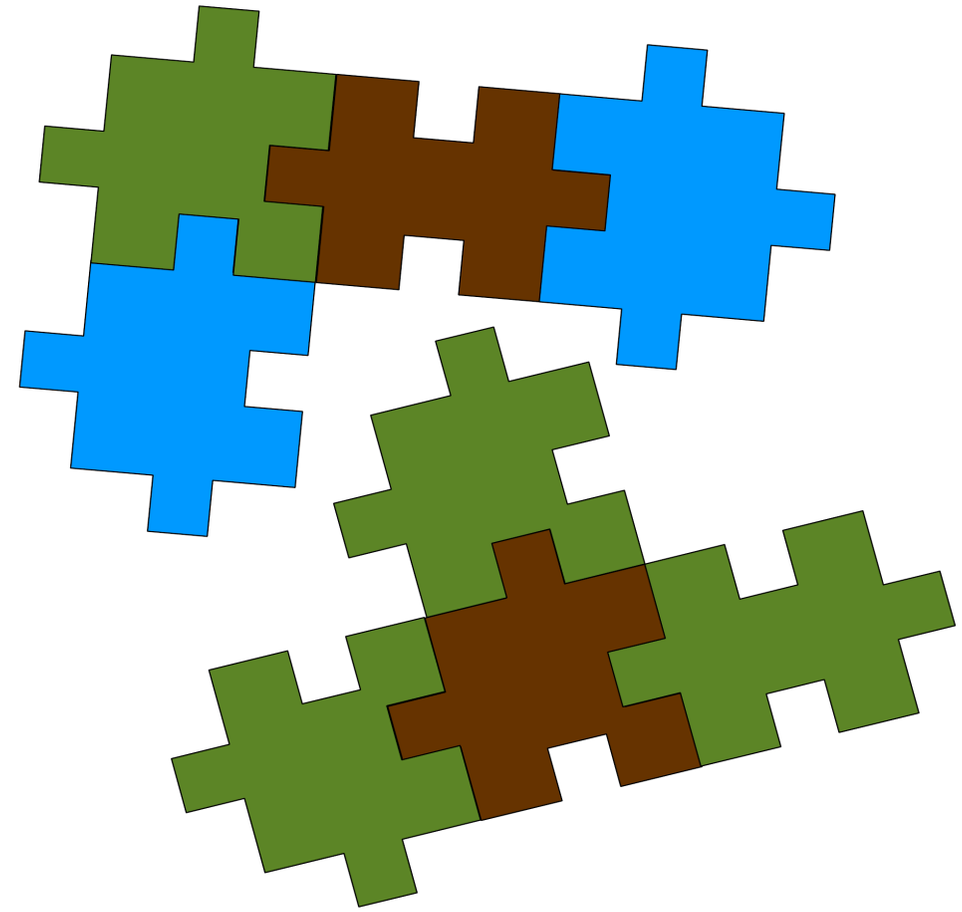
Entity matching

JOHAN DAHLIN



FOI, Swedish Defence Research Agency, is a mainly assignment-funded agency under the Ministry of Defence. The core activities are research, method and technology development, as well as studies conducted in the interests of Swedish defence and the safety and security of society. The organisation employs approximately 1000 personnel of whom about 800 are scientists. This makes FOI Sweden's largest research institute. FOI gives its customers access to leading-edge expertise in a large number of fields such as security policy studies, defence and security related analyses, the assessment of various types of threat, systems for control and management of crises, protection against and management of hazardous substances, IT security and the potential offered by new sensors.

FOI
Swedish Defence Research Agency
Information Systems
SE-164 90 Stockholm

Phone: +46 8 555 030 00
Fax: +46 8 555 031 00

www.foi.se

FOI-R--3265--SE
ISSN 1650-1942

Methodology report
August 2011

InformationSystems

Entity matching

FOI-R--3265--SE

Johan Dahlin

Entity matching

Titel	Entitetsmatchning
Title	Entity matching
Rapportnummer / **Report no**	FOI-R--3265--SE
Rapporttyp / **Report type**	Metodrapport / Methodology report
Utgivningsår / **Year**	2011
Antal sidor / **Pages**	104
Kund / **Customer**	Swedish Armed Forces
Projektnummer / **Project no**	E53200
Godkänd av / **Approved by**	Anders Törne Information Systems
ISSN	ISSN-1650-1942

FOI Swedish Defence Research Agency
Information Systems
SE-164 90 STOCKHOLM

Abstract

This report serves as a review and survey of earlier work in the field of entity matching as well as current software implementations in this area. Entity matching uses string matching methods known as field metrics to find similar text strings that could correspond to similar names or addresses. The outputs from these field metrics are often used with different classification methods to determine if the strings (or the entire entry the strings are a part of) are matching or unmatching. These classification methods include both supervised and unsupervised methods originating in statistics and machine learning.
This report proposes using other classifiers including vertex similarity and text mining-methods to generate additional evidence that two entities match. Vertex similarity is studied in network analysis and aims to identify nodes sharing a large fraction of common neighbors, indicating that the entities have similar social or communication networks. Text mining-methods are useful in finding similar documents and other written longer texts, indicating that two entities have the same language usage or deal with the same topics. Some small experimental evaluations are offered using citation data from two different sources to test these two methods of finding similar entities. Furthermore, the report proposes methods based on data fusion to combine these classifiers with the traditional field metrics into an ensemble.

Keywords

Record matching, duplicate entry detection, entity resolution, vertex similarity, ensemble classification, data fusion, information fusion

Sammanfattning

Denna rapport innehåller en genomgång och diskussion av tidigare arbeten inom entitetsmatchning samt aktuella implementationer av dessa i form av olika programvaror. Entitetsmatchning använder strängmatchande metoder som ofta kallas fältmatchningsmetoder för att hitta liknande textsträngar som kan bestå av exempelvis liknande namn eller adresser. Dessa fältmetoder används ofta tillsammans med olika klassificeringsmetoder för att avgöra om strängar (eller hela den posten som strängarna är en del av) är matchande eller inte. Dessa klassificeringsmetoder innefattar både övervakade (supervised) och oövervakade (unsupervised) metoder som har ursprung i statistik och maskininlärning.
Rapporten föreslår att man även kan använda andra typer av klassificerare som inkluderar nodlikheter och text mining-metoder för att generera ytterligare bevis på att två entiteter är matchande. Nodlikhet studeras i nätverksanalys och syftar till att identifiera noder som delar en stor andel gemensamma grannar, vilket visar att entiteterna har liknande sociala nätverk eller kommunikationsvanor. Text mining-metoder är användbara för att hitta liknande dokument och andra skriftliga längre texter, vilket tyder på att två entiteter har samma språkbruk eller skriver om samma ämnen. Några små experimentella utväderingar presenteras även i rapporten, där de föreslagna metoderna appliceras på citeringsdata från två olika källor. Slutligen diskuteras om metoder från datafusion kan användas för att kombinera dessa nya föreslagna metoder tillsammans med traditionella fätmatchningsmetoder för att skapa en ensemble av klassificerare.

Nyckelord

matchning av dataposter, identifiering av dupletter, entitetsmatchning, ensembleklassificering, nodlikhet, datafusion, informationsfusion

Contents

1 Introduction **13**

2 Field matching **19**
2.1 Character-based similarity metrics 19
2.1.1 Edit distance 19
2.1.2 Jaro-Winkler metric 20
2.1.3 Comparison 21
2.2 Token-based similarity metrics 21
2.2.1 WHIRL metric 22
2.2.2 Comparison 23
2.3 Recommendations 24

3 Matching models **27**
3.1 Fellegi-Sunter model 27
3.1.1 Naive method 29
3.1.2 EM-based method 29
3.2 Machine learning methods 31
3.2.1 Decision Trees 31
3.2.2 Support Vector Machines 35
3.2.3 Clustering 37
3.3 Recommendations 38

4 Implementations **41**
4.1 D-dupe 41
4.2 FEBRL 42
4.3 FRIL 44
4.4 Record Linkage (R Package) 45
4.5 Others 45
4.6 Recommendations 46

5 Graph and text mining-based methods **47**
5.1 Vertex similarity 47

5.2 Latent Semantic Analysis 48

6 Proposed frameworks **51**
6.1 Weak classifiers 51
6.1.1 Statistical learning-based methods 51
6.1.2 Graph-based methods 53
6.1.3 Text mining-based methods 53
6.2 Sequential blocking method 54
6.3 General ensemble method 55
6.3.1 Weighted combination 56
6.3.2 Meta-combination 59
6.4 Recommendations 60

7 Results **63**
7.1 Fellegi-Sunter method with blocking 63
7.2 Graph-based methods 64
7.3 Text mining-based methods 65
7.4 Ensemble methods 67
7.5 Discussion and recommendations 70

8 Concluding remarks **73**

Bibliography **75**

A Tutorial on the RecordLinkage package **79**
A.1 Fellegi-Sunter 80
A.2 Supervised classification 82
A.3 Unsupervised classification 83

B R-implementations **85**
B.1 WHIRL metrics 85
B.2 Blocking strategy 88
B.3 Graph-based methods 89
B.4 Text mining-based methods 90
B.5 Ensemble methods 93

C Python-scripts **101**

C.1 Extract author names 101
C.2 Extract abstracts 102

1 Introduction

Integration of data from several different data sources is an important problem as more and more information becomes available for analysis. One major problem is that sources often are heterogeneous, i.e. the information often has different formats, spellings, and may contain erroneous or conflicting information. This occurs in many different applications, including data mining, census, marketing and other similar fields. Therefore this problem has been studied in many different fields and is thereby also known by many different names. Names used include e.g. record linkage (statistics), database hardening (computer science), data integration (data mining), and entity matching. The latter term is adopted in this report for describing the task of finding entries in a number of data records that corresponds to the same entity (individual).

This report aims to present and summarize the most important and commonly used methods from the different scientific fields working with this problem. The scenario considered is that data from a number of heterogeneous sources should be merged into one. This merger should be accomplished with a minimum of duplicate entries and entries concerning the same entity should be merged so that all information regarding a single entity are contained within one entry. To complete this matching, all possible pairs must be compared between the records. This results in a very large number of comparisons that requires a lot of computational effort.

It is inefficient and time consuming to compare all the possible pairs of data entries. To decrease the number of comparisons needed, some *blocking strategy* is usually applied on the data material. A blocking strategy limits the number of comparisons by e.g. requiring that the day of birth must be the same in two data records before considering them for comparisons. Finding a suitable blocking strategy is difficult and is often done manually . Choosing a bad blocking strategy results in bad results as many comparisons are disregarded, which could result in bad accuracy.

Purpose and intended reader The purpose of this report is to serve as a reference to state of the art research and implementations of entity matching. The intended reader is a FOI researcher who works on information fusion tools, but the report should also be of interest to technically oriented personnel in the Swedish Armed Forces and at FMV.

Motivation and background Continuing by presenting an outline of how record linkage works and thereby how duplicate data entries can be detected. In heterogeneous data sources, data corresponding to the same real-world property can be organized in different ways. A simple example of the problem with data integration is that the date-of-birth can be written as a combined field with MMDDYY (Month, Day, and Year) or as three different fields for month, day, and year. This problem is remedied by a simple data transformation that combines several different data fields into one.

Another more difficult and related problem is the matching of strings, which can describe names, addresses, affiliations, and other important characteristics. These strings can be misspelled, contradicting, old, etc. therefore making it impossible to match them by using normal (crisp) operators such as the equality operator. Some fuzzy metrics are therefore needed to counter the problem with different spellings, word orders, and missing/extra inserted words. These *field metrics* are used to compare two strings and estimate how similar they are with each other.

Figure 1.1: Small example of matching two data fields between two records. The first pair is an obvious match using the equality operator with a small transformation. The second record has some misspellings and errors and does not match using the equality operator. Therefore some other fuzzy operator is needed to compare these fields.

The problem with string metrics is shown schematically in *Figure 1.1* where two pairs of entries are compared by equality operators. The first pair is equal if a small data transformation is applied, the second pair is not considered equal by the equality operator but is in fact probably equal. The differences between the entries are the spelling of the last name and some erroneous input of the birth month. This problem and proposed solutions are discussed in the next two chapters of this report.

A more detailed description of the full method and the problem is found in *Figure 1.2*, where two data records are compared. Each *data record* consists of a number of *data entries* each containing some *data fields*. In this example, two records containing the data fields name, address, phone number, and day-of-birth are to be combined using this record linkage method. Each pair of matching fields are compared using some of the fuzzy string metrics. These metrics are used as an input into an entry matching method which classifies each pair of entries as matching, possible matching, or unmatching. To be able to classify these entries some classification regions are needed. These regions are often estimated from training data or by the data itself using some entity matching method.

Previous and related work Entity matching builds upon the earlier mentioned efforts in statistics, data mining, bioinformatics, and other related fields. The first mention of record linkage, i.e. to combine information about a person from several different sources is due to Dunn Halbert (1946). This idea was later formalized in terms of quantitative measures by Newcombe *et al.* (1959) and then by the first matching model (often used today) in Fellegi & Sunter (1969). Later works have focused on using different forms of machine learning to classify pairs of entities by the use of field matching metrics. A good account of these efforts and the state of research in the field is given by

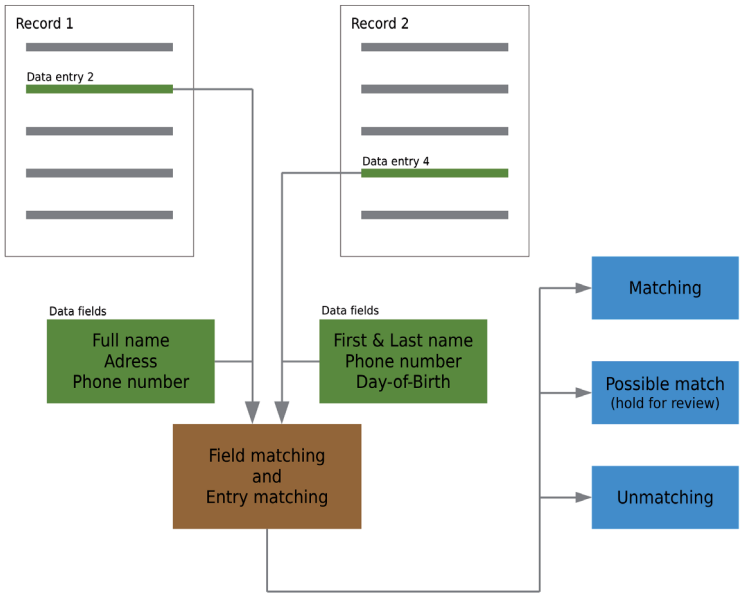


Figure 1.2: Schematic overview over a general entity matching procedure including field matching metrics and automatic entry matching. The problem concerns finding duplicate entries in some data records using metrics to compare similarity. The result is pairs of entries that are classified as: matching, unmatching, or possible matches.

Elmagarmid *et al.* (2007).

Ensemble methods have a long history in the study of neural networks and other classification methods. Some of the earliest work in ensemble of classifiers was done in the form of boosting, the most commonly used method for this is *AdaBoost* which is due to Freund & Schapire (1997a). Later works in ensemble classification are summarized and discussed in e.g. Hastie *et al.* (2003), Polikar (2006), and Rokach (2010).

The graph-based entity matching methods are presented in Bhattacharya & Getoor (2006) and the used vertex similarity measures have been extensively studied in the field of network analysis, see e.g. Leicht *et al.* (2006) for more information and comparisons. Text mining is an additional field, from which methods and results are used in this report. Primarily, Latent Semantic Analysis is used which is due to Deerwester *et al.* (1990).

To the author's knowledge, no previous effort has been given the integration of ensemble classification methods in entity matching, to merge the information found in several different sources (and of several different types). Some previous work done to combine the classifiers in an ensemble for entity matching is presented in Chen *et al.* (2009) using graph-based methods, as well as using

other methods as in e.g. Zhao & Ram (2005) and Shen *et al.* (2007). The major difference between the proposed methods in this report and from previous work is the use of several different methods to compare entries. In previous works, only e.g. different field matching metrics are used to compare the data in two entries.

Contribution and proposed methods Two new methods for entity matching are proposed and discussed in this report: (i) using a number of weak classifiers in sequence as a blocking strategy, and (ii) combining classifiers using voting or meta-learners. The two latter methods are commonly used in data fusion and ensemble classification methods.

As the citation data is combined from a number of different sources, abstracts and co-author information are useful for matching identities. Methods from social network analysis and text mining (analysis) are applied for finding similar pairs of entries. By using some simple faster classifier to screen the set of all possible pairs a functional blocking strategy is found. This is done in sequence with increasing accuracy, as more accurate and complex classifiers have a higher computational complexity. The resulting candidate pairs are analyzed by using standard methods from record linkage or entity matching.

The second proposed method uses the full information from an ensemble of classifiers using some combination rule or a meta-learner. The simplest combination rule is the majority voting rule with an optional weighting of each classifier. Other more advanced combiners are found using Dempster's rule-of-combination (from evidence theory), Bayes' rule, and variance-minimizing linear programming. Meta-learner methods include *stacking*, where the output from an ensemble of weak classifiers are used as an input to a second stage of classifiers. This meta-learner is trained using the classifications found by the weak classifiers in some training set and also the training data itself. Therefore enabling the meta-learner to use different combination of weak classifiers for different types of data.

These ensemble classification methods are quite computational intensive and some blocking strategies are required as a pre-processing step to limit the computational effort. These strategies can e.g. require that a majority of the characters in some strings are common or by applying some field matching metric. To the author's knowledge, no previous work has been done in entity matching using ensembles of weak classifiers from such diverse fields as proposed in this report.

Evaluations and results The proposed methods are evaluated using some citation data gathered by Johansson *et al.* (2011), which contains a co-author network and abstracts from each paper presented at the annual FUSION conference. The actual data set is a combination of citation information obtained from IEEE Xplore and Thomson Reuters' ISI Web of Knowledge, i.e. two heterogeneous data sources. The aim of the evaluations is to find duplicate authors that have different variations of their name, i.e. other spellings, initials, and similar problems. A common problem is e.g. that some authors are cited with two initials in some papers and only one initial in another. As these names are

spelled differently most bibliographical software creates two distinct entities for this single author. The aim is to identify these duplicates and merge them to have only a set of authors, each corresponding to a unique entity.

