandez and Stolfo call this as the *Multi-Pass* approach. Each independent run of the

Multi-Pass approach will produce a set of pairs of records. Although one field in a record may not match with the compared pair, another field may well match. The idea of *transitive closure* can be applied to those pairs to be merged. Transitive closure is a form of *equivalence relation* in the sense that "IF *A* implies *B* AND *B* implies *C* THEN *A* implies *C*". The following example [6] indicates the use of the transitive closure.

Example 4.1 Assume we have three census data entries of the form:
789912345 Kethy Kason 48 North St. (A)
879912345 Kathy Kason 48 North St. (B)
879912345 Kathy Smith 48 North St. (C)
By not changing the creation of the key but changing the order of the components the following data are retrieved:

Pass 1
KSN48NRTH789KET (Kethy Kason 789912345)
KSN48NRTH879KAT (Kathy Kason 879912345)

Pass 2
KATKSN48NRTH789 (Kathy Kason 789912345)
KATKSN48NRTH879 (Kathy Kason 879912345)

Pass 3
87948NRTHKATKSN (Kathy Kason 879912345)
87948NRTHKATSMT (Kathy Smith 879912345)

Hence, by three passes and using the transitive closure, we come up with the decision that the three entries represent the same person.

4.4 Canopy Clustering

The canopy clustering [14] is used to cluster large and high-dimensional data. “The key idea involves using an approximate distance measure to efficiently divide the data into overlapping subsets we call canopies”. After this blocking phase the process continues with the appropriate computationally complex comparison procedures applied to only those pairs that fall under the same canopies. In this approach, mutually exclusive blocks do not exist, instead being the two entries approximate similarity measure under the same *canopy* necessitates the complex comparison. Similarly, if the two entries do not fall into the same canopy that is to say these two entries’ similarity measure is below the preassigned lower threshold, no further check for comparison is applied. Furthermore, each entry does not create a canopy around its surrounding. That is to say, when the two entries compared with the cheap distance metric, if the two entries do fall “very close” to each other i.e. above a preassigned upper threshold, among those entries, only one entry can create a canopy around itself. Hence, the nested loop complexity is avoided this ingenious way. The idea behind this ignorance is that these two entries are so similar that creating a canopy among one of these entries should be also valid for the other entry.

Cohen [16] summarizes the canopy algorithm in the following pseudocode:

```
---------------------------------------------------------------------------------------------------
Input: set S, thresholds BIG, SMALL
Let PAIRS be the empty set.
Let CENTERS = S
While (CENTERS is not empty)
        – Pick some a in CENTERS (at random)
        – Add to PAIRS all pairs (a,b) such that SIM(a,b)<SMALL
        – Remove from CENTERS all points b’ such that SIM(a,b)<BIG
Output: the set PAIRS
---------------------------------------------------------------------------------------------------
```

Figure 4.2
Pseudocode for Canopy Algorithm

The following example with the corresponding figure 4.3 taken from [14] exemplifies a canopy clustering process.

Example 4.2: Points belonging to the same cluster are colored in the same shade of gray. The canopies were created based on the procedure described above. Point A was selected at random and forms a canopy consisting of all points within the outer (solid) threshold. Points inside the inner (dashed) threshold are excluded from being the center of, and forming the new canopies. Canopies for B, C, D and E were formed similarly to A. While there is some overlap, there are many points excluded by each canopy. Expensive distance measurements will only be made between pairs of points in the same canopy in the same canopies far fewer than all possible pairs in the data set. Figure 4.3

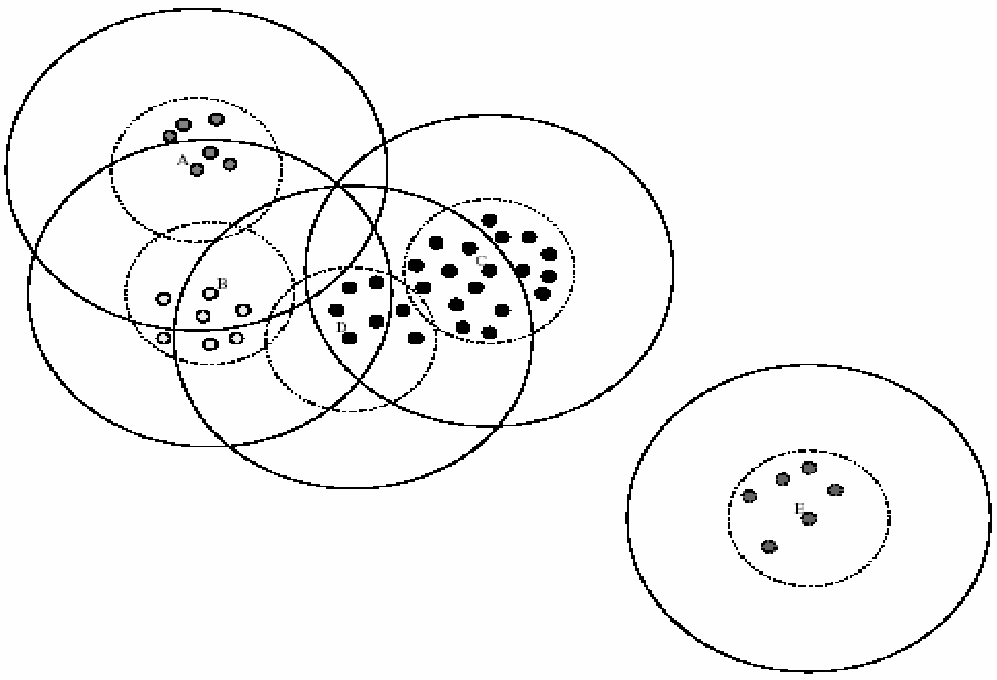

Figure 4.3
An Example of Four Data Clusters and the Canopies that Cover Them

The complexity of the canopy clustering is as follows: The number of record pair comparisons resulting from canopy clustering is *$O(fn^2/c)$* [17] where *n* is the number of records in each of the two data sets, c is the number of canopies and f is the average number of canopies a record belongs to. The threshold parameter should be set so that f is small and c is large, in order to reduce the number of comparisons. However, intuitively, as f is too small, the performance of the canopy clustering will decrease and it will not be able to catch the typographical errors.

4.5 Bigram Indexing

The other main blocking procedure is the so called *Bigram indexing*. This blocking system allows for fuzzy blocking. "The basic idea is that the blocking key values are converted into a list of bigrams (substrings containing two characters) and sublists of all possible permutations will be built using a threshold between 0.0 and 1.0. The resulting bigram lists are sorted and inserted into an inverted index, which will be used to retrieve the corresponding record numbers in a block"[17]

Example 4.3 As an example [17], assume that the word 'baxter' will be used. The word 'baxter' will result in the following bigram list: 'ba', 'ax', 'xt', 'te', 'er'. Assume that a threshold of 0.8 is selected. Since there are 5 bigrams, the following sublists of length 4 (5 x 0.8) will be inserted into the inverted index:

('ax', 'xt', 'te', 'er')
('ba', 'xt', 'te', 'er')
('ba', 'ax', 'te', 'er')
('ba', 'ax', 'xt', 'er')
('ba', 'ax', 'xt', 'te')

All record numbers which contain the blocking key value 'baxter' will be inserted into five inverted index blocks with the five keys above. Hence, there is a definite increase in the number of record pair comparisons compared to the blocking based on a unique key.

The complexity of the bigram indexing, when there exists two data sets with n records each is as in the standard blocking case $O(n^2/b)$. However, as to be seen from the table 4.2 below [17], the number of blocks b will be much larger in bigram indexing.

Table 4.2
Number of blocks produced by Bigram Indexing

Bigram Index Parameter	Number of blocks n = 9974
threshold t = 0.2	23695
threshold t = 0.6	48786
threshold t = 0.9	5663

5. THE COMPARISON

To compare two strings, we need to scale their similarities on real number domain. For that purpose beginning with primitive *ad-hoc* approaches ranging to complicated algorithms various rules and algorithms have been proposed.

5.1 Comparison Function γ

For the comparison of the records, we need a specific function, which has several values for the possible cases (of errors and differences) that are included in the two sources. The comparison function γ is defined for each attribute that are common to the two data sources separately. These attributes represent the identifying information at the records. There are mostly three categories of comparison functions, namely binary, categorical and continues comparison functions. Binary comparison functions assume a value of 0 or 1 after comparison. *Match* is indicated by 0 and an *Unmatch* is indicated by 1. Categorical functions have more distinguishing capacity as can be seen in the example above. Continuous functions, on the other hand, are computationally complex but have the most distinguishing power; and this is the primary reason why we have chosen the function of that type in our case. Most commonly used string similarity functions are explained below. The similarity metrics are mainly divided into three groups. These are character-based similarity metrics, token-based similarity metrics and phonetic similarity metrics.

5.1.1 Character-Based Similarity Metrics

The character-based similarity metrics aim to handle typographical errors. In this section the following character-based similarity metrics are described:

- Hamming Distance
- Edit Distance
- Affine Gap Distance
- Smith-Waterman Distance
- Q-grams
- N-grams
- Jaro's Algorithm

5.1.1.1 Hamming Distance

The Hamming distance is used for numerical attributes of the dataset of fixed size. These can be fields like Zip Code or SSN. It counts the number of different characters between two numbers. For instance, the Hamming distance between Zip codes “54905” and “53901” is 2 since they have 2 different characters [13].

5.1.1.2 Edit Distance

The Hamming distance function cannot be used for fields of variable length. Hence, it can not be used in the field values like “John” versus “Jon” or “John” versus “Johhn”. The edit distance between two strings is the minimum cost to convert one of the two entries into another by a sequence of character insertions, deletions or replacements. Each one of these modifications can have different cost values. Intuitively, changing the character to another character rather than deleting or replacing the characters in a sequence would cost more. Hence, it would imply dissimilarity stronger. For example [13], if we assume that the insertion cost and the deletion cost are each equal to 1, and the replacement cost is equal to 10, then the edit distance between “John” and “Jon” is 1. In order to achieve reasonable accuracy homogeneity, the modification costs should be standardized for every comparison as achieved in the [15].

5.1.1.3 Affine Gap Distance

The edit distance metric described above does not work well when one of the strings is abbreviation of the other like in “John R. Smith” versus “Jonathan Richard Smith”. The affine gap distance metric tries to handle this problem by introducing two extra edit operations open gap and extend gap [12] so that in our case the strings “John” and “Jonathan” as well as “R.” and “Richard” are perceived as coherent.

5.1.1.4 Smith-Waterman Distance

Smith and Waterman [18] described an extension of edit distance and affine gap distance in which mismatches at the beginning and at the end of the strings should be penalized less than mismatches in the middle. This metric allows for better substring matching. Therefore, the strings "Prof. John Smith, University of Washington" and "John Smith, Prof." are considered as similar using the Smith-Waterman distance since the prefix and suffix differences cost less [12].

5.1.1.5 N-grams

The N-grams comparison function forms the set of all the substrings of length n for each string. The distance between two strings is defined as

$$\sqrt{\sum_{\forall x} | f_a(x) - f_b(x) |}$$

where $f_a(x)$ and $f_b(x)$ are the number of occurences of the substring x in the two strings a and b, respectively. The substring length is usually selected of size n=2 or n=3. For example, "John Smith" and "Smith John" results in 0.375 using trigrams and the calculation returns 0.222 using bigrams [13] assuming 0 represents perfect match between two strings.

5.1.1.6 Jaro-Winkler Algorithm

It is also a string comparison that accounts for insertions, deletions, and transpositions. Jaro's algorithm [19] finds the number of common characters and the number of transposed characters in the two strings. The idea behind it is that two strings may represent the same entity if the same characters are close to each other. That is to say, a common character is a character that appears in both strings within half the length of the shorter string. A transposed character is a common character that appears in different positions.

For Example: "John" and "Jon": results in three common characters, none of which is transposed.

The Jaro's comparison function calculates

$(c/ l_1 + c/l_2 + (2c - t) / 2c) / 3$

where c is the number of common characters, t is the number of transposed characters, and l_1, l_2 are the lengths of the two strings [13].

Winkler [20] modified the original Jaro string comparator introduced by Jaro in the following three ways:

- A ‘similar’ character has 0.3 as value among common characters of two strings. Winkler takes the number “1” and the character “l” as similar as well as key punch errors “V” versus “B”.
- The differences between the beginning of two strings are penalized more. This is based on the observation that the typos occur rarely at the beginning of a string and it is expected that key punch errors occur more often in the middle parts of the string.
- The string comparison value is adjusted if the strings are longer than six characters or if more than half the characters aside from the first four letters agree [1].

5.1.2 Token-Based Similarity Metrics

Usually, character-based similarity metrics perform well for typographical errors. However, as the name implies the character-based similarity metrics do not work well for the typographical errors caused by the rearrangement of the words e.g. of the form “John Smith” versus “Smith John”. In such cases the character-based similarity metrics try to compare “John” with “Smith” and “Smith” with “John”. Hence, the result implies that the two entries do not refer to the same entity.

5.1.2.1 Atomic Strings

Monge and Elkan [21] came up with a basic algorithm for matching text fields based on *atomic strings.* An atomic string is a sequence of characters delimited by punctuation characters. Two atomic strings refer to the same entity if they are equal or if one is the prefix of the other. Based on this algorithm, the similarity of two fields is calculated as the number of their matching atomic strings divided by their average number of atomic strings [12].

5.1.2.2 WHIRL

Cohen described a system named WHIRL [22] that adopts from information retrieval the cosine similarity combined with the *tf.idf* weighting scheme to compute the similarity of two fields. Cohen separates each string σ into words and each words w is assigned a weight

$$v_{\sigma}(w) = \log(tf_{w} + 1) \cdot \log(idf_{w}),$$

where tf_{w} denotes the number of times that w appears in the field and idf_{w} is $\frac{|D|}{n_{w}}$, where n_{w} is the number of records in the database D that contain w, $|D|$ is the total number of entries in the database. The *tf.idf* weight for a word w in a field is high if w appears many times in the field (high tf_{w}) and w is a sufficiently "rare" term in the database (high idf_{w}).