The sequential method with graph-based and text mining-based methods is evaluated and compared with a common blocking strategy (requiring equal last name). The graph-based method requires some similarity in the co-author network structure, i.e. that the pair of nodes share a large fraction of common co-authors (neighbors) indicating evidence for the hypothesis that the pair of nodes correspond to a single entity. The text mining-based method analyzes the topics of the paper abstracts to cluster the papers into groups with common contents. The set of possible pairs is thereby limited to all the pairs within each group, using the assumption that authors often write a few papers within the same topic. This also greatly reduces the number of pairs to consider with standard matching methods.

The results indicate that the sequential method is a good blocking strategy for data material containing more advanced data structures than text strings. It is however the case that the three different sequential methods find different candidate pairs. Therefore, it is recommended to use the union of these three considered methods as the input to e.g. the Fellegi-Sunter method with the Expectation Maximization-algorithm in order to find matching entities.

The full ensemble method is only tested using some minor studies on a small data set. This serves only to illustrate the usefulness and potential of this approach and thereby justifying future and continued work with this method. The main problem and reason for the limited study is time constraints and difficulties in finding appropriate labeled data sets for evaluating the performance and accuracy of ensemble methods. In the following simulation experiments field matching metrics, network structure similarity, and abstract similarity are combined as the input of a simple classifier. An implementation is proposed to calculate each of these quantities and some preliminary results using a Support Vector Machine as a meta-learner is presented indicating some tendencies for better performance compared with the sequential method. Despite the limited comparison, it is the conclusion that this ensemble method is the most promising and thereby merits future research effort.

Disposition This report continues with a presentation of current field matching metrics and matching methods from the theory of record linkage. These chapters introduce methods for comparing text strings and determining classification regions. Some learning methods including: classification trees, clustering, and support vector machines, are also presented and discussed. The report continues with a presentation of current implementations for entity matching and recommendations regarding their use. A framework for matching entities in citation data and similar text-based network data is presented using methods from ensemble classification. This framework is evaluated using some citation data and the results are discussed in the following chapters. The appendices contain a tutorial of some software packages and some source code used in the simulation studies.

2 Field matching

The first step towards creating a method for matching similar data entries is to quantify the similarity between different types of data. Many well-known measures already exist for quantifying similarities in integers, floats and sets. These include the usual norm measures (Manhattan and Euclidean distances), set measures (the Jaccard and cosine measures), and others. Comparing the similarity of these types of data is therefore already well-studied and good methods exist in both clustering and classification theory.

Another type of data commonly encountered is text strings, which are non-trivial to compare using standard methods. It is possible to use set measures to compare the similarity between strings, by treating each character as a member in a set comprised by the text string. However, these type of methods heavily penalize misspellings of words and are therefore not suitable for our problem.

To counter the problems of using traditional measures for the studies of sets, some other methods have been developed during the last few decades. These methods are all defined as *field matching techniques*, as they are used to match individual fields in a data set to each other. Field matching methods can be divided[1] into character-based, token-based, and phonetic similarity metrics. We continue by reviewing the most promising methods in the two[2] former metric types. This section follows the presentation in Elmagarmid *et al.* (2007) closely with some additions from other sources.

2.1 Character-based similarity metrics

The first type of similarity measures are concerned with comparing the characters used in two different strings. We present the two most commonly used metrics in this subsection: edit distances and the Jaro-Winkler metric.

2.1.1 Edit distance

Perhaps the simplest of character-based methods is the *Levenshtein edit distance* presented in Levenshtein (1966). This metric is related to the Hamming distance used in information theory. This measure compares two strings by the number of edits needed to transform one string into the other. Denote the two strings s_1 and s_2, then the edit distance is the minimum amount of operations needed to transform s_1 into s_2 or vice versa. The allowed *edit operations* are:

[1] This division into metric families is due to Elmagarmid *et al.* (2007).

[2] Phonetic methods are not treated as they depend heavily on the language considered, i.e. methods developed for e.g. American English does not necessarily perform well for the Nordic languages. Therefore this method is considered impractical in comparing names in a more globalized world in which cultures and people (thereby also names) tend to increasingly mix.

- *insertion* of a character into a string,
- *removal* of a character in a string,
- *replacement* of a character with a different character.

To calculate the edit distance, a dynamic programming problem is usually solved with a complexity of $\mathcal{O}(|s_1||s_2|)$. However, there exist more elaborate versions of the algorithm which limit the complexity to $\mathcal{O}(k|s_1|)$ or similarly $\mathcal{O}(k|s_2|)$ for some k such that the edit distance between s_1 and s_2 is less than k.

Other refinements to the edit distance includes gaps, which allows for a smaller number of operations in strings with left-out words. For example, using gaps a single operation is needed to transform *Sven Anders Svensson* into *Sven Svensson*. This is done by just removing the additional word (the gap) using a single operation, i.e. removing or adding Anders to transform the strings. The cost of this operation can vary depending on the location of the gap, usually higher costs are given to gaps in the beginning and end of strings than in the middle.

2.1.2 Jaro-Winkler metric

Another common similarity measure is known as the *Jaro-Winkler metric* presented in Winkler & Thibaudeau (1987), which is used for comparing short strings such as names. We begin by discussing the simpler *Jaro metric* due to Jaro (1978), which is calculated as follows:

1. Find the length of each string, $n_1 = |s_1|$ and $n_2 = |s_2|$.
2. Find the number of common characters c shared between the two strings. A common character fulfills the following:

$$s_1[i] = s_2[j], \qquad |i - j| \leq \frac{1}{2} \min\{n_1, n_2\}. \tag{2.1}$$

3. Find the number of possible transpositions, t, which is the number of common characters for which $s_1[i] \neq s_2[i]$ where $i = 1, 2, \ldots, c$.
4. The Jaro metric, $\mathsf{J}(s_1, s_2)$, is given by

$$\mathsf{J}(s_1, s_2) = \frac{1}{2}\left(1 + \left[\frac{n_1 + n_2}{n_1 n_2}\right] c - \frac{t}{2c}\right). \tag{2.2}$$

The complexity of this algorithm is $\mathcal{O}(n_1 n_2)$ and is due to the calculation of the number of common characters. A common extension of the Jaro metric is the Jaro-Winkler metric due to Winkler & Thibaudeau (1987), which gives a higher weight to prefix matches by the following

$$\mathsf{JW}(s_1, s_2) = \mathsf{J}(s_1, s_2) + p \max\{l, 4\}\left(1 - \mathsf{J}(s_1, s_2)\right), \tag{2.3}$$

where $p \in [0, 0.25]$ is a factor[3] controlling how the score is increased for having common prefixes, l is the length of the longest common prefix of s_1 and s_2. As previously stated, this method is well suited for comparing names of all types and does not have the drawback of the phonetic family of measures, which is strongly language dependent.

2.1.3 Comparison

The different character-based metrics are best understood by some comparative examples. In *Table 2.1*, some simple examples of misspellings and different formatted strings are compared using the edit distance, Jaro, and Jaro-Winkler metrics. The three metrics find a large similarity between two pairs of strings "Sven Svensson/Sven Svenson" and "Svensson, Sven/Svensson, S". The metrics does not however find any larger similarity between the other three examples in the table.

String 1	String 2	L	J	JW
Sven Svensson	Sven Svenson	1	0.974	0.985
Svensson, Sven	S Svensson	8	0.790	0.811
Svensson, Sven	Svensson, S	3	0.923	0.957
Division of Information Systems	Information Systems Division	21	0.710	0.710
Division of Information Systems	Information Systems	12	0.713	0.713

Table 2.1: Small comparison between some text strings using the Levensthein (edit distance), Jaro, and Jaro-Winkler metrics.

The more advanced metric, Jaro-Winkler finds larger similarity between the three first comparisons in the table. This is as expected as Jaro-Winkler is specially design to weight the first letters in a name higher than the ending of names, i.e. this metric handles initial vs. full first name well. The Jaro and Jaro-Winkler metrics are also designed for matching names and does not perform well with other types of strings, such as organizational names. For these types of text strings, some other measures are needed and token-based metrics are therefore discussed in the following section.

2.2 Token-based similarity metrics

In comparison with character-based methods, token-based methods are more robust to errors in input and spelling. These methods are also better in identifying rearranged and missing words. In this subsection, we discuss the WHIRL metric and two improved versions of this metric using q-grams and by relaxing the assumption of strict (crisp) equality between phrases.

[3] A common choice for this factor is $p = 0.1$ and this value is used in the following examples and simulation experiments.

2.2.1 WHIRL metric

The most promising and commonly applied method from this family is the *WHIRL* metric due to Cohen (1998) which adopts a weighting scheme called *tf-idf* (term-frequency and inverse-document-frequency) with the well-known cosine measure. Each word in a string is assigned a weight using the following expression

$$v_s(w) = \log(\mathsf{tf}_w + 1)\log(\mathsf{idf}_w), \tag{2.4}$$

where tf_w is the frequency of the word w in the entire data set and idf_w is the inverse fraction of entries in the data set in which the word w appears

$$\mathsf{idf}_w = |D|n_w^{-1}, \tag{2.5}$$

where n_w is the frequency of the word w occurring in the data set D. Rare items are therefore given a large weight and common items are given a smaller weight, indicating larger and smaller importance. These weights are used with the cosine measure to find the similarity between two strings as

$$\mathsf{sim}(s_1, s_2) = \sum_{j=1}^{|D|} \frac{v_{s_1}(j)v_{s_2}(j)}{||v_{s_1}||_2||v_{s_2}||_2}, \tag{2.6}$$

where (as above) s_1 and s_2 are two strings, $||\cdot||_2$ denotes the Euclidean norm, and the weights are computed using the relation above in Eq. (2.4). As previously discussed, the main advantage of this method is that it does find similarity between strings with missing and rearranged words. The main drawback is the sensitivity for some spelling errors. For example the strings "Computer Science Department" and "Deprtment of Computer Scence" have zero similarity.

2.2.1.1 WHIRL metric with q-grams

To counter this problem, Gravano *et al.* (2003) extended the WHIRL system using q-grams, which are substrings of q characters common to both strings. The aim is to find long q-grams, indicating that the two strings are similar. This new method does find a non-zero similarity in the previous example and also performs well with insertion and deletion of words.

This method is therefore most useful in comparing for example names of organizations and titles of documents. Gravano *et al.* (2003) suggests that $q = 3$ is a good choice and the change in the previously discussed WHIRL metric is just by changing the words w into q-grams u_q and repeat the same calculations.

2.2.1.2 Soft WHIRL metric

Another improvement to the WHIRL metric is presented in Bilenko *et al.* (2003) by relaxing the summation given in Eq. (2.6). This is done by summing

over all pairs of phrases that are similar (or identical) by some field matching metric. A usual choice is the Jaro-Winkler metric with a limit value, $\theta = 0.9$, for similar phrases. The resulting summation is the set of close phrases,

$$\mathsf{close}(\theta, s_1, s_2) = \{w \in s_1 : \exists v \in s_2 \text{ and } \mathsf{JW}(w, v) \geq \theta\}, \tag{2.7}$$

where $\mathsf{JW}(\cdot)$ denotes the Jaro-Winkler metric. Let $c(w, t)$ denote a weight calculated by

$$c(w, s_2) = \max_{v \in s_2} \mathsf{JW}(w, v), \tag{2.8}$$

which is added into the resulting generalized version of Eq. (2.6), found as

$$\mathsf{sim}(s_1, s_2) = \sum_{w \in \mathsf{close}(\theta, s_1, s_2)} \frac{v_{s_1}(w) v_{s_2}(w) c(w, s_2)}{||v_{s_1}||_2 ||v_{s_2}||_2}, \tag{2.9}$$

with notation as above. This measure is more robust to spelling errors, insertions and deletions of words as only some partial similarity is required to be included in the summation resulting in the field similarity.

2.2.2 Comparison

The different token-based metrics are best understood by some comparative examples. In *Table 2.2*, some simple examples of misspellings and different formatted strings are compared using the originial WHIRL-metric and the two proposed improved versions. The three metrics does not find any stronger pairings in these shorter strings, as the token-based metrics are more useful for longer text strings and not names. The metrics however identifies strong connections in the pair Division of Information Systems/Information Systems.

String 1	String 2	O	Q	S
Sven Svensson	Sven Svenson	0.398	0.330	0.398
Svensson, Sven	S Svensson	0.356	0.271	0.354
Svensson, Sven	Svensson, S	0.356	0.271	0.354
Division of Information Systems	Information Systems Division	0.336	0.219	0.336
Division of Information Systems	Information Systems	0.517	0.408	0.517

Table 2.2: Small comparison between some text strings using the three versions of the WHIRL-metric: Original, Q-gram (with $q = 3$), and Soft (with the Jaro-Winkler metric and limit 0.90).

The q-gram version seems to perform worse in comparison with the original and softer versions. Larger data sets are required to evaluate the measures in more general terms, as the weighting scheme does not work for a smaller collection of text strings (as in this case). In general, the soft-WHIRL measure has been performing best in larger comparisons between these measures (see

the following discussion regarding recommendations). As the WHIRL-measure is only implemented in Java, some effort has been given to implementing the three different measures in R, see *Appendix B.1* for details.

2.3 Recommendations

The aim of this section is to provide some guidance to recommended field matching metrics for some common situations. This section is based on previous benchmarks and simulation experiments conducted by various authors, e.g. Cohen *et al.* (2003) and Yancey (2005).

Field matching methods can be divided into the two categories discussed in this chapter with the addition of phonetic metrics. As previously discussed, phonetic metrics are language dependent and the metric must be adapted for each individual language. For general methods, therefore only character-based and token-based metrics are useful, both individually or in combination using hybrid methods.

To begin with, it is worth to mention the problem with the lack of common benchmark data sets. As studies of metrics tend to use different benchmarks for matching fields, it is difficult to draw any general conclusions in many cases. Some studies however indicate that some measures are better than others, whereas other studies indicate that no single metric is better than others in general. In this section, we provide an overview of a few studies and their results, but urge the reader to keep in mind that the results of these studies are *strongly* dependent on the type of data used for comparisons.

In Cohen *et al.* (2003), the following metrics are compared: Levensthein, Jaro, Jaro-Winkler, Jaccard, WHIRL, Jensen-Shannon, Simplified Fellegi-Sunter, soft WHIRL, and recursive matching. Some of these methods have not been discussed in this report and readers are referred to the paper discussing this study for more details of each method. The methods are evaluated using a large number of different data sets ranging from animal and bird names, to US. census data. The WHIRL and Jaro-Winkler metrics are discussed as having good average performance on the data sets used in this study. The authors recommend soft WHIRL as a good general choice and also discuss the importance of using learning to combine several distance measures. This latter discussion is continued in the following sections of this report, as it is the cornerstone of matching identities as well as a future important research field.

Another study comparing different field matching metrics is found in Bilenko *et al.* (2003), using the same data sets as the previous paper. This paper verifies the same conclusions and continues by comparing methods for learning to combine different metrics, this is further discussed in the next chapter.

Yancey (2005) offers a comparison of several different metrics used on name data from the US. Census Bureau. The paper compare several types of Jaro-Winkler metrics as well as a number of different edit distances. The results indicate that Jaro is the best metric when used in combination with the prefix

adjustment (Jaro-Winkler) and that standard edit distance is the best variation of edit distances. The paper also propose a method to combine the Jaro-Winkler and standard edit distance metrics, however results indicate that this does not yield any large improvements.

In summary, Jaro-Winkler metrics are commonly recommended for matching names, soft WHIRL is recommended for long text strings as titles of papers, affiliations and similar fields. For digits and number data, no specific metrics have been developed so far. Usually digits are therefore compared using string metrics, e.g. edit distances or Jaro metrics. This does not always yield good results and more effort is needed in developing metrics for digit data. Other suitable measures could be norms of different kinds, depending on which kinds of errors that are present in the data. If typos and such are present then edit distances are preferable, but if the digits are estimated numbers then norms are the better choice. As previously stated, no universal best choice is available in string comparisons and therefore it is more important to use good combinations of methods than choosing a particular metric, more on this in the following sections.

3 Matching models

The previous section discussed different methods for comparison of text strings, names, and other common data types. Using these methods it is possible to quantitatively calculate the similarity between two strings. The remaining problem is to find the classification regions (limit values) that specify when two entries are to be merged and are distinct.

We discuss some different methods: both unsupervised and supervised used in conjuncture with the field matching metrics from the previous section to find models for matching data entries. The aim is to present some different methods used in implementations and discuss these from a practical standpoint, thereby producing recommendations for when to use which model.

3.1 Fellegi-Sunter model

The first proposed model for *record linkage* was a probabilistic method using classification into one of two different classes[1]: *matching*, M, and *unmatching*, U. For the remainder of this section, we adopt the following notation which is due to Fellegi & Sunter (1969). Assume that two data entries A and B contain some elements $a \in A$ and $b \in B$ and that the set of ordered pairs

$$A \times B = \{(a, b); a \in A, b \in B\}, \tag{3.1}$$

is the union of the two sets containing matching and unmatching entries,

$$\begin{aligned} A \times B &= M \cup U, \text{ where} \\ M &= \{(a, b); a = b, a \in A, b \in B\}, \\ U &= \{(a, b); a \neq b, a \in A, b \in B\}. \end{aligned} \tag{3.2}$$

The elements of each entry could for example be name, address, telephone number, or shoe size. These entries assume some value denoted by the functions $\alpha(a)$ and $\beta(b)$. Given some *comparison function*, γ, we define a comparison vector between the n entries in a and b as the following

$$\gamma\left[\alpha(a), \beta(b)\right] = \left\{\gamma^1\left[\alpha(a), \beta(b)\right], \ldots, \gamma^n\left[\alpha(a), \beta(b)\right]\right\}. \tag{3.3}$$

In our example, this vector should correspond to how e.g. name, address, phone number and shoe size match using some field matching method. Originally, a binary model was used with an equality operator comparing fields, therefore the similarity assumes only values 0 and 1. Each element in this vector is therefore a quantitative value of how similar each field is.

[1] Sometimes a third class is also used called *possible matches* and holds the pair for clerical review)

The Fellegi-Sunter model assumes that the comparison vectors are generated by one of two different (overlapping) probability distributions. These distributions are unknown beforehand and an important part of the method is to estimate the distribution functions to be able to identify vectors drawn from each distribution. In our case, the two distributions correspond to matching and unmatching pairs of entries and the overlapping part between the distributions are the pairs that are returned for manual review (shown in *Figure 3.1*).

The remaining problem is to find some rule telling when the fields of the data entries are similar enough such that the data entries themselves are considered duplicates. Continuing by defining the conditional probabilities of γ, given that $(a,b) \in M$ by $m(\gamma)$ and $(a,b) \in U$ by $u(\gamma)$ as

$$\begin{aligned} m(\gamma) &= \mathbf{P}\left\{\gamma\left[\alpha(a),\beta(b)\right] \middle| (a,b) \in M\right\}, \\ &= \sum_{(a,b)\in M} \mathbf{P}\left\{\gamma\left[\alpha(a),\beta(b)\right]\right\} \mathbf{P}\left[(a,b)\middle|M\right], \end{aligned} \tag{3.4}$$

$$\begin{aligned} u(\gamma) &= \mathbf{P}\left\{\gamma\left[\alpha(a),\beta(b)\right] \middle| (a,b) \in U\right\}, \\ &= \sum_{(a,b)\in U} \mathbf{P}\left\{\gamma\left[\alpha(a),\beta(b)\right]\right\} \mathbf{P}\left[(a,b)\middle|U\right]. \end{aligned} \tag{3.5}$$

Using these two definitions it is possible to show that the decision rule has the following form,

$$(\alpha(a),\beta(b)) \in \begin{cases} M & \text{if } \mathbf{P}(M|\gamma) \geq \mathbf{P}(U|\gamma) \\ U & \text{otherwise} \end{cases} \tag{3.6}$$

which can be rewritten using Bayes theorem as,

$$(\alpha(a),\beta(b)) \in \begin{cases} M & \text{if } l(\gamma) = \frac{\mathbf{P}(\gamma|M)}{\mathbf{P}(\gamma|U)} \geq \frac{\mathbf{P}(U)}{\mathbf{P}(M)} \\ U & \text{otherwise} \end{cases} \tag{3.7}$$

where $l(\gamma)$ is the likelihood ratio and also the limit between the two sets M and U.

In *Figure 3.1*, the probability distributions for matching and unmatching are shown together with the region corresponding to possible matches. The distributions are approximated using some of the methods described below: naive and EM-algorithmic methods.

The overlap between the distributions are corresponding to the usual alpha error (False Positive) and beta error (False Negative) in statistical hypothesis testing. It is important to note that some data sets could return diffuse probability distribution functions and thereby also large overlaps. In these cases, the result will be uncertain and it is therefore important to observe e.g. the Kullback-Leibler divergence to determine the amount of overlap between the two distributions.

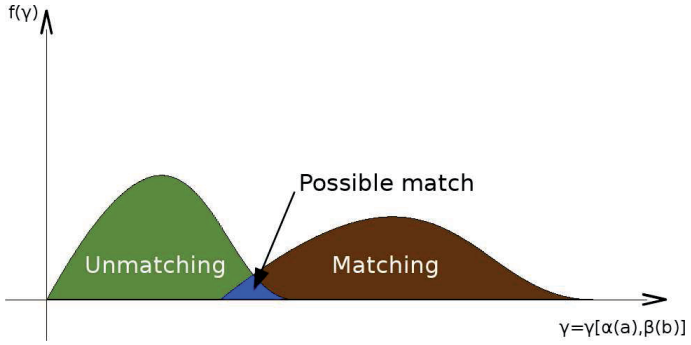


Figure 3.1: The probability density functions for the two different classifications: matching and unmatching. The intersection between the two density functions is the region corresponding to possible matches. If the distributions are clearly separated, a comparison $\gamma[\alpha(a), \beta(b)]$ between two entries a and b is classified by the distribution the vector has support in. In this sketch, only one field is compared and the main problem is to find a limit value, γ^*, such that the entries are matching if the comparison value is larger than this and unmatching otherwise.

3.1.1 Naive method

As the values of $\mathbf{P}(\gamma|M), \mathbf{P}(\gamma|U), \mathbf{P}(U)$, and $\mathbf{P}(M)$ are not known, we are forced to make some simplifying assumptions to compute these unknown quantities. The most common assumption is that the conditional probabilities, $\mathbf{P}(\gamma|M)$ and $\mathbf{P}(\gamma|U)$, are independent and therefore can be rewritten as

$$\mathbf{P}(\gamma|M) = \prod_{i=1}^{n} \mathbf{P}\left(\gamma^i\left[\alpha(a), \beta(b)\right] \middle| M\right), \tag{3.8}$$

$$\mathbf{P}(\gamma|U) = \prod_{i=1}^{n} \mathbf{P}\left(\gamma^i\left[\alpha(a), \beta(b)\right] \middle| U\right), \tag{3.9}$$

which can be obtained using a training set of data by calculating the conditional probabilities, $\mathbf{P}\left(\gamma^i\left[\alpha(a), \beta(b)\right] \middle| \cdot\right)$, using the supplied labels. This method was used in applications for many years, until recently when the assumption of conditional independence was questioned.

3.1.2 EM-based method

To counter this, new methods were developed that did not rely on the assumption of independence to calculate the likelihood ratio in Eq. (3.7). A suitable method which also is unsupervised (does not need any training set of data) is using the Expectation-Maximization (EM) algorithm. Widely used and well-known, the EM-algorithm is used as a maximum likelihood estimator for some

parameter θ of a parametric probability distribution. The algorithm is often used to classify a mix of observations from two different Gaussian distributions. As it is assumed that comparison vectors are drawn from either the Gaussian distribution[2] corresponding to matching or unmatching pairs of entries, the EM-algorithm is most useful and promising for classifying the comparison vectors. The algorithm is divided into two steps (thereby its name): Expectation and Maximization, which are repeated until some predetermined accuracy has been achieved.

The simplest version of using the EM-algorithm to estimate the likelihood ratio in Eq. (3.7) is given by Jaro (1989) using a binary comparison function, $\gamma^i = 1$ if the fields match and $\gamma^i = 0$ otherwise for all $i = 1, 2, \ldots, n$. The aim is to estimate $\theta = (m, u, p)$ where p is the proportion of matched pairs in Q and m and u are the conditional probabilities as previously defined. Let Q denote the set of all pairings between entries in A and B, i.e. all pairs in $A \times B$.

The first step of the EM-algorithm is concerned with finding the expected value of $\hat{\theta} = (\hat{m}, \hat{u}, \hat{p})$, by using the complete data vector $g = (\gamma^i, g_i)$ where g_i is estimated by $\left\{\mathbf{P}(M \cap Q|\gamma^i), \mathbf{P}(U \cap Q|\gamma^i)\right\}$, where the parameters are estimated using

$$\mathbf{P}(M \cap Q|\gamma^i) = \frac{\hat{p} \prod_{j=1}^{n} \hat{m}_j^{\gamma_i^j} \left(1 - \hat{m}_j\right)^{1-\gamma_i^j}}{K}, \tag{3.10}$$

$$\mathbf{P}(U \cap Q|\gamma^i) = \frac{(1 - \hat{p}) \prod_{j=1}^{n} \hat{u}_j^{\gamma_i^j} \left(1 - \hat{m}_j\right)^{1-\gamma_i^j}}{K}, \tag{3.11}$$

$$K = \hat{p} \prod_{j=1}^{n} \hat{m}_j^{\gamma_i^j} \left(1 - \hat{m}_j\right)^{1-\gamma_i^j} + (1 - \hat{p}) \prod_{i=1}^{n} \hat{u}_j^{\gamma_i^j} \left(1 - \hat{u}_j\right)^{1-\gamma_i^j}.$$

In the Maximization-step, the new estimates of $\hat{m}_i$ and $\hat{u}_i$, for $i = 1, 2, \ldots, n$, are given by

$$\hat{m}_i = \frac{\sum_{j=1}^{N} \mathbf{P}(M \cap Q|\gamma_j)\gamma_j^i}{\sum_{j=1}^{N} \mathbf{P}(M \cap Q|\gamma_j)}, \tag{3.12}$$

$$\hat{u}_i = \frac{\sum_{j=1}^{N} \mathbf{P}(U \cap Q|\gamma_j)\gamma_j^i}{\sum_{j=1}^{N} \mathbf{P}(U \cap Q|\gamma_j)}. \tag{3.13}$$

[2]The assumption of Gaussian distributions holds under the *Central Limit Theorem* when the number of comparison vectors is large. The usual rule-of-thumb for the CLT states that the result holds when the number of observations is larger than 30, which is true for almost all sets of data as the number of possible pairs is a quadratic function of the number of data entries.

The estimated proportion of matched pairs is found using the following expression

$$\hat{p} = \frac{1}{N} \sum_{i}^{N} \mathbf{P}(M \cap Q | \gamma^j) \tag{3.14}$$

where N is the number of entries. Iterating these two steps produces maximum likelihood estimations of $(m(\gamma), u(\gamma), p)$ which are used in combination with Eq. (3.7) to classify the links as matching or unmatching. This EM-algorithmic method is due to Winkler (2000) and interested readers are referred to that paper for more details and derivations.

3.2 Machine learning methods

The previously discussed model is based on a probabilistic view of record linkage. This method had the advantage of being unsupervised which suits most applications better than supervised methods, as external labels and training sets are not often available and therefore generating good training sets is difficult. Lacking a good set of labeled training data can result in incomplete or wrongly trained classifiers. Supervised methods often have better accuracy when properly trained, which is the main advantage of this family of methods.

In general, supervised methods also perform better in practical applications and it is therefore interesting to discuss these methods for completeness. Three different methods are discussed: classification using decision trees, support vector machines and unsupervised methods using clustering methods.

3.2.1 Decision Trees

Decision trees are commonly used in classification of patterns and generally in data mining. The simplicity in the interpretation of decision trees in comparison with e.g. neural networks, is one reason for the popularity of this method. A *decision tree* is a tree-like structure which starts at a root node and branches out according to some conditions.

An example of a decision tree is shown in *Figure 3.2*, where two data entries each containing two fields (first and last names) are compared using e.g. the Jaro metric. The comparison function, $\gamma^i = \gamma^i [\alpha(a), \beta(b)]$, is used to classify the entries as matching or unmatching. If the comparison value of the last name fulfills $\gamma^1 > 0.7$ than the comparison proceeds with the first name, otherwise the entries are classified as unmatching. Only if the last name fulfills $\gamma^1 > 0.7$ and the first names $\gamma^2 > 0.8$ are the entries classified as matching.

A popular method for generating decision trees is using *Classification And Regression Trees* (CART) algorithm developed in Breiman *et al.* (1984). In this section, we follow Duda *et al.* (2001) and Hastie *et al.* (2003) to provide a short review of the method and present how it is useful in matching similar records.

Remember the setting in this section, that N observations are available

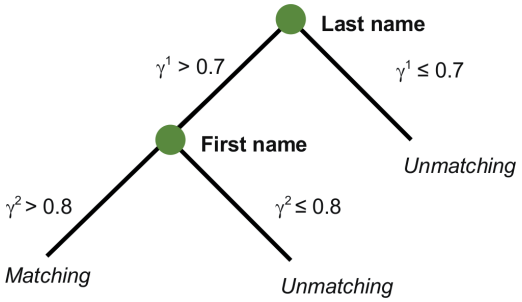


Figure 3.2: A simple decision tree to classify two data entries as matching or unmatching. Each entry contains two fields: first and last names, which are compared using some field matching metric denoted γ^1 and γ^2 respectively.

with n different comparable fields and that the observations (data entries) either match or unmatch.

The CART algorithm splits the space of all observations into rectangle-like sets in which each the response is a constant value, see *Figure 3.3*. Where two features (fields) are used and two regions (corresponding to matching and unmatching pairs of fields) are shown in a 2-dimensional space. In higher dimensional problems the number of separating hyperplanes making a cartoon-like visualization impossible.

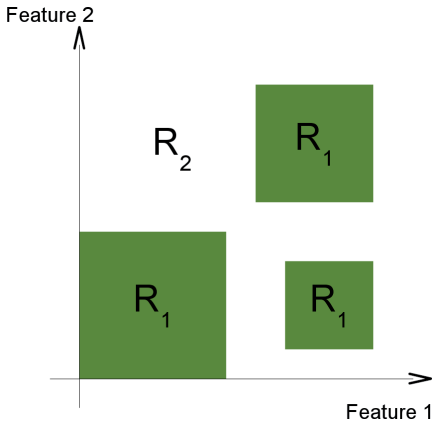


Figure 3.3: An example of some rectangular regions generated by CART in a space of two features and with two classes.

Suppose that the observational space is partitioned into M sets denoted by $R_1, R_2, \ldots, R_M$ which in our setting corresponds to different matching and unmatching sets. The response from an input x (usually $\gamma = x$ as field matching

metrics are used to compare data entries) is expressed as

$$f(x) = \sum_{m=1}^{M} c_m \mathbf{I}\{x \in R_m\}, \tag{3.15}$$

where $\mathbf{I}\{\cdot\}$ is the identity function (assuming value 1 if the expression is true and 0 otherwise) and c_m is the average of the observed response in the region R_m

$$c_m = \frac{1}{N} \sum_{i=1}^{N} y_i \mathbf{I}\{x_i \in R_m\}. \tag{3.16}$$

The half-planes that define the regions (by splitting the observation space) is defined by some point splitting variable j and point s as

$$R_i(j,s) = \{X|X_j \leq s\} \text{ and } R_{i+1}(j,s) = \{X|X_j > s\}, \tag{3.17}$$

for some value of i which indicates the number of branchings. The remaining problem is to determine the number of splittings of the observation space that gives the optimal solution to the classification problem. This is done by using a quality measure usually called the *impurity measure*, $Q_m(T)$, for some number of splittings, m, and tree T. Some different measures for $Q_m(T)$ exist in the literature and Hastie *et al.* (2003) discuss the following:

$$\text{Misclassification error:} \qquad 1 - \hat{p}_{mk(m)}, \tag{3.18}$$

$$\text{Gini index:} \qquad \sum_{i=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{3.19}$$

$$\text{Gross-entropy/deviance:} \qquad -\sum_{i=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}, \tag{3.20}$$

where it is assumed that the classification can take only K values (in our case $K = 2$ but K is kept unspecified for the general case), resulting in that $\hat{p}_{mk}$, the proportion of class k observations in node m, is given by

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbf{I}\{y_i = k\}, \tag{3.21}$$

where m is some node (in the decision tree) representing some region R_m with N_m observations. The observations in node m are classified as class $k(m)$ which is the majority class in node m. Finding the extreme value of $Q_m(T)$ determines the optimal number of splittings. The CART algorithm works backwards to select the optimal model (number of splittings) by first constructing a maximal tree, then pruning it (collapsing nodes) using the impurity measure to find the optimal tree.

3.2.1.1 Boosting and Bagging

Decisions trees are possible to improve by using ensemble methods, e.g. boosting and bagging. These two methods generate large improvements by simple means but require some additional computational effort. The idea is to use multiple classifiers in combination to produce a better classification of an observation. This presentation is based on Hastie *et al.* (2003), which is a good reference for additional information about these methods and their generalizations.

Boosting is a powerful method for improving classification by arranging a large number of weak classifiers into a *committee*, which finally classifies by some form of voting. The boosting method considered in this section is called the *AdaBoost.M1* and is due to Freund & Schapire (1997b). Given a data material on the form (y_i, X_i) where $y_i \in \{-1, 1\}$ indicate the correct labeled class (unmatching or matching) and X_i are some observed data (similarities of data fields). The error rate on the training set of M entries is found by the following

$$\bar{\epsilon} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{I}\{y_i \neq G(x_i)\}, \tag{3.22}$$

where $G(\,\cdot\,)$ is the output of the classifier. The weak classifiers are found iteratively by reweighing the training data set using the following

$$w_i = w_i \exp\left[\alpha_m \mathbf{I}\{y_i \neq G_m(x_i)\}\right], \text{ where} \tag{3.23}$$

$$\alpha_m = \log\left[\frac{1-\epsilon_m}{\epsilon_m}\right], \quad \epsilon_m = \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} w_i \mathbf{I}\{y_i \neq G(x_i)\},$$

for $i = 1, 2, \ldots, M$ (the number of data fields). Repeating the CART algorithm for K such reweighed data sets yield K weak classifiers that together classify an observation x using the following voting rule

$$G(x) = \text{sign}\left[\sum_{k=1}^{K} \alpha_m G_m(x)\right], \tag{3.24}$$

using the value of α_m computed in the m-th step of the iterative procedure to generate weak classifiers.

Another approach to improve classifiers is by using *bagging* (bootstrap aggregating), which draws upon the well-known bootstrap methods from statistics. The bagging method for predictors is due to Breiman (1996) and operates by creating an ensemble of classifiers each generated from a bootstrap sample of the training data.

Let $Z = \{(x_i, y_i) | i = 1, 2, \ldots, M\}$ denote the training data set and draw B bootstrap samples with replacement, Z^{*b}, for $b = 1, 2, \ldots, B$. For each Z^{*b} construct a classifier returning a K-vector, $\hat{f}^{*b}(x) = (\hat{f}_1^{*b}(x), \ldots, \hat{f}_K^{*b}(x))$, with

$\hat{f}_i^{*b}(x) = 1$ where $i \in \{1, 2, \ldots, K\}$ is the classification given by the classifier and $\hat{f}_i^{*b}(x) = 0$ for all remaining elements. The classifier can be constructed by e.g. using the CART-algorithm on the bootstrap sample from the training data set. Let

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x), \tag{3.25}$$

denote a K-vector where each element, K_k, correspond to the proportion of the bootstrapped trees predicting that an observation x is of class k. The observation is classified as the class with the highest proportion of predictions, i.e. a majority vote from the B bootstrapped decision trees. It is possible to show that this method reduces the variance of the estimated classification but does not reduce the bias.

A popular improvement to bagging is by using randomized decision trees, i.e. a random forest (disjoint trees in graph theory), to create an ensemble of classifiers. The random trees are grown using a bootstrapped sample from the training set, Z^{*b}, but by splitting the best node found in a random sample of nodes from the tree until a minimum node size have been found. This ensemble classifies an observation using e.g. majority voting. For more information about random forests, see Hastie *et al.* (2003).