For instance, for a collection of company names, rare terms such as "AT&T" or "IBM" will have higher *idf* weights than frequent terms such as "Inc.". The cosine similarity of σ_1 and σ_2 is defined as

$$sim(\sigma_1, \sigma_2) = \frac{\sum_{j=1}^{|D|} v_{\sigma 1}(j) \cdot v_{\sigma 2}(j)}{\| v_{\sigma 1} \|_2 \cdot \| v_{\sigma 2} \|_2}$$

The cosine similarity metric works well for various entries with different characteristics [12]. The similarity value of two strings does not change when the location of words within two tokens are different given the two words are identical. For example, "John Smith" is equivalent to "Smith John". Also, the appearance of frequent words only minimally affects the similarity of the two strings due to the low *idf* weight of the frequent words since these words are of low discrimination. For example, "John Smith" and "Mr. John Smith" would have similarity close to one. On the other hand this similarity metric does not capture word spelling errors. For example, the strings "Compter Science Department" and "Deprtment of Computer Scence" will have zero similarity under this metric since every word composing the two token is not identically same.

5.1.2.3 SoftTF-IDF

Bilenko [23] suggests to overcome the shortcomings of cosine similarity metrics by seeking "similarity" rather than "equality" of the words building up the tokens to be compared. Pairs of tokens that are similar in the character-based similarity metrics mentioned above are also considered in the cosine similarity formulae. However, the product of the weights is multiplied by this similarity measure as well hence discriminating the identical tokens from the non identical but similar tokens.

5.1.2.4 Q-Grams with *tf.idf*

Another token-based similarity comparison is posed by Garavano [24]. In this setting the words composing the tokens are not compared but rather as in the bi-gram indexing, *q-grams* are used. Since q-grams are robust to typographical errors [12], the two tokens given above as a shortcoming of WHIRL "Compter Science Department" and "Deprtment of Computer Scence" will have high similarity under this setting. Furthermore, the two tokens of "Gateway Communications" and "Communications Gateway International" will have higher similarity value since the word "International" will appear in more than one entries, hence will have a low *idf* weight in the same spirit of Cohen's idea.

5.1.3 Phonetic Similarity Metrics

As the name implies, phonetic similarity metrics do not focus on the identical characters or words in the entries. They are searching for the strings which are phonetically similar even though their spellings may differ considerably. As an explanatory example, the word "Kageonne" is definitely not similar to "Cajun" when compared with the character-based metrics described above. However, they are phonetically similar. Phonetic similarity metrics do not perceive two identical strings as different entities as well, since trivially, the two identical words are phonetically identical as well. For that reason, in the traditional blocking methods the phonetic similarity has been considered during the key generation of the entries in the dataset. The most prominent and widely accepted phonetic similarity metric is the *Soundex* encoding, there are two subsequent improvements to the *Soundex* encoding which are *New York State Identification and Intelligence System (NYSIIS)* and *Oxford Name Compression Algorithm (ONCA).* These three representations are described below.

5.1.3.1 Soundex Encoding

Soundex encoding is the most common phonetic coding scheme [12]. It is based on the assignment of identical code digits to phonetically similar groups of consonants and is used mainly to match surnames. Soundex encoding's basic idea is to group the letters with similar sounds into one symbol. The procedure is as follows [12]:

- Keep the first letter of the surname as the prefix letter and ignore W and H in every position but the beginning.
- Assign the following codes to the remaining letters:
    - B,F,P,V → 1,
    - C,G,J,K,Q,S,X,Z → 2,
    - D,T → 3,
    - L → 4,
    - M, N → 5,
    - R → 6
- Keep the letter prefix and the three first codes, padding with zeros if there are fewer than three codes
- Complete the code with zeros if the soundex code of the string has less than three numbers are encoded.

As an example [13], the Soundex code for both "Hilbert" and "Heilbpr" is H416; the Soundex code for both "John" and "Jon" is J500.

5.1.3.2 New York State Identification and Intelligence System (NYSIIS)

The NYSIIS system differs from Soundex in that it retains information about the position of vowels in the encoded word by converting most vowels to the letter A. Furthermore, NYSIIS does not use numbers to replace letters; instead it replaces consonants with other, phonetically similar letters, so that it retains a purely alpha code i.e. no numeric component in the code.

Usually, the NYSIIS code for a surname is based on a maximum of nine letters, of the full alphabetical name, and the NYSIIS code itself is then limited to six characters.

5.1.3.3 Oxford Name Compression Algorithm (ONCA)

ONCA is a two-stage technique, designed to overcome most of the unsatisfactory features of the Soundex encoding scheme but still sticking to the fixed four letter encoding of Soundex scheme. In the first step, the algorithm uses a British version of the NYSIIS method of compression. Then in the second step, the transformed and partially compressed name is put into process of Soundex encoding as described above. This technique gives successful results when grouping similar names together [12].

5.2 Factors Influencing the Performance of the Comparison Function

The factors influencing the performance of the comparison function are various. The main reasons are variability of representations and the appearance of null values.

5.2.1 Reasons for Different Representations of the Same Entity

The influence of the correct selection of the comparison functions for the corresponding entries is significant. i.e. a decision process can perform efficiently only if the similarity values between the duplicates and the non-duplicates are significantly different.[25]. However, the selection of the appropriate comparison function is difficult since the characteristics of the differences can be various. There are three main causes for the differences. These are typos, datatype dependency and domain dependency.

First, typos, i.e. typographical errors are the easiest case to catch. They may be due to wrong, additional or interchanged characters in a string to compare. String similarity measures described above are perform well on these types of errors.

Second characteristic is the datatype dependency in a dataset. If a value is not a simple string but some kind of a primitive data type the string similarity measures perform poorly on this case. For instance [25], consider the two date values “1999” and “2000”. If these two values are considered as two strings the string similarity function will conclude that the two strings are highly dissimilar to each other, but a numeric comparator would decide that they are highly similar. Conversely, if a numeric comparator is used in the corresponding field, the date values “2001” and “2010” will be denoted as highly dissimilar strings even though the reason for the different values may be a typographical error and would be classified as similar by a string comparator.

Third is the so called domain dependency. This is the most difficult case to handle. As an explanatory example [25] “VLDB-95” and “Int. Conference on Very Large Databases, 1995” look completely different although they have the same meaning. For this type of differences, usually a dataset expert’s help is needed.

5.2.2 Null Values

Datasets often contain null values. For instance, in a census data the address information of an individual may well be missing in an entry whereas in another entry the information may be partially available. To overcome this obstacle there are three main methods [25].

First, it can be assumed that a null value never matches with the corresponding component of the entry to be compared. Second, contrary to the first approach, it can be assumed that the null value matches completely with the corresponding component of the entry to be compared. Third, the null value is replaced with a similarity value. This similarity value can be the most probable value for that pair, for instance the mean value of the other comparisons on the corresponding components.

6. SURVEY OF RECORD LINKAGE METHODS

Record linkage methods based on corresponding samples gave rise to probabilistic record linkage as well as supervised and unsupervised learning schemes. A survey is given below.

6.1 Probabilistic Record Linkage Model

The probabilistic record linkage [26] was developed in 1969 and is still widely used in the statistical domains. It tries to estimate the matched and unmatched probabilities of realization of a computer vector in the whole population.