3.2.2 Support Vector Machines

Support Vector Machines (SVM) are commonly used in machine learning to classify observations by splitting the observation space by hyperplanes. This method is due to Boser *et al.* (1992) and this presentation is adopted from Duda *et al.* (2001), Hastie *et al.* (2003), and Han & Kamber (1992). The usual formulation of this problem is as follows

$$\min ||\beta|| \text{ subject to: } \begin{cases} y_i(X_i^\top \beta + \beta_0) \geq 1 - \eta_i \\ \eta_i \geq 0, \sum_i \eta_i \leq C \end{cases}, \tag{3.26}$$

where (as before) X_i denotes observation i, the labels are denoted as y_i, β denotes the vector of parameters defining the hyperplane, $i = 1, 2, \ldots, n$, and $C \in \Re$ denotes some real-valued constant. This optimization problem is quadratic and can therefore be solved using Lagrangian multipliers. The resulting Lagrangian dual function is found as

$$\mathcal{L}_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left\langle h(x_i), h(x_j) \right\rangle, \tag{3.27}$$

where $h(x_i)$ are *transformed feature vectors*, $\langle \cdot, \cdot \rangle$ denotes the inner product, and $\alpha_i \in (0, C)$ are constants. The dual Lagrangian function is maximized

using the following conditions

$$\begin{aligned} \beta = \sum_{i=1}^{n} \alpha_i y_i x_i, \quad \alpha_i = C - \mu_i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \\ \alpha_i \left[y_i \left(x_i^\top \beta + \beta_0 \right) - (1 - \eta_i) \right] = 0, \quad \mu_i \eta_i = 0, \\ y_i \left(x_i^\top \beta + \beta_0 \right) - (1 - \eta_i) \geq 0, \end{aligned} \tag{3.28}$$

where $C \in \Re$ is real-valued contact often interpreted as some *cost* and $i = 1, 2, \ldots, n$. The final solution can be written on the form

$$f(x) = h(x)^\top \beta + \beta_0 = \sum_{i=1}^{n} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0, \tag{3.29}$$

where $h(x)$ are defined through a *kernel function*, $K(x, x') = \langle h(x), h(x') \rangle$, commonly chosen as any of the following:

$$\text{dth-Degree polynomial:} \quad K(x, x') = \left[1 + \langle x, x' \rangle \right]^d, \tag{3.30}$$

$$\text{Gaussian (Radial) basis:} \quad K(x, x') = \exp \left[-c ||x - x'||^2 \right], \tag{3.31}$$

$$\text{Neural network (Sigmoid):} \quad K_{c,d}(x, x') = \tanh \left[c \langle x, x' \rangle + d \right], \tag{3.32}$$

for some real-valued constants $c, d \in \Re$. The cost parameter C is chosen to minimize over-fitting by e.g. leave-one-out cross-validation. Finding the classifier, $f(x)$, by solving this optimization problem with a chosen kernel yields the support vectors which are all of equal distance from the classifier.

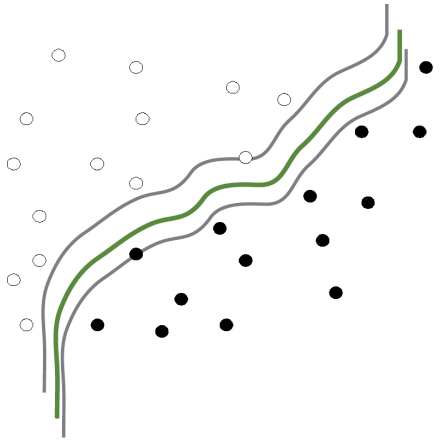


Figure 3.4: An example of a non-linear SVM with two classes. The support vectors are the two data points that are crossed by the margins (grey lines) of the separating hyperplane (green line)

This is shown schematically in *Figure 3.4*, where the classifier (separating hyper planes) have been calculated (green line) between two classes. The grey lines are the margins of the classifier and the two points crossed by these margins are the so called support vectors. This example is of a non-linear SVM using some of the previously discussed kernels that maps the classifier into a linear space, in which the quadratic optimization problem in (3.26) can be solved.

3.2.3 Clustering

The two previously discussed methods, decision trees and SVM, are examples of supervised methods that require some training set of data to find good classifiers. Another example of unsupervised methods in addition to the probabilistic EM-method discussed above is *data clustering* methods, which also can be used in classification. Clustering algorithms are commonly divided into two main categories: hierarchical method that merge data points iteratively from the bottom-up, or top down, and partitional methods that merge data points simultaneously into clusters. In this discussion, we only review partitional methods as they are commonly used in classification of data records.

As previously stated, partitional algorithms merge data points into clusters in one step instead of only two data points/clusters at a time (as in hierarchical methods). The most common (and perhaps simplest) partitional method is *k-means* which operates by dividing data points into k clusters using central points known as *centroids*. Each data point is placed in the cluster which corresponds to centroid with the smallest distance (largest similarity) to the specific data point. The centroids are often initially placed randomly in the observation space and are moved after each data point has been assigned to a cluster. The movement is often determined by the mean of the resulting clusters. This iterative process repeats until the clustering is stable and no data points change clusters.

In *Figure 3.5*, the output after some iterations of the k-means algorithm is shown. Three different clusters have been used to cluster this set of data (shown with '+'-signs) and the data points corresponding the the cluster (shown with squares, circles, and triangles). The result is dependent on the measure used to calculate the updated position of the centroids. Often the simple mean is used but there are other possible measures to use for calculating centroids, each resulting in different clusters.

The main problem with this method is the interpretation of the resulting output, as it is unknown what the identified clusters mean in terms of our labeling: matching, unmatching and possible match. If only two labels are used: matching and unmatching, then the cluster containing higher similarity metrics is usually taken as the matching cluster and the remaining cluster is treated as containing unmatched records.

Another proposed remedy for this problem is by using only a few labeled observations, which are used to identify clusters after the k-means algorithm

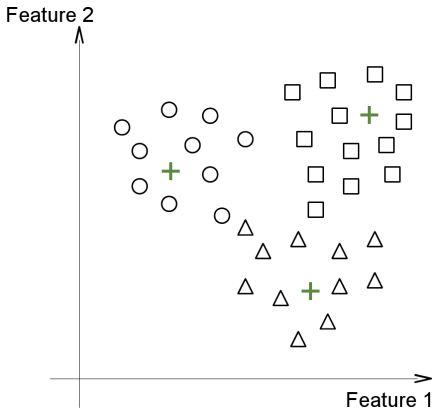


Figure 3.5: A data set with three clusters (shown with squares, triangles, and circles) and the corresponding centroids shown with '+'-signs.

has found some stable solution. This semi-supervised method is due to Verykios (2000) and is a version of a more general method that is referred to as *co-training* in the machine learning literature.

This method is also possible to use together with bootstrap samples from the observed data and then aggregated using bagging to find estimated classifications. The centroids and therefore also the resulting classifications are found from the centroids resulting from each the bootstrapped sample of the training data by hierarchical agglomerative clustering methods using the average linkage method. For details see Verykios (2000).

3.3 Recommendations

This section aims to give some general recommendation regarding the choices of unsupervised and supervised methods to combine several different fields and/or metrics. The importance of which field metric to use was discussed in the previous chapter with the summary that it depends on the data set and that matching methods are more important. Most of the introduced methods in this chapter are compared in Bilenko *et al.* (2003), Sariyar *et al.* (2009) and Köpcke *et al.* (2010).

Sariyar *et al.* (2009) compares all of the introduced methods (in this chapter) using randomly generated (scrambled) data sets and real-world data. It is shown that methods that perform well on random data sets do not always perform well on real-world data. The paper compares the following methods: Fellegi-Sunter with the EM algorithm, CART (with/without bagging and boosting), and SVM. The results indicate that CART with bagging, CART with boosting, and SVM perform best if trained with data that are representative to the data that is later to be matched. This is not the case with

stochastic methods like the Fellegi-Sunter-EM method that do not require any training and therefore is preferred when no suitable training data is available. The former methods are therefore recommended in general when some previous knowledge of the data exist, otherwise the latter method is recommended for matching data entries.

The large problem indicated in other studies using supervised methods, is finding the optimal training data. Some random methods commonly used do not have the right proportions of matching and unmatching entries. It is also difficult to find duplicates in large materials to include and examples of common types of errors. This can be solved by using other methods e.g. active learning methods where the algorithm asks the user when in doubt regarding the classification of an entry.

In Bilenko *et al.* (2003), the authors compare three different methods to combine the similarity of several fields: using one field, the average of many fields, and support vector machines. These methods are used in combination with the following field comparison metrics: soft TF-IDF, Jaro, and Levenshtein. The three different methods are quite different in complexity: the simple single field comparison, the naive mean approach, and the elaborate support vector machine. In this case, complexity is better than simplicity and the authors strongly recommend using SVMs for identity matching.

The SVM method is however supervised and requires a small amount of training data, which therefore imply some manual effort. Supervised methods are often better than their unsupervised counterparts (when good training data exists), and as SVM is the recommended supervised methods as it only requires small training sets. Some implementations have simple tools for generating small training sets by asking the user for manually labeling a few entries (or using active learning). As we later uncover, this is the recommended method for most problems. We urge the reader to keep in mind the comment from the last chapter, these surveys have been conducted on different data sets. Therefore it is difficult to find the best general method but it is possible to find methods that perform satisfactory on a range of different data sets.

4 Implementations

This section introduces and discusses some available tools for matching data entries using the previously discussed methods. Two types of implementations are discussed: autonomous programs and libraries useful in programming languages such as R, C, and Python.

The autonomous programs have advantages in being user-friendly and ready for instant use. The drawback is that the autonomous programs are manual and often do not contain any automatic component, thus requiring user interactions for all (or some) possible matching entries.

This is the advantage of the discussed libraries, which allows for automatically classifying which data entries that are matching and unmatching. In this chapter, we discuss a number of tools for entity matching (duplication detection) that are free for non-commercial usage:

- D-Dupe - Standalone tool developed by a group at Department of Computer Science at University of Maryland.
- FEBRL (Freely Extensible Biomedical Record Linkage) - Python libraries with GUI developed by the ANU Data Mining Group at The Australian National University.
- FRIL (Fine-Grained Records Integration and Linkage Tool) - Standalone Java tool developed by Pawel Jurczyk at Emory University.
- Merge ToolBox (MTB) - Standalone Java tool developed by University of Duisburg-Essen.
- Record Linkage (R Package) - developed by Andreas Borg and Murat Sariyar.
- The Link King (SAS library) - developed by Washington State's Division of Alcohol and Substance Abuse.
- SimMetrics (Java library) - developed by Sam Chapman at the University of Sheffield
- Jellyfish (Python library) - developed by Michael Stephens and James Turk.

4.1 D-dupe

D-dupe is a standalone program for merging duplicate entities in network data, such as citation networks. The program provides a graphical user interface (GUI) that is user-friendly and easy to understand. Possible merges are calculated by user-defined metrics applied on the fields of the different data entries.

Candidates for merges are displayed on the main screen together with the subgraph and more detailed publication data for each of the two authors that are suspected to be duplicates.

The program does not provide any support for data preprocessing, which could be easily done in e.g. Python before importing the material into the program. Data preprocessing includes the standardization and transformation of data fields, which could include removal of punctuations and capitalization. Other common preprocessing steps are splitting and merging strings in such manner that the data fields are comparable between the two data sets. For example, names could be written as several fields (first name, middle name, last name) in one set and as only one field in the other. With transformations, this would result in much lower comparison values between the data sets if compared untransformed.

D-dupe supports a large number of different field matching metrics including: Levenstein, Jaccard, Jaro, and Jaro-Winkler. Different measures can be applied to different fields and each comparison is weighted using a scheme specified by the user. The program does however not include any of the matching methods previously discussed and relies on that the user chooses manually which nodes that should be merged and marked distinct. This is the main drawback of this program as automatic functionality is a requirement for processing any larger data set.

The advantage is that user interaction probably result in better matching results but is impractical for large data sets. For more information and background about D-dupe and the research behind it, see Bilgic *et al.* (2006) and the homepage: `http://www.cs.umd.edu/projects/linqs/ddupe/`.

4.2 FEBRL

FEBRL is a Python-based library that includes field matching metrics as well as several different matching models. It is well-developed and includes advanced methods based on Hidden Markov Models (HMM) to standardize and preprocess data. It is written in Python, making it relatively fast and easy to modify and combine with other methods. The main drawback is that the library no longer is updated and therefore not compatible with current versions of Python and required libraries. It is however possible to use older versions of the program and libraries (if they can be found), or harvest the code (as it is open source[1]) for developing a framework for entity matching.

The library includes many different comparison metrics including compression, edit distance, Jaro-Winkler, q-grams, and phonetic methods. It is possible to use different metrics for different data fields and as in D-dupe these metrics can be weighted by the user's expected ranking of importance concerning the data fields. It also includes a large number of matching models including:

[1]The latter is not possible for commercial projects but some parts can be used for research purposes.

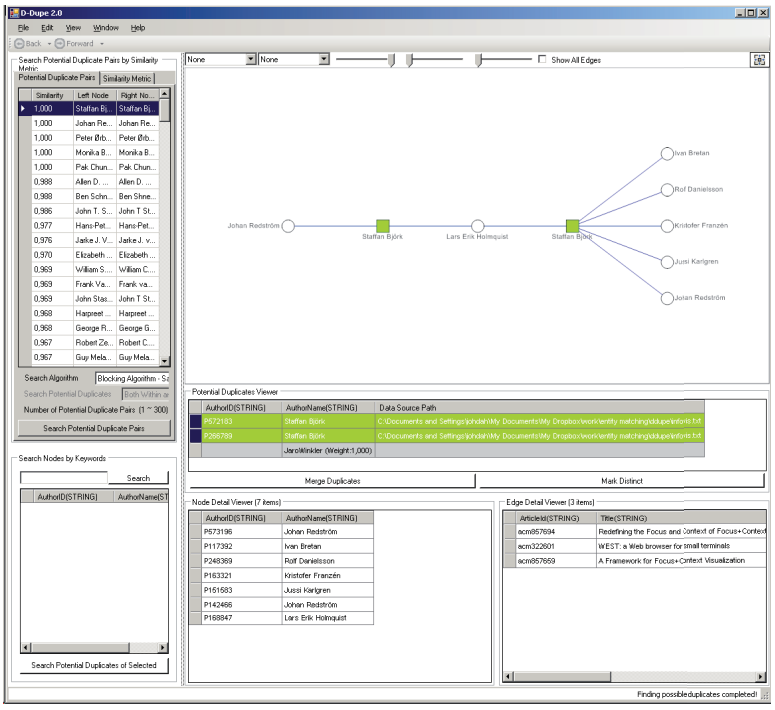


Figure 4.1: The user interface of D-dupe, indicating a match and all connections between two entries in a bibliographic file. The tool let the user select if these two nodes are to be merged or are distinct.

Fellegi-Sunter, k-means, and Support Vector Machines (thereby supporting supervised, unsupervised, and probabilistic methods). For more information and background see Christen (2008) and materials at the project's homepage: `http://datamining.anu.edu.au/projects/linkage.html`. Some documentation of the different libraries and functions included in FEBRL are found in the documentation, which is a good starting-point for salvaging/reusing code: `http://cs.anu.edu.au/~Peter.Christen/Febrl/febrldocu-0.4.01.zip`.

As the library GUI is not functional, it is difficult to evaluate this tool for any future use. The large number of included methods and metrics are however good indicators of that this tool at least was a good choice in the past. Some documentation exists of the different functions and there usage and it would therefore probably be possible to harvest[2] the code included.

[2]This has not however been pursued during this work but is included as a comment for future work.

4.3 FRIL

Another Java-based standalone program is FRIL which supports some different field matching metrics as well as automatic matching through the Fellegi-Sunter method using the EM technique. The supported metrics are: edit distance, phonetic metrics and q-grams. FRIL includes methods for some pre-processing of the data fields, e.g. merging and splitting strings of names and other important fields. The program also allows for using different metrics for comparing different fields and as in the previous programs, these are also possible to weight manually and automatically (by the EM-algorithm).

The user interface is friendly and easy to learn and requires only some small amounts of user interactions. Most of the procedure is automated and the program is quite fast in merging and deduplicating data sets. The main drawback of the program is the lack of functionality as it only supports one method for record matching. The program is supported by some video tutorials on their homepage. Some additional materials are found in Jurczyk *et al.* and at the homepage: `http://fril.sourceforge.net/`.

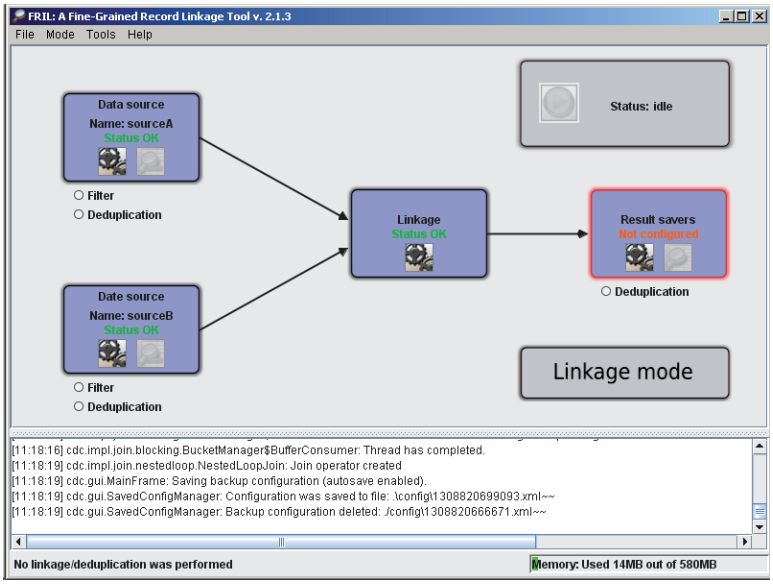


Figure 4.2: The user interface of FRIL. The graphical representation of the process is shown: pre-processing of the data sources then a linkage rule that is weighted to finally produce an output.

4.4 Record Linkage (R Package)

Record Linkage is a library for the statistical freeware R, which includes a large number of field matching metrics and automatic matching methods. The metrics supported are phonetic codes, Jaro-Winkler, and Levenshtein. The automatic matching methods included are e.g. Fellegi-Sunter (using the EM method), k-means, classification trees (with bagging or boosting), support vector machines, and neural networks.

As this is a library, it is possible to use the included methods in any manner by writing scripts. It is also possible to include new functions and metrics, by just adding functions to the library code. This makes libraries more versatile and simpler to modify for special needs. R also includes a large number of libraries for ensemble methods, classification, clustering, text analysis, data mining, and statistical analysis/evaluation. This is a large advantage for rapid prototype development of ensemble methods which include more than strings metrics for matching data entries and identities.

The main drawback is that libraries requires the user to write script to utilize the methods included. This therefore requires some familiarity with R and is not as user-friendly as some of the previously discussed implementations. As R also is quite unknown for many, examples of code and tutorials are provided in *Appendix A* and *Appendix B*. More information is found in Sariyar & Borg (2010) and on the package's homepage at the Comprehensive R Archive Network: `http://cran.r-project.org/web/packages/RecordLinkage`.

4.5 Others

Other software and libraries are also available freely for non-commercial usage. We continue by naming a few of these with a shorter description.

The first tool is called Merge Toolbox (MTB) which includes automatic matching using distance-based methods. See Schnell *et al.* (2009) and the homepage: `http://www.uni-due.de/soziologie/` for more information.

Another popular tool is The Link King. The main drawback of this tool is that it requires SAS to function. More information is found at the homepage: `http://www.the-link-king.com/`.

Some libraries exist for Java and Python calculationing field matching metrics. For Java, SimMetrics can be used for calculation of a large number of metrics including: Levenshtein, Jaro-Winkler, phonetic, Jaccard, and q-grams. The package is found at: `http://sourceforge.net/projects/simmetrics/`. For Pyhton, Jellyfish can be used for calculating metrics such as: Levenshtein, Jaro-Winkler and phonetic metrics. The package is found at the Python library collection: `http://pypi.python.org/pypi/jellyfish`.

4.6 Recommendations

D-dupe is the only tool that is specifically developed for entity resolution in social networks, as it shows the user the relevant subgraphs and any common neighbors. This allows the user to make better decisions regarding if two entries (nodes) should be merged or if they are distinct. The drawback with this method is that it requires user interaction and it is difficult and time consuming to apply this tool on any large data sets. This tool is more suitable for smaller data sets or for constructing labeling for training a supervised method.

FRIL is a good tool for finding duplicates or similar entries in large sets of data. The tool supports many different metrics but only one matching method and as previously discussed the Fellegi-Sunter-EM algorithm performs worse than the discussed supervised methods. The advantage is that the selected method does not require any training and that the tool is user-friendly and simple to use.

The two last discussed tools are libraries for Python and R. While these tools are versatile and useful, they both require some working knowledge in programming and the respective languages. FEBRL is widely used in benchmark papers and seems to be an efficient implementation of many of the discussed methods. The only drawback is the lack of compatibility with current versions of Python and required packages. Some functionality can be recovered using the functions without the user interface, or by rewriting parts of the code to run even without some required packages (but with less functionality).

The other library, which is adopted for some future work in *Chapter 7*, is the R-package RecordLinkage. This tool implements many of the different methods described to combine the similarity of several fields to find similar entries. The drawbacks with the package is the lack of more complex metrics and that R is quite unknown as a programming tool, which we compensate by some newly developed code that is presented in *Appendix B* together with a tutorial on using the package in R, found in *Appendix A*.

For future usage, we therefore recommend these four tools but for different situations. Python is in general faster than R, so for larger data sets it is preferable to use Python and FEBRL (if possible). The advantage with the R package is that it is still under development and is actively maintained. D-dupe is good for verifying results and creating sets of training data. FRIL is good for some applications where training data is unavailable or unsupervised methods preferred.

5 Graph and text mining-based methods

Traditional entity matching (and record linkage) methods are often concerned with applying the field matching metrics previously discussed. These field metrics are useful if the data fields are shorter text strings. However, these metrics are not useful if the fields contain longer written text documents.

In this chapter, we discuss other methods to compare the similarity between data entries, namely graph-based and text mining-based methods. The former analyzes the relationship between different data entries to find authors with many common connections, which could indicate that these entries are only one entity. The latter methods use text analysis to identify texts found in different entries that discuss the same topics and uses the same type of words. This could indicate that the texts are written by the same entity and that therefore these two entries correspond to a single real-world entity.

These methods are later applied in a data fusion framework that uses more than one data source (field metrics, vertex similarities, and text similarities) to determine if two entries correspond to a single entity or not.

5.1 Vertex similarity

An approach for identifying similar entities in networked data (such as link or author networks) is using the network structure itself. The idea for this originates from the fields of complex networks and sociology, where *vertex similarity* and *structural equivalence* have been studied. The main underlying thought is that nodes are similar if they share a large fraction of neighbors, e.g. share many friends in a social network or share many coauthors in citation networks. An example of this is shown in *Figure 5.1*, where two nodes corresponding to authors in a citation network are shown with their neighbors. As these nodes share a large fraction of neighbors and have similar names, it is possible that they correspond to the same real-world person.

We follow Leicht *et al.* (2006) by discussing some common simple metrics for determining vertex similarity. The simplest possible measure of the similarity of two nodes in the graph-based setting is the number of shared neighbors found as

$$\sigma(v_i, v_j) = |\Gamma_i \cap \Gamma_j|, \tag{5.1}$$

where Γ_i denotes the set of neighbors of node v_i. The drawback of this measure is that nodes with a high degree (many neighbors) will have a larger similarity than nodes with smaller degree. To obtain a comparable measure, it is necessary to normalize the similarity by the node degrees. Some common similarity measures, $\sigma(v_i, v_j)$, are

$$\text{Jaccard: } \frac{|\Gamma_i \cap \Gamma_j|}{|\Gamma_i \cup \Gamma_j|}, \quad \text{Cosine: } \frac{|\Gamma_i \cap \Gamma_j|}{\sqrt{|\Gamma_i||\Gamma_j|}}, \quad \text{Vertex: } \frac{|\Gamma_i \cap \Gamma_j|}{|\Gamma_i||\Gamma_j|},$$

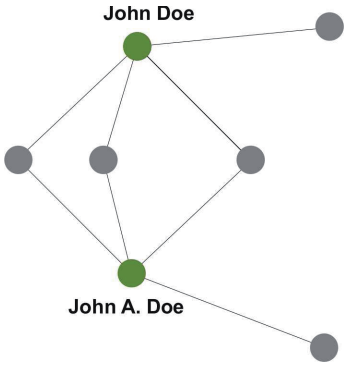


Figure 5.1: Two nodes and their neighbors (grey nodes) corresponding to a subnetwork of some citation data.

$$\text{Dice: } \frac{2|\Gamma_i \cap \Gamma_j|}{|\Gamma_i| + |\Gamma_j|}, \qquad \text{Inv. log-weighted: } \sum_{x \in \Gamma_{ij}} \frac{1}{\log |\Gamma_x|},$$

where $x \in \Gamma_{ij}$ is common neighbors to nodes i and j, i.e. $\Gamma_{ij} = \Gamma_i \cap \Gamma_j$ in the inverse log-weighted similarity that is due to Adamic & Adar (2003). Similar nodes in graphs will have high values of similarity, which may indicate that two nodes are not distinct. By using some predetermined limit value, l, classification is achieved using some linkage function such that

$$f(x) = \begin{cases} -1 & \text{if } x = \sigma(v_i, v_j) \leq l \\ 1 & \text{if } x = \sigma(v_i, v_j) > l \end{cases}, \tag{5.2}$$

i.e. the two nodes match if $f(x) = 1$ and unmatch if $f(x) = -1$ for some similarity x.

If the data comes from a citation network, nodes sharing many neighbors and having similar names may indicate duplicate records. One separate evidence (similar names or sharing many neighbors) is not proof enough to indicate duplicate entries. Combining the two evidences may offer a stronger proof of that the nodes correspond to the same author. This is the starting point of the idea of fusing multiple classifiers into one better classifier, which is discussed in the following chapter.

5.2 Latent Semantic Analysis

Text mining is usually applied to find documents with similar context or for finding semantic expressions (grouping words with similar meaning). The most commonly used method is often referred to as *Latent Semantic Analysis* (LSA) and follows a series of three steps: (i) the creation of a weighted Term-Document (TD) matrix, (ii) calculating a truncated singular value decompo-

sition, (iii) calculating similarity and clustering similar documents. These different steps are now discussed in detail.

The grouping of similar documents is done using the partitional clustering algorithm called k-means with the *cosine similarity* between documents

$$\mathsf{sim}(\mathbf{D}_i, \mathbf{D}_j) = \frac{|\mathbf{x} \cap \mathbf{y}|}{\sqrt{|\mathbf{x}||\mathbf{y}|}}, \tag{5.3}$$

where $\mathbf{D}_i$ and $\mathbf{D}_j$ are some vectors describing the content of two (different) documents i and j. These vectors are found by a truncated *Singular Value Decomposition* (SVD) of the weighted TD-matrix. The TD-matrix, $\mathbf{X} = [X_{ij}]$, is found by calculating the frequency of words occurring in each document, the rows denote the different terms used in all documents in the collection of documents (the Corpus) and the columns denote the different documents. Each element in the TD-matrix, X_{ij}, describes the frequency with which the word i occurs in the document j.

The TD-matrix is weighted using the TF-IDF measure previously discussed in connection with the WHIRL measure. The aim is to give uncommon words more weight than more commonly found words[1]. The matrix is weighted using the same expression that is shown in Eq. (2.4) for the WHIRL metrics. This weighted TD-matrix is decomposed using the following relation

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top, \tag{5.4}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices with *singular vectors* and Σ is a diagonal matrix with the *singular values*. To truncate this decomposition, an appropriate number of singular values needs to be determined. This is usually done by some rule-of-thumb, e.g. the Kaiser criteria (stating that all values larger than unity should be included). Another method usually applied in Principal Component Analysis (PCA) is to the find the *elbow point* in the *scree plot*. The latter is a plot of the decreasingly sorted singular values and the former is found as the point after which the scree line tends to level out, see the cartoon in *Figure 5.2*.

The truncated expansion is found by choosing some appropriate number of singular values, k, as the following

$$\hat{\mathbf{X}}_k = \sum_{i=1}^{k} \mathbf{U}_{ik}\Sigma_{ii}\mathbf{V}^\top_{ki}, \tag{5.5}$$

where the columns of $\mathbf{X}$ are the new *coordinates* describing each document. These column vectors are used in the cosine measure as the vectors describing documents. The advantage of this method is that the truncated SVD reduces

[1]It is worth noting that *stop words* often are removed before constructing the term-document matrix. These stop words include common prepositions and other functional words, e.g. the, is, and, which, and that, which carries no specific meaning.

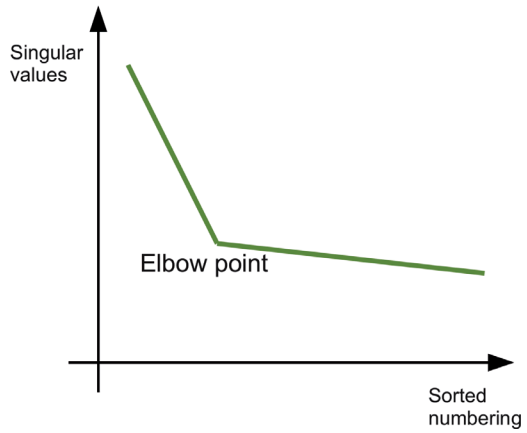


Figure 5.2: A scree plot with the singular values sorted in decreasing magnitude. The elbow point is found as the point after which the line tends to level out, as indicated by the label in the graph.

the number of words (by creating groups of words with similar meaning as new basis vectors) and by reducing the amount of noise (infrequently occurring words). This makes it easier to find documents (abstracts) with similar content.

The cosine similarity is computed for each pair of documents and placed in a similarity matrix, $\mathbf{S} = [S_{ij}]$, where the element $S_{ij} = \mathsf{sim}(\hat{\mathbf{X}}_{k,i}, \hat{\mathbf{X}}_{k,j})$ denotes the cosine similarity between columns i and j in the truncated SVD computed above. This matrix describes the similarity between documents and is used together with the k-means algorithm to cluster similar documents into groups. The k-means algorithm is used to generate k clusters of documents and the result is a label for each document c_i, where $i = 1, 2, \ldots, N$ and N is the number of documents, describing to which cluster each document belong.

6 Proposed frameworks

As previously discussed, an interesting possibility for creating better classifiers for matching duplicate data entries is by fusing the output from a number of different classification methods or data from several sources. The motivation for this is the difficulty of finding matching entries/entities by using only one of the previously discussed metrics or matching models. It may therefore be fruitful to investigate if it is possible to employ data fusion methods on this problem, as this allows for combining different methods and data from several sources.

The simplest methods for data fusion are based on averaging and voting, but some more elaborate techniques exist including Dempster-Shafer theory and various versions of decision trees. These methods are reviewed in this section and applied to the problem analyzed in this report concerning entity matching.

Remember the scenario outlined in the introduction of this report, some data has been gathered from a large number of sources, e.g. by mining blogs and similar social pages on the web. The combination of these heterogeneous sources generates a large data structure with incomplete and contradictory information. The aim is to identify similar elements and merge them together to clean and integrate the information found in different sources. This can be done in one of two different ways, either by reducing the number of pairs to compare using traditional record linkage methods, or by applying data fusion methods to combine evidence from a number of different classifiers. The two methods are presented graphically in *Figure 6.1*, where graph-based and text mining-based methods are used to identify duplicate candidates or for estimating evidence for the hypothesis that two nodes are duplicates.

6.1 Weak classifiers

The information obtained from the different sources in this scenario are of at least three different types: names, number, etc. (field data), links and interactions (network data), and finally the language structure (text mining data) found in the blog posts. It is the aim of this chapter to describe how this data could be used in combination to recover the underlying network by summarizing and applying the background presented in the previous two chapters.

6.1.1 Statistical learning-based methods

The entity matching methods originating in machine learning and statistics were discussed in *Chapter 3*, included the Fellegi-Sunter model (probabilistic/statistical method) and supervised and unsupervised learning. The latter methods require some string comparison measure, a field metric, to compare

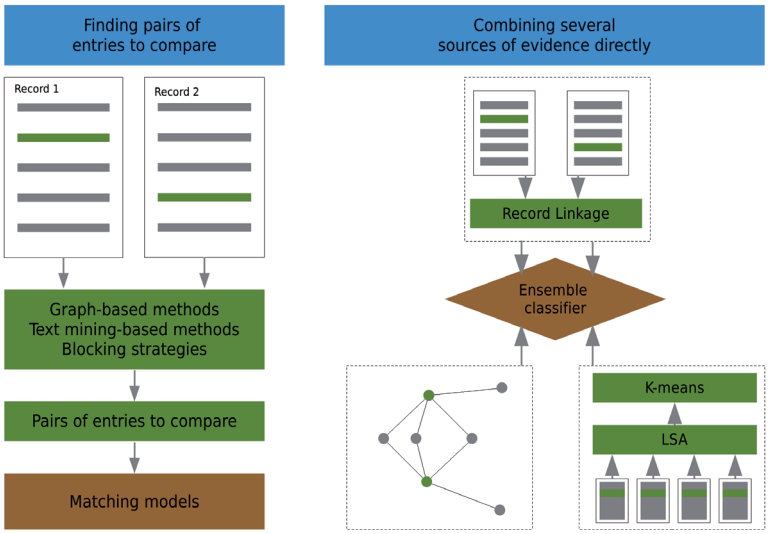


Figure 6.1: The two different methods proposed in this report. (i) to find a small number of possible pairs to be compared with standard entity matching methods including the Fellegi-Sunter method and other methods based on machine learning. (ii) to merge the output from an ensemble of classifiers either directly by voting, or by combining evidences using Dempster-Shafer theory or fuzzy sets.

strings of characters or digits. This fuzzy metric is used as an input to e.g. classification trees or partitional clustering, which are used to classify observations of pairs of entries as: matching, unmatching, or possible matches. Some of these methods require training and perform well when trained with good data sets. The Fellegi-Sunter model and clustering methods does not require training but instead require that some important assumptions are fulfilled, e.g. large sets of conditionally independent data.

The field matching methods suitable for a certain type of data vary and no general best choice exist as a result of the no-free-lunch theorem, stated by Wolpert & Macready (1995). The theorem states that there exists no single best method for learning (classification), the best method for a certain problem is dependent on the data material used. In the experiments found in the following chapter, the only suitable data material in the compiled data are the names of authors, therefore the Jaro-Winkler measure is considered the best choice following the recommendations in the previous chapter on field matching metrics.

One additional problem with the author names is that they only contain the last names and the initials, making it difficult to find a good blocking strategy and thus limiting the required number of comparisons. In other applications, it is often the case that address data, day of birth, etc. also are available and then

a blocking strategy could be that year of birth and address is equal, year or day of birth is equal, etc. In this situation, only two different blocking strategies are possible: requiring that the initials are equal or that the last names are equal, thus limiting the possibility of applying any traditional blocking strategy. To counter this problem, we later propose some other methods used as a preprocessing step to limit the number of comparisons of pairs.

6.1.2 Graph-based methods

Graph-based methods are discussed in *Section 5.1* together with a number of different measures used for comparing the similarities of nodes in networks. Suitable networks in the scenario are e.g. co-author, link, communication networks, where each node indicate an individual and each link corresponds to some interaction or relation observed between two nodes. In co-author networks, each node corresponds to an (unique) author and each link indicates that two authors have published a paper together. Other possible networks could indicate links between blogs, or evidence of communication between two individuals. The aim is to find nodes with similar interactions, which is the basis for some evidence that the nodes may not be distinct but in reality are duplicates (using different names/aliases or spellings).

Similar interactions are equivalent with sharing a large fraction of neighboring nodes, i.e. nodes that are connected by a link to a specific node. By identifying similar nodes, the number of pairs to investigate using statistical and learning-based methods are greatly reduced. Thus, this graph-based method can be used as a pre-processing step to speed up computations but also to include some additional evidence that two nodes are duplicates. The latter function is discussed in more detail later in this chapter, in connection with ensemble methods to combine evidences for multiple sources.

Graph-based methods are used in the following chapter, in some experiments to find duplicates in co-author networks. To find pairs of nodes with similar neighborhoods, the Jaccard, Dice, and inverse-log weighted similarities are used in combination with a voting rule. Each similarity measure is compared with a predefined limit value (classification region) to classify each pair of nodes as similar or dissimilar. The classifications from the three different methods are combined using a majority rule, i.e. choosing the label most frequently occurring in the output from the similarity measures for each node. It is important to note that there are many other rules for combining evidences (results from different classifiers), e.g. using the median, sum, product, or generalized mean of the similarity and compare this value with some classification region.