6.1.1 General Framework

The process of probabilistic record linkage can be described in the following manner: The conditional probabilities for any values of ɣ are:

P(ɣ | M) stands for the probability of that particular realization given that the pair is matched. Based on the previous example, assume that the two entries represent the same person. Either one or both of the SSN's lack and ɣ = (0,2,1). Now assume also that there are 394 pairs in the matched set. We just count the frequency of this realization in that set. Assume we observe that 5 of them take the value in the corresponding comparison vector (0,2,1); so *P(ɣ | M)* = 5 / 394. Similarly we define the unmatched conditional probability *P(ɣ | U).*

Now we evaluate the Likelihood Ratio:

$$\lambda(\gamma) = \frac{P(\lambda | \mathrm{M})}{P(\lambda | U)}$$

For every realization of ɣ we have three possible decisions which are Match (Link), Possible Match (Possible Link), Unmatch (Nonlink). Table 4.1 [27] describes the possible errors:

Table 6.1: The different cases of matches and links
[1]error of first type arises (called as α error)
[2]error of second type arises (called as β error)

reality decision	match (M)	nonmatch (U)
link (L^+)	O.K.	false link[1]
Nonlink (L^-)	False nonlink[2]	O.K.
Possible link ($L^{\pm}$)	left to clerical review	left to clerical review

A good linkage rule minimizes the probability of the second decision (the possible link) under the condition that the probability of errors made by false decisions (the false links and false nonlinks) are bounded by some constants $\alpha,\beta \in (0,1)$.

Based on the mathematically proven to be pareto-optimal model of Fellegi and Sunter, we can derive the upper bound λ_u (UPPER) and lower bound λ_l (LOWER) in the following way:

We denote the conditional probabilities as

$$m(\gamma) \equiv P(\gamma|\mathrm{M}) \text{ and } u(\gamma) \equiv P(\gamma|U)$$

Then we order the different realizations of γ (e.g. in our case it may be (0,2,1), (0,1,3)) such that the Likelihood – Ratios also called *weights* of that particular realization

$$\lambda(\gamma) = \frac{m(\gamma)}{u(\gamma)}$$

are monotone decreasing. When the Ratio is the same for more than one realization of γ, we order these γ arbitrarily. If the realizations of γ were $u(\gamma) = 0$ we put at first into this ordering.
We index the ordered set $\{\gamma\}$ by the subscript i, (i =1, 2, ... , N_R) where N_R stands for the different number of realizations of γ and write

$m_i \equiv m(\gamma_i)$ *and* $u_i \equiv u(\gamma_i)$

Now based on predetermined values of error rates α and β we want, i.e. the error probabilities, the UPPER and LOWER values create themselves if we choose two numbers $r, s \in (1, 2, ..., N_R)$ where N_R stands for the number of records as follows:

$$\alpha = \sum_{i=1}^{r} u_i, \quad \beta = \sum_{i=s}^{N_R} m_i, r < s$$

So; based on *r* and *s* the bounds UPPER and LOWER are then

$$\lambda_l = \frac{m(\gamma_s)}{u(\gamma_s)} \text{ and } \lambda_u = \frac{m(\gamma_r)}{u(\gamma_r)}$$

In order to determine the parameters $m(\gamma)$ and $u(\gamma)$, it is assumed that there exists the conditional independence between the components retrieved from the corresponding entries of the fields of the database. Under this assumption the corresponding conditional match and unmatch probabilities are as follows:

$$m(\gamma) = \prod_{i=1}^{k} m_i(\gamma_i)$$
$$u(\gamma) = \prod_{i=1}^{k} u_i(\gamma_i),$$

where *k* stands for the corresponding number of components of a given entry.

Thus, we come up with the following rule:

- *IF R > UPPER, THEN DESIGNATE THE PAIR AS LINK.*
- *IF LOWER ≤ R ≤ UPPER, THEN DESIGNATE THE PAIR AS A POSSIBLE LINK*

- *AND HOLD FOR CLERICAL REVIEW.*
- *IF R < LOWER, THEN DESIGNATE THE PAIR AS NONLINK.*

where R stands for the ratio of that realization γ *namely*:

$$R = \lambda(\gamma) = \frac{m(\gamma)}{u(\gamma)}$$

Remark: Based on above, it is easy to calculate the reverse conditional probability using the Bayes rule. Given any realization γ, one can calculate the un/matching probability using the basic identity $P(\gamma \mid M)\, P(M) = P(M \mid \gamma)\, P(\gamma)$ where $P(\gamma)$ represents the relative frequency of the particular realization and $P(M)$ represents the relative frequency of the size of Matched set to the size of the sum of the Matched and Unmatched pairs. $P(\gamma \mid U)\, P(U) = P(U \mid \gamma)\, P(\gamma)$ can be used similarly.

There are two main deficits of the Fellegi-Sunter approach. First, the formula based on conditional independence does not hold usually in real life scenarios. For instance, given a census data for containing the address information of the people the street and city numbers are definitely not independent. Second, as it will be mentioned in the supervised machine learning approach, there needs to exist a significant amount of training data of matched and unmatched pairs so that the conditional probabilities of the matched and unmatched pairs are estimated in the whole data set and the upper and lower thresholds are valid for the entire dataset.

6.1.2 The Construction of the Learning Sets

It is easy to generate the Unmatched set, since while the size of A and B are huge whereas there are few pairs which represent the same identity. The randomly chosen set U is an appropriate approximation of the real case.

The matched set M is difficult to construct. This is often done manually. Firstly, we can take all (exact) matchings of the two sources and after that we add some other pairs, where we can see that they belong together. The procedure should be done with a lot of caution since the pairs that we put into M contain exactly the information about the data (especially the errors and differences inside the corresponding fields of the two sources), that we apply in the subsequent steps of the Record Linkage process. That is to say, in statistical terms, our *sample* should contain the characteristics of the whole *population*. To overcome this problem Winkler proposes to use the EM algorithm to calculate the conditional probabilities of matched and unmatched as well as the upper and lower bounds by iteratively injecting the training samples into the model until the estimated parameters do not change considerably after each training sample.

6.2 Machine Learning Approach

After the domination of the Fellegi-Sunter probabilistic record linkage for tens of years, with the rise of machine learning techniques, the record linkage problem has been begun to be considered by the AI community as a classification problem. The two main approaches in the machine learning process are the supervised and unsupervised record linkage.

6.2.1 Supervised Record Linkage

One of the limitations of the probabilistic model is that they do not handle highly discriminative continuous comparison vectors very well. Decision models based on machine learning techniques can overcome this shortening.

In supervised training, a training set of patterns, in which the class of each pattern is known a priori, is used to build a model that can be used afterwards to predict the class of each unclassified pattern. A training instance has the form of <x, f(x)> where x is a pattern and f(x) is a discrete-value function that represents the class of the pattern x. In case of the record linkage, x is the comparison vector c and f(c) is either M, U (or P).

The most popular classification technique is decision trees. Predictions are made based on the training data. Decision trees perform well, when there is a considerable amount of past experience of similar situations available. The reason for adapting the decision trees to the record linkage problem is that they do not just classify the given instance based on its experience, but they can also give the corresponding rule of the corresponding conjunctive and/or disjunctive conditions. In the record linkage problem Quinlan's C4.5 algorithm [28], successor of his ID3 algorithm [29] can be used. The reason for the preference of C4.5 over ID3 is as follows: First, since ID3 is limited to only categorical values, it can not handle continuous valued similarity values retrieved from the comparison of two entries in the dataset. Second, there may be the case that one of the entries has null values in the corresponding comparison field, in which case C4.5 is able to handle these null values successfully.