6.1.3 Text mining-based methods

Text mining-based methods are discussed in *Section 5.2*. This approach compares entries containing long text fields by using Latent Semantic Analysis to

find similar topics in texts. In general terms, these methods can also be used for blog posts, articles and other documents to find similar topics and use of language. The aim in the following simulation experiments is to use text mining methods for finding abstracts with similar contents, which can be used as a indicator for similar authors. This is useful as most scientists work within a certain sub-field and the abstracts should therefore contain similar words.

Other applications are to analyze topics of texts written in blogs and perhaps also implement methods to analyze the style of writing in texts. These two approaches can return some form of quantitative measure of how similar texts are and therefore also evidence that two authors correspond to the same real-world entity. These methods are also useful in limiting the number of comparisons to be done in e.g. the Fellegi-Sunter model. Therefore acting as a blocking strategy by grouping abstracts (authors) together into groups and only comparing authors within the same group. If the assumption of authors consistently writing about the same topics is correct, this strategy should be effective in limiting the computational effort needed for entity matching.

6.2 Sequential blocking method

As previously discussed, a sequential blocking method can be used to identify possible candidates for pairs of nodes that are duplicates. This section will present the idea in a more formal manner. This method is particularly useful in the cases where only small amounts of text strings are available for each data entry. This makes it difficult to apply standard methods including blocking strategies and the Fellegi-Sunter method, as only a few fields are comparable with field comparison metrics.

The idea is instead to use other information, e.g. from the network structure or some text documents to find pairs of nodes that could be similar. As previously stated, this reduces the number of possible pairs to compare and therefore also the computational effort needed. If the data material consists of many data fields including standard information such as name, addresses, day-of-birth, etc. ordinary record linkage methods can be applied. Finding a good blocking strategy is difficult and is often limited by the amount of information at hand.

A similar problem is that a data material can sometimes contain very few data fields, thereby limiting the effectiveness of standard entity matching using field matching metrics. Using only a name field makes it very difficult to accurately determine if two entries are duplicates. An example is the following two names: *John A. Doe* and *John Doe*, are these two entries referring to the same real-world person? It is impossible to tell using only this data but by considering some other aspects, more evidence are gathered and more certain classifications are possible. This sequential method filters other possible duplicates and therefore adds evidence from network structures and text sources in several steps.

The sequential method reduces the considered set of data entries, $\mathbf{A} = (A_i)$, where A_i is a data entry with $A_i = (A_{ij})$ as the j data fields contained in the entry. The set of all possible pairs is found as

$$\hat{\mathsf{A}} = \mathbf{A} \times \mathbf{A} = \{(A_i, A_j); A_i, A_j \in \mathbf{A} | i, j \in \{1, \ldots, n\}, i \neq j\}\,. \tag{6.1}$$

The number of possible pairs is related to the number of data entries, $n = |\mathbf{A}|$, in the data record $\mathbf{A}$. For $n = 10^3$ data entries there are therefore $n^2 = 10^6$ comparisons to be made.

The reduction is done by using some other method, e.g. graph-based methods and/or text mining-based methods to screen all possible pairs. The set of possible pairs are nodes that according to the selected method(s) are similar. Using the graph-based method, this is equivalent to assuming that only pair of nodes sharing at least a certain fraction of their neighbors are interesting to compare using record linkage methods. By screening the total set of possible pairs, $\hat{\mathsf{A}}$, it is possible to greatly reduce the number of comparisons to be made by this form of blocking strategy, that does not depend on the data fields. The name sequential blocking comes from the idea that some methods can be used in sequence for step-wise reducing the set of pairs.

An example that corresponds to our setting is to first require that authors should share some fraction of co-authors to be considered candidates for duplicates, and then require that a number of abstracts have at least some specific similarity. This drastically reduces the amount of possible candidate pairs, $\hat{\mathsf{A}}$, to consider using e.g. traditional record linkage with the Fellegi-Sunter model.

6.3 General ensemble method

This proposed method differs from the sequential approach in using all the available information from different classification methods simultaneously. The sequential approach limited the number of pairs to consider step-by-step. If the amount of information is small, this approach could fail to find similar authors and entries. Instead, the information is used in a more optimal manner by collecting as much information as possible before classifying pairs as matching or unmatching.

This method is based on extensive earlier work in ensemble methods, interested readers are referred to e.g. Hastie *et al.* (2003), Polikar (2006), and Rokach (2010) for more information. In general, ensemble methods use a combination of weaker classifiers to create a better one using a wisdom-of-crowds-effect. Examples of ensemble techniques include the previously discussed: bagging and boosting method, where the training data is re-sampled to generate an ensemble of classifiers. These methods are generally referred to as *generative methods*, as some random method is used to create a group of classifiers by changing the training data.

Given an ensemble of classifiers (generated randomly or by some other method), the aim is to combine the classifications into a single ensemble classification of an observation. Two different methods are generally used to do this,

according to the taxonomy used in Rokach (2010), namely weighting methods and meta-learning methods.

Weighted methods use the combination of the classifications from a number of different types of classifiers each trained on the entire (or a subset) of the data set. This type of methods require that the output between the classifiers are of the same form, e.g. one of two different labels or a number in a certain interval.

Meta-learning methods use the output from several different classifiers as an input to some new supervised learner, which is trained by the same labels as the ensemble of classifiers. This latter method uses the labeling twice, first to train the ensemble of weak classifiers and to train the meta-classifier that combines the output of the ensemble to generate a single ensemble classification of an observation.

In our setting, both of these methods are useful for combining the classification from the ensemble of weak classifiers, where each classifier is some metric comparing pairs of fields, vertex similarity, or similarity in abstracts. However, for the sake of simplicity and due to time constraints, only the former method using a weighted scheme is discussed in detail in this section. Using meta-learning is quite straight-forward and details of this method is provided in Rokach (2010).

The different weak classifiers available are the probabilistic matching, machine learning-based matching, graph-based, and text mining-based methods. All these classifiers can be used to compare the text strings, network information, and documents contained within the data entries. These classifiers return either a labeling of a pair of nodes (as matching or unmatching) using some classification regions, or a quantified measure indicating how similar two nodes are in some respect.

This is partly shown in *Figure 6.2*, where the three different discussed methods are combined into an ensemble that classifies entities and data entries. Some methods are used to classify (or quantitatively determine the similarity) of some data entries. These results are merged using some method from data fusion, such that the most probable label or a probability is returned. Thus determining the similarity between data entries and if each pair of entries should be considered duplicated and therefore merged.

6.3.1 Weighted combination

To combine a number of weak classifiers using some weighted combination method, each classifier must return some label and also sometimes a quantified probability or certainty that the classification is correct. There exists a large number of different methods for combining these labels, e.g.

- *(Weighted) majority voting*, combines the classifications by selecting the most common label.

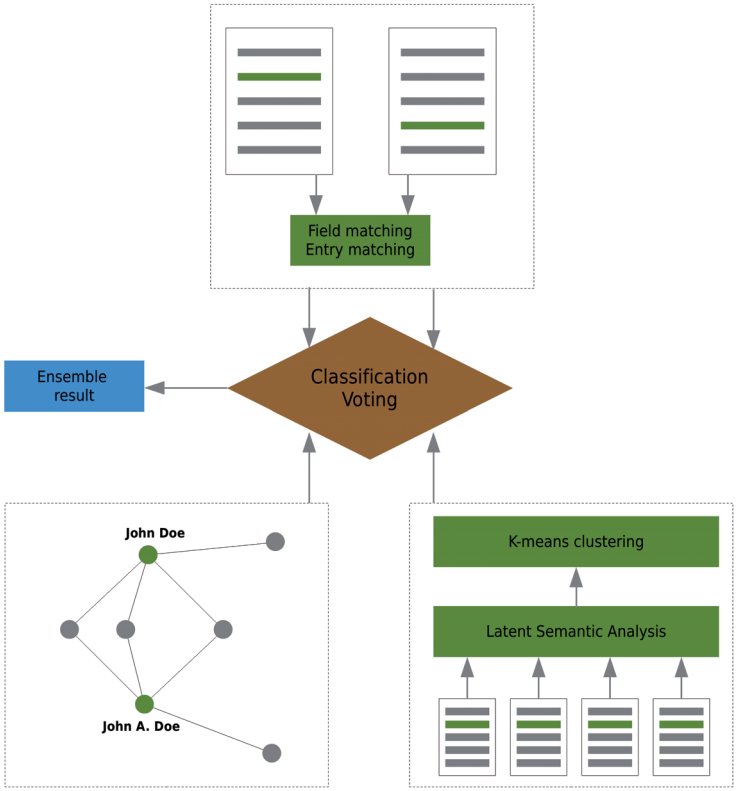


Figure 6.2: A cartoon over the proposed method to combine text metrics, graph-based methods and LSA into a classifier for finding duplicate entries and unique entities.

- *Bayesian combination*, selects the most probable label using the posterior probability given a training set. (Buntine, 1990)
- *Dempster-Shafer combination*, uses a generalized form of probability which naturally allows for combination of several sources of evidence. (Shlien, 1990)
- *Vogging*, uses an interesting variation of Markowitz Mean-Variance Portfolio Theory to find a linear combination of classifiers to maximize the accuracy and minimize the variance. (Derbeko *et al.*, 2002)
- *Naive Bayes*, classification by using the naive (assuming independent weak classifiers) Bayes' rule for combining several labels.

All these methods and some other are discussed briefly in Rokach (2010), which also provides references for further reading for interested readers. In this report, only a few of these methods are discussed for completeness and future work includes studies and adaptations of these methods for use on a case-by-case basis.

6.3.1.1 (Weighted) majority voting

One of the commonly used and in general considered best method is using a simple voting scheme (either weighted or unweighted). Each classifier votes on the label which it has determined is the most probable one. These votes can be weighted using some measure describing the certainty and performance of the classifier, which is often previously determined from larger studies or assigned by some human expert.

Let $\mathbf{y} = (y_i)$ denote the classifications from the ensemble, where y_i is the label designated by classifier i. A simple *majority rule* is found using

$$\hat{y} = \mathsf{cm}(\mathbf{y}), \tag{6.2}$$

where the function $\mathsf{cm}(\,\cdot\,)$ returns the most frequently found label in the argument. Another method is found by weighting each classifier with some weight, w_i, found from training data or manually determined. The corresponding *weighted majority voting rule* is

$$\hat{y} = \mathsf{cm}(\mathbf{w}^\top \mathbf{y}), \tag{6.3}$$

where $\mathbf{w} = (w_i)$ is the vector of weights with w_i as the weight for classifier i.

As previously discussed, this combination method is one of the simplest methods and often performs well. This is the background for why it is often recommended as the best method, simplicity and focusing on increasing the data set are often more important than using more advanced methods.

6.3.1.2 Dempster-Shafer combination

Another well-known and commonly used method in data fusion is *Dempster-Shafer theory* and *Dempster's rule of combination*. In this method, each classifier either provides an estimate of the probability that the labeling is correct, or is given a probability by some earlier experiments or from a human expert. These probabilities are combined into the *belief* and *plausibility* that some outcome is true, e.g. that a pair or entries are duplicates.

In Shlien (1990), the authors propose the use the following *mass function*, $m(\,\cdot\,)$, to assign probabilities to different outcomes

$$m(c_i, x) = 1 - \prod_k \left[1 - \hat{p}_{M_k}(y = c_i|x)\right], \tag{6.4}$$

where $\hat{p}_{M_k}(y = c_i|x)$ is the estimated probability that the classifier i correctly classifies the observation x to the class c_i which corresponds to the external

labeling y. The *belief function*, $\mathsf{Bel}(\cdot)$, is found using the following expression

$$\mathsf{Bel}(c_i, x) = \frac{1}{K} \frac{m(c_i, x)}{1 - m(c_i, x)}, \tag{6.5}$$

where the *amount of conflict*, K, is calculated by

$$K = \sum_{c_i} \frac{m(c_i, x)}{1 - m(c_i, x)} + 1, \tag{6.6}$$

and the classification from the combined ensemble of classifiers is the label, c_i, that maximizes the belief function. This is only one of many different methods for using Dempster-Shafer to combine the outcomes of classifiers. Important future work includes more investigations into suitable mass and belief functions as well as selecting an appropriate combination rule.

6.3.2 Meta-combination

The other class of methods for combining ensembles of weak classifiers discussed in Rokach (2010) are *meta-combination* methods. One of the most accurate is *stacking* several different classifiers generated by different inducers, i.e. parts of training data or different classification methods. The aim is to use learning with the original training data and the observed outcomes from the classifiers to determine e.g. the weights in a decision tree that optimizes the accuracy.

In our setting each classifier i generates some output label, c_i, from the training data and the observation x. These outcomes are used by the meta-combiner with the training data to find a *meta-classifier*, y, on the following form

$$y = f(\mathbf{x}) = f(c_1, c_2, \ldots, c_n, x). \tag{6.7}$$

It has been shown that this method generates approximately the same accuracy as selecting the best classifier i using cross-validation. Some more advanced methods have been proposed to increase the accuracy beyond that of the best weak classifier.

Stacking is one of the simpler methods to combine the outcomes from some weak classifiers using a trained classifier. However some more interesting methods exist in this family of combination methods:

- *Arbiter trees*, a bottom-up method to combine classifiers two at a time into arbiters that are combined until only one remain. (Chan & Stolfo, 1993)
- *Combiner trees*, similar to arbiter trees but uses combiners trained with the classifications of each pair of weak classifiers. (Chan & Stolfo)
- *Grading*, uses trained meta-classifiers to determine when weak classifiers are correct in their classification. Only weak classifiers that are considered

> classifying the given type of observation correctly are included in the combination. (Seewald & Fürnkranz, 2001)

As in the weighted combination methods, less is often more and simple method are often considered the better choice for meta-combiners as well. The main issue with this method is to select an appropriate method to classify the combination. CARTs are useful as the weights are easily understood in terms of the accuracy and performance of the weak classifiers. SVMs are robust methods with high accuracy and also require only a small training set. It is not straight-forward to select the best method for ensemble method for entity matching and this remain an area of future study.

6.4 Recommendations

As previously stated, the following evaluation experiments use the sequential methods for generating blocking strategies. This is mainly due to time constraints and the better method of choice for future work is to implement the full data fusion method. This method requires some computer experiments to find appropriate methods for either finding the weights for majority voting, functional expressions for the Dempster-Shafer method, or for selecting an appropriate meta-learning and train this using a manually constructed training set.

The easiest method to implement is probably the majority voting rule using some form of weights distributed by a human expert. These weights should be selected in such manner that the more accurate (and relevant) weak classifiers receive more influence of the combined classification. An expert could select which parameters that are most important for the currently considered data set and therefore control the combination method. Some aid to selecting weights could be found using a trained decision tree as a meta-combiner. The weights of each weak classifier is then easily found as the height of the tree from that classifier to the root.

Meta-learning could also be an important method for combining the ensemble of classifiers. Using a simple meta-classifier, such as the decision tree with bagging or boosting creates a simple but also powerful meta-combiner. Stacking is therefore the best considered choice in this more advanced form of ensemble classification.

Finally, summarizing the different approaches introduced in this chapter. The simplest and fastest method to implement is the sequential method for generating appropriate blocking strategies for the usual record linkage methods. These methods are well-known and tested using many different data sets. The drawbacks of record linkage methods with the sequential method is that all properties are assumed to be independent of each other. The cross-effects of e.g. the network structure and abstract text are neglected and therefore some relevant pairs are missing from the list of candidate pairs.

Other more advanced methods based on data fusion methods include voting

and stacking. These methods are probably more relevant in future applications and remains a future topic for research.

7 Results

To test and evaluate the framework sketched in the previous chapter, we apply some of the methods proposed on the data material investigated in Johansson *et al.* (2011). The authors of this reference have combined citation data from the FUSION conference using two different sources. This corresponds well with the previously discussed problem with heterogeneous data sources that e.g. have different spellings of names.

Following the discussion in the last section, the number of pairs of candidate duplicates is limited by blocking strategies or by requiring graph or text similarities. The graph similarity consists of sharing a significant fraction of neighbor nodes, i.e. a large number of co-authors in citation data. Text similarity consists of a high cosine similarity in the truncated SVD from the citation abstracts. High similarity corresponds to similar paper topics, which is assumed to be an evidence supporting that two authors are duplicate candidates.

The considered data material consists of citation information about 786 papers[1] with a total of $1,899$ authors (neglecting duplicate names). The number of possible pairs, $n^2 = 3,606,201$, is large and comparing all possible pairs is time consuming thus requiring some smart method to reduce this set. The data material consists of the usual citation information with e.g. authors, titles, and abstracts. These are the only data fields that are useful for matching different spellings of names of authors. To reduce the number of candidate pairs, we now continue with the blocking strategies, graph-based, and text mining-based methods discussed in the previous chapters.

7.1 Fellegi-Sunter method with blocking

The most common method for reducing the number of possible pairs is a blocking strategy. As previously discussed, this strategy requires/assumes that some fields are exactly equal in matching pairs. As only the last names and initials are available for most papers, this approach is not suitable as requiring last names to be equal only leaves the initial to be compared.

Instead, we are limiting ourselves to the papers where the full names of the authors are provided. These names are extracted using Python (the code is found in *Appendix C.1*) and analyzed using the `RecordLinkage` packages in R following the tutorial given in *Appendix A* using the Fellegi-Sunter model with the EM-algorithm and equal last names as the blocking strategy. The results with the highest weights are presented in *Table 7.1*, where pairs of matching names are shown.

[1] Only a randomly selected subset of the original data set from Johansson *et al.* (2011) has been used in the evaluation of the introduced methods. This is due to the limitation of the 32-bit Windows installation on the computers used for the calculations.

Name 1	Name 2
Le Cadre, J. -P	Le Cadre, J.-Pierre
Yang, Shanchieh Jay	Yang, Shanchieh J
Chang, KuoChu	Chang, Kuo-Chu
Soysal, Goekhan	Soysal, Gokhan
Mellema, Garfield	Mellema, Garfield R
Wu, Jian Kang	Wu, Jian-Kang
Jilkov, Vesellin P	Jilkov, Vesselin P
Lobbia, Robert B	Lobbia, Robert N
Oxenham, Martin G	Oxenham, Martin
Rheaume, Frangois	Rheaume, Francois
Strat, Thomas	Strat, Thomas M
Wright, Edward J	Wright, Edward

Table 7.1: Possible matches found using the Fellegi-Sunter model using the EM-algorithm and the blocking strategy that last names should be equal.

This method is able to match names with initials and full first names, as well as some smaller spelling errors. An example of the former in that *Le Cadre, J. -P* is matched with *Le Cadre, J.-Pierre* and the latter is exemplified by matching *Jilkov, Vesellin P* with *Jilkov, Vesselin P*. The drawback of this blocking strategy is that last names are required to be equal in each pair, this is a pretty strong assumption to make as spelling differences in last names also occur frequently. We therefore continue by considering other methods as well, the results of each method is then possible to merge using one of several different rules. The simplest is just to combine the matching pairs of each method into a list but other methods (discussed in the previous chapters) includes voting.

7.2 Graph-based methods

The graph-based methods are applied on the co-author network generated from the citation data using the software Bibexcel. The details of the steps used to extract the citation data from web sources to creating a pajek-file is described in the supplemental material to Johansson *et al.* (2011). The resulting network is exported to a pajek-format which is imported in the software R. The following analysis is done using the package `igraph` to calculate the vertex similarities.

As previously discussed, the classification is determined by a limit value and in this experiment the value 0.9 is used for all three measures. A similarity larger than 0.9 results in classifying two nodes as similar. The resulting label from the three applied measures: Jaccard, Dice, and inverse-log weighted similarity, are used in majority voting for determining if nodes are considered similar in the graph-based sense.

The output is used as the possible candidate pairs in the Fellegi-Sunter method using the EM-algorithm and the Jaro-Winkler metric for finding sim-

ilar names. The analysis is done using the implementations in the R-package `RecordLinkage` with the code as presented in *Appendix B.3*. The resulting matches between author names are presented in *Table 7.2*.

Name 1	Name 2
Grindle, C	Girindle, C
Seo, YW	Seo, Y-W
Fisher, JW III	Fisher, JW
Zhang, M	Zhang, Miao
Xu, Q	Xu, QF
Raprey, V	Rapley, V

Table 7.2: Possible matches found using the graph-based voting method with the Jaccard, dice and inverse-log weighted measures. The 70 results have been manually reviewed and the remaining candidates are shown in this table.

As clearly seen in the table, the method is successful in identifying some common problems in citation data from multiple sources. These errors include different spellings, writing out initials or full name, etc. In comparison with usual record matching methods, this graph-based method has somewhat higher accuracy with lower computational complexity. The success of this method depends on the assumption that duplicate nodes share a large fraction of neighbors.

It is therefore not probable that this method always is suitable. However, the method is useful in networked data, e.g. citation information, communication data, and link extracted from web pages. Other measures for comparing vertex similarity are discussed in *Chapter 5.1*. Further studies using these measures are necessary for selecting the optimal measure and the limit values for the classification regions and parameters for weighted voting.

7.3 Text mining-based methods

The abstracts are extracted using a simple Python-script found in *Appendix C.2* and is used as an input to the `lsa` package in R. This package has implemented methods for constructing weighted Term-Document (TD) matrices from text files. Singular value decomposition is applied on the TD-matrix to determine the optimal number of singular values using a Scree plot. The value at which the decomposition is truncated is determined manually as the elbow point in the plot.

The cosine similarity is calculated for each pair of abstracts using the truncated SVD and used as a similarity measure for partitioning the set of abstracts. The clustering is done using the k-means algorithm with the same number of clusters as singular values used. The resulting clustering of abstracts generates a number of groups within which the abstracts have similar contents. The accuracy of this method is undetermined but some overlap in topics are present

Title
Reliable hidden Markov model filtering through coherent lower previsions
State Estimation with Sets of Densities Considering Stochastic and Systematic Errors
Bayesian Estimation with Uncertain Parameters of Probability Density Functions
Nonlinear Fusion of Multi-Dimensional Densities in Joint State Space
Optimal parametric density estimation by minimizing an analytic distance measure
Parameter identification and reconstruction for distributed phenomena based on... Hybrid Density Filter
Predictive analysis network tool for human knowledge elicitation and reasoning
The Hybrid Density Filter for nonlinear estimation based on hybrid conditional... density approximation
Revisiting JDL model for automotive safety applications: The PF2 functional model
Approximate nonlinear Bayesian estimation based on lower and upper densities
Parameterized joint densities with gaussian and gaussian mixture marginals
Optimal mixture approximation of the product of mixtures
Distributed Greedy Sensor Scheduling for Model-based Reconstruction of Space-Time... Continuous Physical Phenomena
Gaussian Filtering using State Decomposition Methods
Extension of the Sliced Gaussian Mixture Filter with Application to Cooperative... Passive Target Tracking

Table 7.3: Titles of papers included in the one of the topic groups found by the k-means algorithm.

in a fast overview, in *Table 7.3* some titles are presented with similar topics created by the SVD.

It is difficult to identify any common theme in the titles presented in *Table 7.3* without any deeper knowledge of what the papers contain. Some frequently found words are *estimation* and *density functions*, which could hint of the underlying concepts that describe this cluster.

Without any deeper analysis of this list of titles, we continue by applying record linkage-method onto the authors of the papers found in one cluster.

As discussed before, authors are assumed to write papers falling into the same cluster and this corresponds to that the authors write papers about the same topic for each conference. The names of authors within each cluster are paired using the Fellegi-Sunter method with the EM-algorithm and no blocking strategy. An example of the result for one randomly selected cluster is shown in *Table 7.4*.

Some names are obvious matches in the table, but a large number of erroneous matched pairs are also present. This is due to only using the initials as first names. The string matching metrics does not work well for only one letter and it is therefore better to use the entire first name. A solution for this problem is to combine several different methods and use voting for determining matching and unmatching pairs.

Another interesting result from using the k-means method on the truncated

SVD of the TD-matrix is the possibility to calculate the similarity between abstracts clustered together. An example of such a similarity calculation is shown in *Figure 7.1*, where the blue indicate large similarity and purple indicate lower (but still quite high) similarity. As clearly seen there are off-diagonal elements with a high amount of similarity, which should (under our assumption) indicate abstract with similar topics. The clusterings appear to be quite distinct with high cohesion as the abstracts clustered together often have high cosine similarity.

7.4 Ensemble methods

The previous two sections discussed the performance of sequential methods as blocking strategies used in combination with the Fellegi-Sunter model. Another approach, previously discussed in this report, is to combine the output from an ensemble of classifiers. In this section, we use both standard field matching methods as well as the same graph-based and text mining-based methods used in the previous two sections. The aim is to investigate if it is possible and better to use an ensemble approach in comparison with a sequential approach.

In *Table 7.5*, the result of some possible matches are presented. Twelve different queries are input into a program which computes the field matching metric (Jaro-Winkler) of the name fields, the vertex similarities of the co-author network (Jaccard and Dice), and the maximum (non-unity) similarity of abstracts written by the two authors. The column CA describes if two authors have appeared in the same paper, indicating that they are co-authors and therefore cannot be duplicates (1 if they have co-authored a paper and 0 otherwise).

As previously discussed, the Jaro-Winkler metric is one of many different methods for comparing shorter text strings, such as names. In this case, the

Name 1	Name 2	Name 1	Name 2
Johansson, F	Johansson, R	Li, H	Li, XR
Johansson, F	Johansson, P	Li, H	Li, Xiao-Bai
Hall, D	Hall, CM	Karakowski, JA	Karakowski, J
Hall, D	Hall, DL	McMullen, SAH	McMullen, MJ
Williams, M	Williams, J	Sudit, D	Sudit, M
Williams, M	Williams, S	Kruger, K	Kruger, M
Das, S	Das Subrata	Guerriero, M	Guerrero, JL
Powell, G	Powell, GM	Levit, I	Levitt, T
Rogova, GL	Rogova, G	Owen, T	Owens S
Brown, M	Brown, T	Chan, M	Chang, KC
Tan, CH	Tan, Ah-Hwee	Chan, M	Chang, KC
Garcia, O	Garcia, J		

Table 7.4: Possible matches found using the text-mining based voting method.

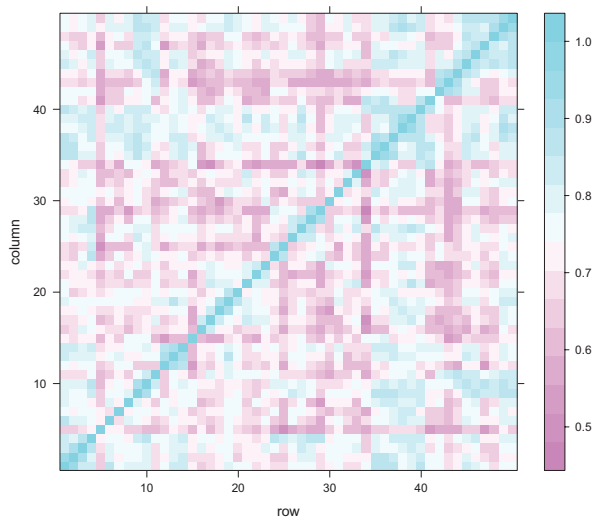


Figure 7.1: Heatmap of the cosine similarity between 50 documents in one cluster generated by the truncated SVD-method and k-means.

metric indicates that a large number of pairs are duplicates (if a limit of say 0.9 is used). The Jaccard and Dice measures are based on the similarity of node neighborhoods in the co-author network with the hypothesis that sharing many co-authors indicate duplicate entities. Three pairs of nodes have similarity 1 and therefore have identical co-author neighborhoods, the remaining pairs do not have any larger vertex similarities.

Finally, we analyze the abstracts using LSA to determine papers with similar topics, indicating evidence for that the pair of names are referring to the same entity. The Cosine measure is calculated as the maximum similarity between the abstracts written by the two authors. It is interesting to conclude that some pair of names that refer to the same entity (e.g. Grindle and Girindle) also have a high abstract similarity (0.811) and other pairs (e.g. Xu Q and Xu QF) have a rather low abstract similarity (0.639). A possible reason for this is that no concern is given if the name was the lead author of the paper or written the abstract. As it is probable to assume that only one author writes the abstract, this similarity could in some cases be misleading. It however evidently works well in some cases and these results also support the assumption that authors often write (at least) a few papers concerning the same topic or topics in the same sub-field at FUSION conferences.

The remaining problem is to train a meta-classifier such as a SVM to classify

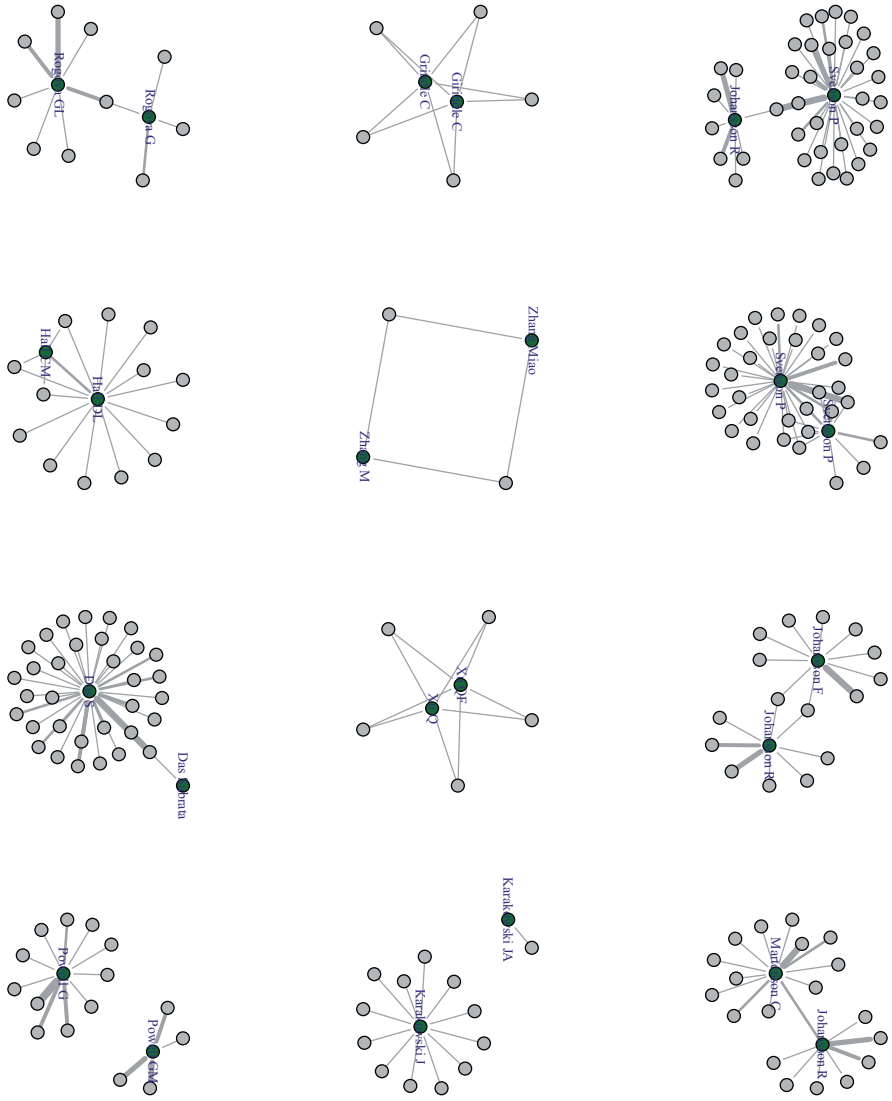


Figure 7.2: Co-author neighborhoods for some pairs of authors (marked with green) in the FUSION citation data. Edge weights indicate the number of papers that authors have written together.

Name 1	Name 2	JW	Jaccard	Dice	Cosine	CA
Svenson P	Johansson R	0.603	0.027	0.053	0.703	0.000
Svensson P	Svenson P	**0.980**	0.265	0.419	0.807	**1.000**
Johansson F	Johansson R	0.964	0.125	0.222	0.733	0.000
Martenson C	Johansson R	0.697	0.000	0.000	0.668	**1.000**
Grindle C	Girindle C	0.937	**1.000**	**1.000**	**0.811**	0.000
Zhang M	Zhang Miao	0.940	**1.000**	**1.000**	0.398	0.000
Xu Q	Xu QF	0.960	**1.000**	**1.000**	0.639	0.000
Karakowski JA	Karakowski J	**0.985**	0.000	0.000	0.680	0.000
Rogova G	Rogova GL	**0.978**	0.100	0.182	0.703	0.000
Hall DL	Hall CM	0.886	0.200	0.333	0.805	**1.000**
Das S	Das Subrata	0.891	0.030	0.059	**0.814**	0.000
Powell G	Powell GM	**0.978**	0.000	0.000	**0.882**	0.000

Table 7.5: The output from the proposed ensemble of weak classifiers comparing two names using the citation data. The field matching metric Jaro-Winkler (JW), the network structure similarity measures (Jaccard and Dice), abstract topic similarity by the LSA with the maximum cosine similarity, and CA indicates if the authors have written a paper together (1) or not (0). Bold faced figures indicate the three largest similarities in each column.

pairs of names as (un)matching using the five metrics presented in *Table 7.5*. For example, a high JW-metric does not necessary indicate that two names refer to the same entity, e.g. Svensson P and Svenson P are two distinct authors. The network similarity is probably the best metric in these types of networks as authors tend to write papers together with friends and colleagues. Abstract similarity could be useful in some instances, but it is difficult to know the amount of involvement the author had in the production of the paper and the content therein. The CA indicator is the most certain metric discussed in this example, as if two authors have co-authored a paper it is impossible that they refer to a single entity.

No evaluation of this method has been done by training a SVM, this is due to problems with finding appropriate pre-labeled data sets. As constructing such test and training sets are very time consuming and has therefore not been done. This is an important future area of study as combination of several different types of metrics probably is a good method for finding a robust and flexible method for entity matching in large data materials containing a large number of different data types.

7.5 Discussion and recommendations

In this chapter, we have tested some of the sequential strategies for finding candidate pairs for further analysis using standard record linkage. This sequential blocking strategy was outlined in the previous chapter and is tested in this

chapter using graph-based methods on the co-author network as well as text mining-methods on the abstracts of the papers in some gathered citation data.

The conclusions are that both methods generate some relevant (but also irrelevant) candidate pairs and greatly reduce the number of pairs to consider using the matching models. Each method finds different candidate pairs and it is therefore important to combine the candidate pairs from all the discussed (and other relevant classification) methods to find many of the duplicate pairs. If these methods does not find all candidate pairs then e.g. the Fellegi-Sunter method can never be used to analyze the text strings for matching misspelled entries etc. As previously discussed, it is important to consider other methods than this sequential blocking method that allows for interactions between e.g. the network structure and the topics of papers. This solution is probably too simple to find all of the relevant pairs to consider using record linkage methods.

The recommendations are therefore to further study the ensemble methods outlined in the previous section using network structures and text-mining as additional weak classifiers in addition to the field matching methods and matching models previously discussed. The selection of an appropriate combination method and weak classifiers should be the focus of further studies in this area. Previous work in this area indicates that majority voting, Dempster-Shafer, and stacking are relevant methods to consider for combining the classifications found by the ensemble.

8 Concluding remarks

In this report, the problem of matching entries containing a large number data fields have been considered. This is often referred to as data integration from a number of heterogeneous data sources. Common problems in data integration include: missing and conflicting information that is organized in different forms. It is therefore necessary to use methods from statistics and machine learning to reason if two entries are referring to the same entity and should be merged into one.

Summary and implications We have provided the reader with an extensive background of earlier work done for matching text strings such as names and addresses. Text string matching is usually done using character-based or token-based similarity metrics, where the latter are better for longer strings and the former for shorter strings such as names. These metrics are an integrate part of the matching models introduced for matching data entries with each other, despite misspellings and conflicting information.

These models include methods from statistics and machine learning, where some methods are supervised and other unsupervised. In general, if good training data is available the supervised methods are the better choice with higher accuracy. If no data can be used for training these models, the Fellegi-Sunter model using the EM-algorithm to estimate the classification regions is a good choice. Despite the large amount of field matching metrics and matching models available for use, there are no generally best methods for every type of data material but there are some methods (including SVM) that perform satisfactory on many types of data materials. The choice of an appropriate classifier is dependent on the data material used and it is difficult to know beforehand which method that gives accurate results.

The report also discusses some implementations of matching models and field matching metrics. The conclusions and recommendations are that there is no stand-alone simple method that is versatile enough for the considered scenario. There exist however some libraries for the software R that are useful for record linkage and for matching data records from heterogeneous data sources. It is therefore recommended to use some library for Java, R, or Python instead of the implementations available.