7. UNSUPERVISED RECORD LINKAGE

To overcome the disadvantage of the necessity of a large training data which is representative of all records; the unsupervised learning method can be applied. In unsupervised training the notion of a training set does not exist. The whole set of patterns is given as input to the unsupervised learning algorithm to predict the class of each unclassified pattern, or in our case the matching status of each record pair. Clustering is the only known way for unsupervised learning. Formally speaking, clustering is separation of data into groups of similar objects. The objects in each group, called cluster, are similar to each other and dissimilar to objects of other groups [30].

In this approach, each pattern generated from a pair based on a comparator is represented as a point in the space and the clustering algorithms try to cluster these points into k clusters, in our case k =2 denoting the three available classes: M, U. (or k=3 depending on whether the rejection area P is to be created)

In the context of record linkage problem, three clustering algorithms can be applied to the record linkage problem, namely the k-means clustering, the hierarchical clustering and the expectation-maximization (EM) algorithm for clustering.

7.1 k-means Clustering

The *k-means* algorithm [31, 32] is the most popular clustering tool used in scientific and industrial applications [30]. The algorithm takes its name from representing each of k clusters C_j by the mean c_j of its points, the so-called centroid [30].

The k-means clustering can be summarized as follows [33]:

1. Partition the whole dataset into k clusters randomly.
2. Compute the mean (centroid) of each cluster.
3. For each point, calculate the distance from the data point to its mean. Take the data point

to the closest centroid.

4. Repeat steps 2 and 3 until the coordinates of the centers of the clusters do not change.

In the context of record linkage, after applying the clustering algorithm to the set of comparison vectors, the issue is to determine which cluster represents which of the two classes. Assuming that the exact match has 1 in all its dimensions; and the ideal unmatch like "John" versus "Adam" has 0 in all its corresponding fields; the cluster that has its centroid close to the point with all 1's is the cluster of the matched records, whereas the cluster closest to the origin 0's is the cluster of the unmatched records.

The general drawbacks of the algorithm are strong dependence of results on the initial guess of centroids, sensitivity with respect to outliers and coverage of numerical attributes only [30]. Furthermore, the main disadvantage of k-means specific to the record linkage problem is that the distances to the centroid are in general calculated by a simple Euclidean distance, which necessitates that the cluster shape is spherical. Unfortunately, this is not the case in real life scenarios; the shape of the cluster is usually different from a sphere. Based on the distinguishing power, the features of the comparison vector show very different distributions and they may also depend on each other, which may result in non-spherical cluster shapes [34].

7.2 Expectation-Maximization (EM) Algorithm Based Clustering

EM algorithm [35] is a statistical model that makes use of the finite Gaussian mixtures model. The parameters are computed consequently in the expectation and maximization steps and until the convergence of the corresponding parameters. The finite mixtures model assumes all attributes to be independent random variables.

To give an example [36], an EM process for two-component Gaussian mixture is described below:

$Y_0 \sim N(\mu_0^2, \sigma_0^2)$

$Y_1 \sim N(\mu_1^2, \sigma_1^2)$

$Y = (1-\Delta)\, Y_0 + \Delta\, Y_1$

$\Delta \in \{0, 1\}$

$Pr\,(\Delta=1) = p$

The density of Y is

$g_Y(y) = (1\text{-}p)\,\phi_{\theta_0}(y) + p\phi_{\theta_1}(y)$

The log-likelihood is

$$l(\theta;Y) = \sum_{i=1}^{n} \log[(1-p)\phi_{\theta_0}(y_i) + p\phi_{\theta_1}(y_i)]$$

Since the Δ's are unknown, substitute for Δ_i its expectation i.e.

$\gamma_i = \mathrm{E}(\Delta_i | \theta, Z) = \Pr(\Delta_i = 1 | \theta, Z)$

Hence, the algorithm has the following steps:

- Take initial guesses for the parameters $\mu_0, \mu_1, \sigma_0, \sigma_1, p$.
- *Expectation* Step: compute the responsibilities:

$$\gamma_i = \frac{p\phi_{\theta_1}(y_i)}{\left(1-p\right)\phi_{\theta_0}(y) + p\phi_{\theta_1}(y)}$$

- *Maximization* Step: compute the weighted means and variances:

$$\mu_0 = \frac{\sum_{i=1}^{n}(1-\gamma_i)\,y_i}{\sum_{i=1}^{n}(1-\gamma_i)},\quad \sigma_0^{\,2} = \frac{\sum_{i=1}^{n}(1-\gamma_i)(y_i-\mu_0)^2}{\sum_{i=1}^{n}(1-\gamma_i)}$$

$$\mu_1 = \frac{\sum_{i=1}^{n} \gamma_i y_i}{\sum_{i=1}^{n} \gamma_i}, \quad \sigma_1^2 = \frac{\sum_{i=1}^{n} \gamma_i (y_i - \mu_1)^2}{\sum_{i=1}^{n} \gamma_i}$$

and the mixing probability $p = \sum_{i=1}^{n} \gamma_i / n$

- Iterate expectation and maximization steps to convergence

7.3 Hierarchical Clustering

Given a set of *N* items to be clustered and an $N \cdot N$ similarity matrix, the hierarchical clustering [37] has the following basic steps:

1. Start by assigning each item to a cluster so that there are *N* clusters of size 1 initially. Let the distances (similarities) between the clusters the same as the distances (similarities) between the items they contain.
2. Find the most similar pair of clusters and combine them into a single cluster so that the total number of clusters is decreased by one.
3. Compute distances between the new generated cluster and the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size *N*.

After acquiring the complete *hierarchical tree* with this procedure, if *k* clusters are needed, the *k*-1 longest links are cut if we want two clusters. This is the exact operation done in our record linkage problem. The longest link is separated from the whole so that two clusters are retrieved denoting the Matched and Unmatched pairs.

Step 3 can be done in three different ways, these are *single-linkage*, *complete-linkage* and *average-linkage* clustering. In *single-linkage* clustering, the distance between one cluster and another cluster equals to the shortest distance from one of the members of a cluster to any member of the other cluster. *Complete-linkage* clustering uses the opposite approach. That is to say, the distance between one cluster and another cluster equals to the longest distance from one of the members of a cluster to any member of the other cluster. In *average-linkage* clustering, the distance between one cluster and another cluster equals to the average distance from one of the members of a cluster to any member of the other cluster [38].

The main drawbacks of the algorithm are that they do not scale well that is to say time complexity is of at least $O(n^2)$, where n is the number of total objects. Moreover, they can never undo what was done previously [34].

8. IMPLEMENTATION and EXPERIMENTS

To function effectively, one of the more widely used string similarity metrics along with the prominent machine learning tool are used. The libraries and the test results are described and evaluated below.

8.1. Tools and Libraries

To create the comparison vectors, SimMetrics [15] tools' similarity metrics written in JAVA have been integrated into the project. It is an open source tool of similarity or distance metrics. It provides float based similarity functions between string entries. It is intended for researchers in information integration and other related fields. "It includes a range of similarity measures from a variety of communities, including statistics, DNA analysis, artificial intelligence, information retrieval, and databases" [15]. It is very easy to use by giving two corresponding strings, then it returns a normalized float number between 0.0 and 1.0 where 0.0 means entirely different and 1.0 means complete match.

For the k-means clustering, the widely-used WEKA's [39] source codes have been modified to adapt to the project, which is a collection of machine learning algorithms for data mining tasks. The k-means clustering has been realized by inputing the number of the clusters to be created, namely the two clusters of Matched and Unmatched, to the interface and stop whenever the iterations do not change the two clusters considerably.

The Expectation-Maximization (EM) algorithm is also part of the WEKA clustering package. The algorithm is implemented in the following fashion: The two clusters represent the two sets to achieve as before. The probability distributions are assumed to be normal. Our algorithm tries then to find the value of three parameters for each cluster, i.e. the mean, standard deviation of each cluster and the sampling probability for the two clusters. Using these values repeatedly, our algorithm follows [40]:

1. Guess initial values for the five parameters.
2. Use the probability density function for a normal distribution to compute the cluster probability for each instance.

3. Use the probability values to re-estimate the five parameters.
4. Return to Step 2

The algorithm terminates when the measure of cluster quality no longer shows significant increases. The likelihood is computed by multiplying the sum of the probabilities for each of the instances. For instance, assume that the matched pairs composes 1% of the entire comparison space. With two clusters of M and U containing comparison vectors of $\gamma_1, \gamma_2, .., \gamma_N$ the computation is:

$$[0.01 \cdot P(\gamma_1|M) + 0.99 \cdot P(\gamma_1|U)]\ [0.01 \cdot P(\gamma_2|M) + 0.99 \cdot P(\gamma_2|U)] \ldots\ [0.01 \cdot P(\gamma_n|M) + 0.99 \cdot P(\gamma_n|U)]$$

The hierarchical clustering is proceeded with the Cluster 3.0 tool [41]. Cluster 3.0 is an open source clustering software used primarily for gene expression analysis in the bioinformatics. The input values are formatted in its standard input form. The output files are processed to adapt to our specific problem. That is to say, the hierarchical tree created by the tool is divided into two subtrees up from the root node.

8.2 Datasets

In this thesis, the following prominent datasets with different characteristics have been used: The cora dataset with 1916 entries and 121 distinct papers. About one quarter of the papers have only one or two reference to them, whereas the rest has many duplicates with the most popular paper cited 108 times. Secondly, the restaurant dataset with a collection of 864 restaurant records from the Fodor's and Zagat's restaurant guides that contains 112 duplicates are used. These two datasets are retrieved from RIDDLE [42]. Thirdly, artificially generated census dataset provided by dbgen [9] with 1000 entries of which 500 are distinct and 500 are duplicates of those entries, that is to say there is exactly one duplicate for each entry.

8.3 Experiments and Evaluation

For each of these three datasets, the three clustering algorithms are implemented. These are k-means algorithm with 2 clusters, expectation maximization algorithm with 2 clusters assuming that the underlying probability distributions of the matched and unmatched pairs are normal and a modified hierarchical clustering which partitions the data into two clusters where the two clusters represent the matched and unmatched pairs.

All the experiments are conducted on Intel Celeron CPU of 1.60GHz with 1GB random access memory.

The evaluation metrics for the experiments are taken from the information retrieval literature and adapted to our specific case as follows:

$$\text{Precision} = \frac{|\,CorrectlyIdentifiedDuplicates\,|}{|\,IdentifiedDuplicates\,|}$$

$$\text{Recall} = \frac{|\,CorrectlyIdentifiedDuplicates\,|}{|\,TrueDuplicates\,|}$$

$$\text{F-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Using these metrics, the similarity metrics' performances are measured from the early primitive approach to the most complicated one. Namely, the Levenstein, Smith and Waterman and Jaro-Winkler algorithms are compared using the three main clustering algorithms. Token-based cosine-similarity metric is not put as a result in the tables below, since it searches only for identical representations and for that reason, the results were almost random. Also, the effect of single, average and complete linkage techniques of the hierarchical clustering algorithms is evaluated.

The results are given below.

Table 8.1
Performance Measurements in Percentages for Restaurant Dataset

Levenstein Similarity Metric			
	Recall	Precision	F1
k-means	63,7	36,17	46,14
EM	42,3	22,8	29,63
Hierarchical	65,27	56,6	60,63

Table 8.2
Performance Measurements in Percentages for Artificial Dataset

Levenstein Similarity Metric			
	Recall	Precision	F1
k-means	64,43	22,24	33,07
EM	41,7	15,23	22,31
Hierarchical	70,18	71,5	70,83

Table 8.3
Performance Measurements in Percentages for CORA Dataset

Levenstein Similarity Metric			
	Recall	Precision	F1
k-means	64,91	41,6	50,70
EM	52,1	33,2	40,56
Hierarchical	56,4	65,03	60,41

Table 8.4
Performance Measurements in Percentages for Restaurant Dataset

Smith Waterman Similarity Metric			
	Recall	Precision	F1
k-means	83,47	36,17	50,47
EM	77,94	32,98	46,35
Hierarchical	69,32	76,26	72,62

Table 8.5
Performance Measurements in Percentages for Artificial Dataset

Smith Waterman Similarity Metric			
	Recall	Precision	F1
k-means	84,18	16,24	27,23
EM	79,39	19,89	31,81
Hierarchical	74,82	76,55	75,68

Table 8.6
Performance Measurements in Percentages for CORA Dataset

Smith Waterman Similarity Metric			
	Recall	Precision	F1
k-means	74,01	46,17	56,87
EM	65,69	43,82	52,57
Hierarchical	60,54	70,38	65,09

Table 8.7
Performance Measurements in Percentages for Restaurant Dataset

Jaro-Winkler Similarity Metric			
	Recall	Precision	F1
k-means	92,64	32,17	47,84
EM	93,12	34,23	49,79
Hierarchical	73,57	84,23	78,11

Table 8.8
Performance Measurements in Percentages for Artificial Dataset

Jaro-Winkler Similarity Metric			
	Recall	Precision	F1
k-means	98,7	12,24	21,38
EM	98,7	15,89	26,02
Hierarchical	83,71	94,85	88,15

Table 8.9
Performance Measurements in Percentages for CORA Dataset

Jaro-Winkler Similarity Metric			
	Recall	Precision	F1
k-means	94,91	42,17	58,05
EM	92,69	53,82	67,25
Hierarchical	70,57	71,7	70,5

The results above underline the distinguishing power of the similarity metric. The most successful similarity metric is the Jaro-Winkler algorithm, which is usually the *penalyzer* similarity metric in the Soft-TFIDF schemes. The other earlier two algorithms, Smith-Waterman and the most primitive Levenstein distance measures have less distinguishing power. As a result of that, as the size and the variability of different representations of the same entity based on the size of the dataset increase, the importance of the distinguishing power of the similarity measure become more apparent. This fact is highlighted especially in the CORA dataset experiments. The EM algorithm which is trying to approximate the parameters of the normal distribution has performed worst with the similarity metrics of low distinguishing power. The reason for that is probably, as the comparison of different pairs gives close results; the underlying distribution gets dissimilar to a normal distribution. The k-means clustering scheme has also been affected by the low quality of the similarity metric, however, as the results show the decrease is not as drastic as was in the EM-clustering case.

Based on most sophisticated and most successful similarity metric, i.e. Jaro-Winkler algorithm, the characteristics of the EM-algorithm and k-means algorithm are similar whereas the hierarchical clustering has a complete different nature, which leads to different performances. The k-means and EM-algorithm are trying to find a general nature of the match and unmatch clusters, whereas the greedy hierarchical clustering algorithm combines the two entries that are “close” to each other based on the string similarity algorithms. Thus, the recall values of the EM and k-means clustering are high. The reason for that is that these two try to divide the pairs into two clusters, without taking into consideration the most similar ones. However, the hierarchical clustering proceeds case by case, by combining the two most similar ones and proceeding repeatedly.

Based on these considerations, the artificially generated dataset with exactly one replicate for each entry has been successfully processed by hierarchical clustering, but the other two algorithms did perform poorly compared to the hierarchical one.

The restaurant dataset has at most one duplicate for each entry, but not necessarily exactly one duplicate for each entry. Thus, again the hierarchical clustering has a high performance, but not as high as was the case in artificial clustering. The reason for that was probably that the greedy hierarchical algorithm by its nature had to find some matched pairs even though there are not any for that particular entry in the dataset.

In the CORA dataset, the picture has again changed considerably since this dataset has few duplicates for some entries (bibliographical citations in this case) but has many duplicates for some citations. Thus, the k-means and EM algorithm with again high recall has a higher precision thus a higher F1-measure. The hierarchical clustering on the other hand had the worst performance among the three datasets implemented. The reason for that probably was that some entries of the dataset have few duplicates, but a considerable amount of the dataset has many duplicates where the hierarchical clustering could not catch all these pairs. That is to say, these pairs were "closer" to the "unmatched" ones at that step of the greedy process of the hierarchical clustering. However, even though the hierarchical clustering has decreased considerably and the k-means and EM-algorithm recall values due to many duplicates in many entries have increased, the hierarchical clustering had still the highest F1-measure.

As a further study, three different linkage strategies of the hierarchical clustering are also studied to see their effect on the performance of the record linkage. Three datasets with Levenstein, Smith-Waterman and Jaro-Winkler algorithm are studied using single, average and complete linkage. The results are given below.

Table 8.10
Different Linkage Schemes of the Restaurant Dataset

Levenstein			
Linkage Types	Recall	Precision	F1
Single	64,91	55,96	60,10
Average	65,5	55,17	59,89
Complete	65,27	56,6	60,63

Table 8.11
Different Linkage Schemes of the Artificial Dataset

Smith-Waterman			
Linkage Types	Recall	Precision	F1
Single	70,11	70,69	70,40
Average	69,56	71,14	70,34
Complete	70,18	71,5	70,83

Table 8.12
Different Linkage Schemes of the CORA Dataset

Jaro-Winkler			
Linkage Types	Recall	Precision	F1
Single	57,18	66,09	61,31
Average	56,89	65,59	60,93
Complete	56,4	65,03	60,41

Table 8.13
Different Linkage Schemes of the Restaurant Dataset

Levenstein			
Linkage Types	Recall	Precision	F1
Single	67,01	75,46	70,98
Average	68,53	75,91	72,03
Complete	69,32	76,26	72,62

Table 8.14
Different Linkage Schemes of the Artificial Dataset

Smith-Waterman			
Linkage Types	Recall	Precision	F1
Single	73,98	75,04	74,51
Average	74,6	76,11	75,35
Complete	74,82	76,55	75,68

Table 8.15
Different Linkage Schemes of the CORA Dataset

Jaro-Winkler			
Linkage Types	Recall	Precision	F1
Single	61,78	71,42	66,25
Average	60,4	71,9	65,65
Complete	60,54	70,38	65,09

Table 8.16
Different Linkage Schemes of the Restaurant Dataset

Levenstein			
Linkage Types	Recall	Precision	F1
Single	92,1	33,97	49,63
Average	92,53	34,82	50,60
Complete	93,12	34,23	50,06

Table 8.17
Different Linkage Schemes of the Artificial Dataset

Smith-Waterman			
Linkage Types	Recall	Precision	F1
Single	82,1	94,04	87,67
Average	82,56	94,51	88,13
Complete	83,71	94,85	88,93

Table 8.18
Different Linkage Schemes of the CORA Dataset

Jaro-Winkler			
Linkage Types	Recall	Precision	F1
Single	69,78	69,94	69,86
Average	70,24	71,59	70,91
Complete	70,57	71,7	71,13

In most cases, hierarchical clustering based on complete linkage is slightly better than the two other cases. The reason for that is, probably, the complete linkage tries to measure the distance based on the maximum distance between two corresponding clusters. Hence, two clusters at that step are separated from each other as much as possible. Since, theoretically, the unidentical pairs will be far away from the identical ones, the complete linkage scheme favours this nature.

A sample time performance based on Jaro-Winkler similarity metrics has also been put to compare the time performances of the clustering algorithms to give a general idea of the time consumption of three clustering algorithms. The results are given in Table 6.18.

Table 8.19
Time Measurements in Seconds for 3 Datasets

	Restaurant	Artificial	CORA
k-means	4,93	13,35	17,84
EM	13,57	24,76	68,11
Hierarchical	7,14	17,23	29,79

As to be seen from the table above, based on time measurements, the k-means algorithm is the most efficient algorithm to be used; EM algorithm on the other hand, has the worst time performance overall.

8.4 Internal and External Threats

During this study, there has been used three software tools. These are [15], [39] and [41]. The experiments and the corresponding statistics are based on these tools. The classes of the corresponding tools may be bugous. Due to that, our performance measurements may not reflect the actual figures. Similarly, the time measurements of the clustering algorithms will not necessarily hold even on a second trial of the same algorithms at the same platform, but they give a general outline of the performance of the algorithms both in terms of the time and performance metrics. This should be remarked as an internal threat of our experiments.

Furthermore, due to our limited capacity of the platform we are using, the data sets are small and the results based on these datasets do not necessarily hold for other datasets of larger size and possibly of different characteristics. This fact should also be remarked as an external threat of our experiments.

9. CONCLUSION

The main problem with record linkage is the lack of training data for building an accurate model for classification. In this study, three clustering algorithms are applied to the record linkage problem on three datasets with different characteristics to see their performances both in terms of precision and recall and in terms of time performances.

The reason for the interest is due to the fact that the state of the art record linkage work [13] has used the standard k-means algorithm which is implemented in almost any data mining toolbox without taking into account the working principal of the k-means clustering which necessitates a sphere like distribution of the pairs to compare. This drawback has been witnessed clearly in this benchmark study.

The experiments imply that if the data size is small and there is a one-to-one relation between two datasets of matched pairs the greedy agglomerative type clustering works well in terms of precision. On the other hand, as the data size gets larger the EM algorithm improves its performance on the real case scenarios if the similarity metric is sophisticated enough. The idea behind it is probably that the population resembles more to a Gaussian mixture distribution, hence gives better results, since for the experiments EM-algorithm assumes that the underlying probability distributions are in fact normal. However, EM-algorithm has very poor time performance and the experiments imply that for datasets of millions of data, the EM-algorithm would not be feasible at all. k-means clustering, on the other hand, due to its Euclidian distance metric calculations, are not feasible in terms of precision even though it has the best time performance.

Furthermore, our experiments also highlight the importance of the distinguishing power of the different similarity metrics. The more distinguishing power a similarity measure has, the more efficient is the underlying clustering algorithm. That is to say, as the quality of the similarity comparators increases, the algorithms would perform better on the datasets.

To conclude our thesis, the hierarchical clustering outperforms the widely-used k-means clustering as well as the EM-algorithm in terms of the precision. However, the k-means algorithm and if the similarity comparator is sophisticated enough like Jaro-Winkler algorithm, the EM-clustering outperforms the hierarchical clustering in terms of recall. One performance metric can be more important than the other metric in case of different real time scenarios. Since the EM algorithm has very slow rate of convergence and hence very poor time performance, k-means clustering and the hierarchical clustering seem to be two valid candidates in record linkage applications. However, as far as our experiments imply, we can not conclude that there is *one* clustering algorithm which is superior to the other clustering scheme both in terms of recall and in terms of precision.

REFERENCES

[1] L.Gu, R.Baxter, D. Vickers, and C.Rainsford. Record Linkage: Current Practice and Future Directions. *CMIS Technical Report No*. 03/83. 2003.

[2] D. Dey, S. Sarkar, and P.De. A Probabilistic Decision Model for Entity Matching in Heteregonous Databases. *Management Science*, 44(10):1379-1395, 1998.

[3] E. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *IEEE International Conference on Data Engineering*, pages 294-301, 1993.

[4] L. P. Chen, Pauray S. M. Tsai, and Jia-Ling Koh. Identifying Object Isomerism in Multidatabase Systems. *Distributed and Parallel Databases*, 4(2):143-168, 1996.

[5] Y.R. Wang and S. Madnick. The Interdatabase Instance Identification Problem in Integrating Autonomous Systems. In *Proc. Fifth Intl. Conf. Data Eng.*, pages 46-55, 1989.

[6] M.A. Hernandez, and S.J.Stolfo. The Merge/Purge Problem for Large Databases. In *Proc. of 1995 ACT SIGMOD Conf.*, pages 127-138, 1995.

[7] D. Dey, S. Sarkar, and P. De. A Distance-Based Approach to Entity Reconciliation in Heterogeneous Databases. *Management Science*, 44(10):1379-1395, 1998.

[8] P. Christen and T. Churches. *Febrl: Freely extensible biomedical record linkage*, release 0.2 edition, April 2003.

[9] M.A. Hernandez, and S.J.Stolfo. Real-World Data Is Dirty: Data Cleansing and the Merge/Purge Problem. *Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 9-37, Jan. 1998.

[10] L.Gu, and R.Baxter. Adaptive Filtering for Efficient Record Linkage. *Proc. of the Fourth SIAM International Conference on Data Mining,* pp. 477-481, 2004.

[11] W.E. Winkler. Matching and Record Linkage. In Cox et al, editor, *Business Survey Methods.* J. Wiley & Sons Inc., 1995.

[12] A.Elmagarmid, P.G.Iperotis, V. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, NO. 1, January 2007.

[13] M.G. Elfeky, V.S. Verykios, and A.K. Elmagarmid. TAILOR: A Record Linkage Toolbox. *In Proc. of the 18th Int. Conf. on Data Engineering, IEEE*. pp. 17-28, 2002.

[14] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In Proc. *of the sixth ACM SIGKDD Int. Conf. on KDD*, pages 169-178, 2000.

[15] S.Chapman. http://www.dcs.shef.ac.uk/~sam/simmetrics.html. SimMetrics. 2006.

[16] William W. Cohen. www.cs.cmu.edu/~wcohen/Matching-1.ppt

[17] R.Baxter, P.Christen, and Tim Churces. A Comparison of Fast Blocking Methods for Record Linkage. In *Proc. of ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*, Washington DC, USA (2003) 25-27.

[18] T.F. Smith and M.S. Waterman. Identification of Common Molecular Subsequences. *J. Molecular Biology*, vol 147, pp. 195-197, 1981.

[19] M.A. Jaro, Unimatch: A Record Linkage System: User's Manual, technical report. *US Bureau of the Census*, Washington, D.C., 1976.

[20] E.H.Porter and W.E.Winkler. Approximate string comparison and its effect on an advanced record linkage system. In *Proc. of an International Workshop and Exposition – Record Linkage Techniques*, Arlington, VA, USA, 1997.

[21] A.E. Monge and C.P. Elkan. The Field Matching Problem: Algorithms and Applications. *Proc. Second International Conf. Knowledge Discovery and Data Mining*, pp. 267-270, 1996.

[22] W.W.Cohen. Integration of Heterogenous Databases without Common Domains Using Queries Based on Textual Similarity. *Proc. 1998 ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '98)*, pp. 201-212, 1998.

[23] M. Bilenko, R.J. Mooney, W.W.Cohen, P.Ravikumar, and S.E.Fienberg, Adaptive Name Matching in Information Extraction, *IEEE Intelligent Systems*, vol. 18, no. 5, pp. 16-23, Sept./Oct. 2003.

[24] L.Garavano, P.G.Ipeirotis, N.Koudas, and D. Srivastava, *Text Joins in an RDBMS for Web Data Integration, Proc. 12th Int'l Word Wide Web Conf. (WWW12)*, pp. 90-101, 2003.

[25] P.Lehti. Unsupervised Duplicate Detection Using Sample Non-Duplicates. Springer-Verlag Berlin 2006.

[26] I.P. Fellegi, and A.B.Sunter. A Theory for Record Linkage. *Journal of the American*

Statistical Association, 64, pages 1183-1210, 1969.

[27] M.Neiling. Data Fusion with Record Linkage. *In 3. Workshop "F derierte Datenbanken" Magdeburg* 1998. only included in the online-proceedings at http://www.iti.cs.unimagdeburg.de/fdb98/online-proc/

[28] Quinlan, J. R. C4.5: Programs for Machine Learning. *Morgan Kaufmann Publishers*, 1993.

[29] Quinlan, J. R. Induction of Decision Trees. *Machine Learning 1, 1 (Mar. 1986)*, pp. 81-106. 1986.

[30] P. Berkhin. Survey of Clustering Datamining Techniques. *Accrue Software, Inc.* 2002.

[31] J. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, NY. 1975.

[32] J. Hartigan, and M. Wong. Algorithm AS136: A k-means clustering algorithm. *Applied Statistics*, 28, 100-108. 1979.

[33] L.Gu, and R.Baxter. Decision Models for Record Linkage. Data Mining, *LNAI 3755*, pp. 146-160, 2006.

[34] P.Lehti, and, P.Frankhauser. Unsupervised Duplicate Detection Using Sample Non-duplicates. *Journal on Data Semantics VII*, LNCS 4244, pp. 136-164, 2006.

[35] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistical Society*, Series B, vol. 39, 1:1-38. 1977.

[36] T.Hastie, R.Tibshirani, and J.Friedman. The Elements of Statistical Learning. Springer. 2001

[37] S. C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 2:241-254.1967

[38] http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html. Hierarchical Clustering Algorithms.

[39] http://www.cs.waikato.ac.nz/ml/weka/. WEKA.

[40] http://grb.mnsu.edu/grbts/doc/manual/Expectation_Maximization_EM.html. The EM Algorithm for Unsupervised Clustering.

[41] http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/software.htm. Cluster 3.0.

[42] http://www.cs.utexas.edu/users/ml/riddle/. RIDDLE: Repository of Information on Duplicate Detection, Record Linkage, and Identity Uncertainty.