Data fusion is considered as a good tool for taking different aspects of the data set into consideration. This include e.g. the comparisons of network structures and topics of written text as evidence that two entities are matching or not. We propose two different approaches to include this information into the usual framework for entity matching. The first is a sequential method to screen the set of all pairs for candidate pairs that are further investigated using matching models to identify matching entities. The second method is applying ensemble classification methods to combine an ensemble of weak classifiers, including text mining-methods, vertex similarity methods, and field matching

methods. This latter method is considered the most promising approach for future work in the area of entity matching. This approach is also recommended as previous work has been done at FOI with data fusion methods.

The sequential method is evaluated using a citation data set combined from two different data sources. Therefore some names are spelled differently and the aim is to identify these different spellings of author names, so that they can be merged in the co-author network. The results indicate that each considered method (using last names as blocking strategy, network structures and text mining-methods) find different matching entities and it is therefore important to consider several of these methods to find all matching entities. This also supports the recommendation that data fusion methods are an important future area of research.

Future work This future research should focus on finding a good weighted combination or meta-combination method for ensembles of weak classifiers. In addition, there could exist other good weak classifiers to include in the ensemble that e.g. quantifies similarity in word use and writing styles. Ensemble methods are too computationally demanding for usage without strict blocking strategies. This is an important aspect in all entity matching and some fast and coarse methods for this are needed to meet the requirements for practical usage and implementations.

Other important future methods to evaluate are stacking methods that automatically select the best combination of classifiers for entity matching. This is however a quite complex procedure requiring a large training set and it is therefore more likely that the sequential method with simple Fellegi-Sunter model or the proposed ensemble methods are the better choices.

Lastly, it is important to evaluate the proposed methods using large labeled data sets to train the meta-classifier and then evaluate the performance using e.g. the F-measure or any other related measures of accuracy. A good entity matching method should be flexible in that it can be used on different types of data without any larger modifications. The method should also be robust with a high accuracy, that is effectively finding duplicates and merging them even in noisy and incomplete data. It is doubtful that such a method can be found and still retain it's scalability in large data sets. But as ensemble methods have proven themselves in the past, it is a hopeful avenue for further study.

Bibliography

Adamic, L. & Adar, E. 2003 Friends and neighbors on the Web. *Social Networks*, **25**(3), 211–230. ISSN 03788733. doi:10.1016/S0378-8733(03)00009-1.

Bhattacharya, I. & Getoor, L. 2006 Entity Resolution in Graphs. In Cook, D. & Holder, L. (Editors), *Mining Graph Data*. Wiley.

Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P. & Fienberg, S. 2003 Adaptive Name Matching in Information Integration. *IEEE Intelligent Systems*, **18**(5), 16–23. ISSN 1541-1672. doi:10.1109/MIS.2003.1234765.

Bilgic, M., Licamele, L., Getoor, L. & Shneiderman, B. 2006 D-Dupe: An Interactive Tool for Entity Resolution in Social Networks. In *Visual Analytics Science and Technology (VAST)*. Baltimore.

Boser, B. E., Guyon, I. M. & Vapnik, V. N. 1992 A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, New York, NY, USA. ISBN 0-89791-497-X.

Breiman, L. 1996 Bagging Predictors. *Machine Learning*, **24**(2), 123–140. doi:10.1023/A:1018054314350.

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A. 1984 *Classification and Regression Trees*. Chapman & Hall/CRC, 1 edition. ISBN 0412048418.

Buntine, W. L. 1990 *A Theory of Learning Classification Rules*. Ph.D. thesis, Sydney. URL `citeseer.nj.nec.com/buntine92theory.html`.

Chan, P. K. & Stolfo, S. J. ???? .

Chan, P. K. & Stolfo, S. J. 1993 Toward Parallel and Distributed Learning by Meta-Learning.

Chen, Z., Kalashnikov, D. V. & Mehrotra, S. 2009 Exploiting context analysis for combining multiple entity resolution systems. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 207–218. ACM, New York, NY, USA. doi:http://doi.acm.org/10.1145/1559845.1559869. URL `http://dx.doi.org/http://doi.acm.org/10.1145/1559845.1559869`.

Christen, P. 2008 Febrl: An open source data cleaning, deduplication and record linkage system with a graphical user interface (Demonstration Session. In *In ACM International Conference on Knowledge Discovery and Data Mining*, pages 1065–1068.

Cohen, W. W. 1998 Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. pages 201–212.

Cohen, W. W., Ravikumar, P. & Fienberg, S. E. 2003 A Comparison of String Distance Metrics for Name-Matching Tasks. In Kambhampati, S., Knoblock, C. A., Kambhampati, S. & Knoblock, C. A. (Editors), *IIWeb*, pages 73–78. URL `http://dblp.uni-trier.de/rec/bibtex/conf/ijcai/CohenRF03`.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. & Harshman, R. 1990 Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, **41**(6), 391–407.

Derbeko, P., El-Yaniv, R. & Meir, R. 2002 Variance Optimized Bagging. In *In ECML 2002*, pages 60–71. Springer-Verlag.

Duda, R. O., Hart, P. E. & Stork, D. G. 2001 *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition. ISBN 0471056693.

Dunn Halbert, L. 1946 Record Linkage. *American Journal of Public Health*, **36**(1), 1412–1416.

Elmagarmid, A. K., Ipeirotis, P. G. & Verykios, V. S. 2007 Duplicate Record Detection: A Survey. *Knowledge and Data Engineering, IEEE Transactions on*, **19**(1), 1–16.

Fellegi, I. P. & Sunter, A. B. 1969 A Theory for Record Linkage. *Journal of the American Statistical Association*, **64**(328), 1183–1210. doi:10.2307/2286061.

Freund, Y. & Schapire, R. E. 1997a A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**(1), 119–139.

Freund, Y. & Schapire, R. E. 1997b A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, **55**(1), 119–139. ISSN 00220000. doi:10.1006/jcss.1997.1504.

Gravano, L., Ipeirotis, P. G., Koudas, N. & Srivastava, D. 2003 Text joins for data cleansing and integration in an RDBMS. In *19th International Conference on Data Engineering*, pages 729–731. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1260850`.

Han, J. & Kamber, M. 1992 *A training algorithm for optimal margin classifiers*. COLT '92. ACM. ISBN 0-89791-497-X. doi:10.1145/130385.130401.

Hastie, T., Tibshirani, R. & Friedman, J. H. 2003 *The Elements of Statistical Learning*. Springer, corrected edition. ISBN 0387952845.

Jaro, M. A. 1978 *UNIMATCH: A Record Linkage System*. Bureau of the Census, Washington.

Jaro, M. A. 1989 Advances in Record Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, **84**(406), 414–420. doi:10.2307/2289924.

Johansson, F., Mårtenson, C. & Svenson, P. 2011 A Social Network Analysis of the Information Fusion Community. In *34h Conference on Information Fusion (FUSION)*.

Jurczyk, P., Lu, J. J., Xiong, L., Cragan, J. D. & Correa, A. ???? In *American Medical Informatics Associations (AMIA) Annual Symposium*.

Köpcke, H., Thor, A. & Rahm, E. 2010 Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, **3**, 484–493.

Leicht, E. A., Holme, P. & Newman, M. E. J. 2006 Vertex similarity in networks. *Physical Review E*, **73**(2), 026120+. doi: 10.1103/PhysRevE.73.026120.

Levenshtein, V. I. 1966 Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, **10**, 707+.

Newcombe, H. B., Keneedy, J. M., Axford, S. J. & James, A. P. 1959 Automatic linkage of vital records. *Science*, **130**, 954–959. ISSN 0036-8075.

Polikar, R. 2006 Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, **6**(3), 21–45. ISSN 1531-636X. doi: 10.1109/MCAS.2006.1688199.

Rokach, L. 2010 Ensemble-based classifiers. *Artificial Intelligence Review*, **33**(1), 1–39. ISSN 0269-2821. doi:10.1007/s10462-009-9124-7.

Sariyar, M. & Borg, A. 2010 The RecordLinkage Package: DetectingErrors in Data. *The R Journal*, **2**.

Sariyar, M., Borg, A. & Pommerening, K. 2009 Evaluation of Record Linkage Methods for Iterative Insertions. *Methods of Information in Medicine*, **48**(5), 429–437. ISSN 0026-1270.

Schnell, R., Bachteler, T. & Reiher, J. 2009 Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making*, **9**(1), 41+. ISSN 1472-6947. doi:10.1186/1472-6947-9-41.

Seewald, A. K. & Fürnkranz, J. 2001 Grading Classifiers. Technical Report OEFAI-TR-2001-01, Austrian Research Institute for Artificial Intelligence, Wien, Austria. URL `citeseer.ist.psu.edu/seewald01grading.html`.

Shen, W., DeRose, P., Vu, L., Doan, A. & Ramakrishnan, R. 2007 Source-aware Entity Matching: A Compositional Approach. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 196–205. doi:10.1109/ICDE.2007.367865. URL `http://dx.doi.org/10.1109/ICDE.2007.367865`.

Shlien, S. 1990 Multiple binary decision tree classifiers. *Pattern Recognition*, **23**(7), 757–763. doi:10.1016/0031-3203(90)90098-6.

Verykios, V. 2000 Automating the approximate record-matching process. *Information Sciences*, **126**(1-4), 83–98. ISSN 00200255. doi:10.1016/S0020-0255(00)00013-X.

Winkler, W. E. 2000 Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 667–671. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.175`.

Winkler, W. E. & Thibaudeau, Y. 1987 An Application Of The Fellegi-Sunter Model Of Record Linkage To The 1990 U.S. Decennial Census. In *U.S. Decennial Census. Technical report, US Bureau of the Census*. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.2433`.

Wolpert, D. H. & Macready, W. G. 1995 No Free Lunch Theorems for Search. Technical Report SFI-TR-95-02-010, Santa Fe, NM. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.5447`.

Yancey, W. E. 2005 Evaluating String Comparator Performance for Record Linkage. Technical report, Statistical Research Division, U.S. Census Bureau.

Zhao, H. & Ram, S. 2005 Entity identification for heterogeneous database integration: a multiple classifier system approach and empirical evaluation. *Inf. Syst.*, **30**(2), 119–132. ISSN 0306-4379. doi:10.1016/j.is.2003.11.001.

A Tutorial on the RecordLinkage package

The RecordLinkage package is available for download in the R interface, by entering the following commands into the R Console window:

```
# Tell R to use the same proxy as Windows,
# needs only to be done when at FOI.
setInternet2(TRUE)

# Installs the package from the mirror of your choosing
# (select Sweden in the pop-up window)
install.packages("RecordLinkage")

# Load the library into R
library(RecordLinkage)
```

The package (and it's supporting packages) have now been installed and loaded into R. The first step in most analysis is to import some data from an external file. This is most simply done by using the following command:

```
# Import data from file at M:\folder\file.txt
xdata=read.table("M://folder/file.txt",sep="\t",stringsAsFactors=F);
```

this loads the data in some text file into the variable xdata in R. The `sep="\t"` indicate that the columns are separated by tabs (other common separators are commas "," and white spaces ""), the last argument `stringsAsFactors=F` tells R to treat the data as strings and not as factors in some statistical experiment. The complete syntax for read.table is accessable using the help command, `help(command)` or `?command`, which in this case generates an output (as a html document) beginning with the following:

```
Description
Reads a file in table format and creates a data frame from it,
with cases corresponding to lines and variables to fields in the file.

Usage
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
```

```
          stringsAsFactors = default.stringsAsFactors(),
          fileEncoding = "", encoding = "unknown")
```

Which are all the arguments accepted by the command `read.table`, for more information about the meaning of each argument, see the corresponding help page.

R packages often include some example datasets to use for tutorials and other demonstrations. We follow a modified version of the official tutorial on the RecordLinkage package before presenting some own examples. Some datasets are also included in the RecordLinkage package and are accessible by the following command:

```
data(RLData500)
```

This loads some randomly generated register data into the variable called RLData500. To print the first 5 rows of the loaded data material, write the following

```
> RLdata500[1:5, ]
  fname_c1 fname_c2 lname_c1 lname_c2   by bm bd
1  CARSTEN     <NA>    MEIER     <NA> 1949  7 22
2     GERD     <NA>    BAUER     <NA> 1968  7 27
3   ROBERT     <NA> HARTMANN     <NA> 1930  4 30
4   STEFAN     <NA>    WOLFF     <NA> 1957  9  2
5     RALF     <NA>  KRUEGER     <NA> 1966  1 13
```

which shows us that the data material is divided into a number of fields (columns) presenting first name, last name, and day of birth (year, month, and day).

A.1 Fellegi-Sunter

We continue by carrying out a standard analysis of this data material using some of the field metrics and the Fellegi-Sunter model previously discussed. Firstly, comparison pairs need to be formed from the data material. For large data materials, complete comparisons (all entries are compared with each other) is impractical and some blocking method is used to decrease the number of comparisons. This blocking is often done by applying a simpler field matching method to find duplicate candidates. Comparison pairs are generated in R using:

```
compare.dedup (dataset, blockfld = FALSE, phonetic = FALSE,
  phonfun = pho_h, strcmp = FALSE, strcmpfun = jarowinkler,
  identity = NA, ...)

compare.linkage (dataset1, dataset2, blockfld = FALSE,
```

```
  phonetic = FALSE, phonfun = pho_h, strcmp = FALSE,
  strcmpfun = jarowinkler, identity1 = NA, identity2 = NA, ...)
```

where `dataset`(1 and 2) denotes some different imported sets of data with columns indicating fields. `blockfld` controls which field(s) that should be used for blocking, `identity` to include labels (indicating correct classifications), and the remaining arguments control the field matching methods. See the help file accessible by `?compare.dedup` for more information. The difference between the two introduced commands are that the first evaluate entries in the same data set for duplicates, as the other command tries to match entries from two different data sets together.

The included data set that we are working on also have predetermined labels for matching and unmatching entries. These are typically used to compare the obtain solution with the correct solution. By issuing the following command, it is required that year and month, month and day, or year and day are the same for two entries to be compared (this is the blocking strategy) and also include the identity labels as provided by the data set and the Jaro-Winkler metric to compare strings:

```
> pairs = compare.dedup(RLdata500, identity = identity.RLdata500,
+ blockfld = list(c(5, 6), c(6, 7), c(5, 7)),strcmp=T)
> summary(pairs)
Deduplication Data Set
500 records
571 record pairs
49 matches
522 non-matches
0 pairs with unknown status
```

This indicate that there are 571 record pairs of which 49 are duplicates. Neglecting the block strategy, results in 124750 record pairs to be compared and of these considered pairs 50 are duplicates. By issuing the blocking strategy, we are missing one duplicate entry but greatly reduces the number of comparisons that have to be done.

After the comparison pairs have been generated, the next step is to apply some matching method to find entries that are linked, unlinked, or possibly linked with each other. The RecordLinkage package provide two different methods for constructing the weighting of the data fields: using Fellegi-Sunter model with the EM-algorithm and by using the EpiLink approach. Continuing by applying the EM-method, using the following command:

```
> weightedpairs = emWeights(pairs)
> summary(weightedpairs)
[...]
[-25,-20] (-20,-15] (-15,-10]  (-10,-5]    (-5,0]     (0,5]
      351       160         7         0         0         3
```

```
(5,10]   (10,15]   (15,20]   (20,25]   (25,30]
7         7        33         2         1
```

which outputs the distribution of the weights. To classify the pairs of data entries as matching, unmatching, or undecided, using the EM-algorithm, do the following:

```
> xpairs = emClassify(weightedpairs)
> summary(xpairs)
53 links detected
0 possible links detected
518 non-links detected

alpha error: 0.000000
beta error: 0.007663
accuracy: 0.992995

Classification table:
           classification
true status   N   P   L
      FALSE 518   0   4
      TRUE    0   0  49
```

where the output indicated no link (N), possible link (P), and detected link (L).

A.2 Supervised classification

We continue by using the previous data set as training data for some classifiers that will classify a larger data set called `RLdata10000`. This data set is generated using the same random method as the smaller `RLdata500`. To train a classifier, some comparisons need to be formed and labels included to indicate if pairs match or not. By using the same blocking strategy as in the previous example, classification training pairs and evaluation pairs are formed by:

```
>data(RLdata500)
>data(RLdata10000)

>trainingpairs=compare.dedup(RLdata500,
identity=identity.RLdata500,strcmp=T);
>evaluationpairs=compare.dedup(RLdata10000,
identity = identity.RLdata10000,strcmp=T,
blockfld = list(c(5, 6), c(6, 7), c(5, 7)));
```

RecordLinkage includes five different methods for supervised classification: CART, CART with bagging(default: 25 bootstrap replicates)/boosting(default:

50 iterations), SVM (radial kernel as default), and single-layered neural networks. These classifiers are trained by using the following commands:

```
>xcart=trainSupv(trainingpairs, method = "rpart")
>xbagging=trainSupv(trainingpairs,method = "bagging")
>xboosting=trainSupv(trainingpairs,method = "ada")
>xsvm=trainSupv(trainingpairs, method = "svm")
>xnnet=trainSupv(trainingpairs, method = "nnet")
```

which takes some time (especially boosting) for this large training data set. When the classifiers have been trained, they can be used to classify new data by the following:

```
>outcart=classifySupv(xcart,evaluationpairs)
>outbagging=classifySupv(xbagging,evaluationpairs)
>outboosting=classifySupv(xboosting,evaluationpairs)
>outsvm=classifySupv(xsvm,evaluationpairs)
>outnnet=classifySupv(xnnet,evaluationpairs)
```

which returns the classifications of the larger data set `RLdata10000`, statistics and accuracy are found using the summary command. The corresponding output for each of the five methods are presented in *Table A.1*.

Method	α-error	β-error	Accuracy
CART	**0.030**	< 0.001	0.999
CART with boosting	0.051	**0.000**	0.999
CART with bagging	0.056	0.000	0.999
SVM	0.054	0.000	0.999
Single-layered neural network	1.000	0.000	0.995

Table A.1: The error rates and accuracy of five different supervised methods trained with the data set RLdata500, tested on the larger set RLdata10000.

The simple CART method seems to give the lowest α-error when applied on this data set. Using CART with boosting results in a higher α-error but lower β-error in comparison with the CART method. This could be preferred in some applications but we do not go into any deeper analysis at this stage, these examples only serve to give some hands-on experience with R and this library.

A.3 Unsupervised classification

The last type of record matching methods are the unsupervised methods, this library supports two different clustering based methods: k-means and bagged k-means. The latter performs a number of k-means clustering on bootstrap

sampled data and clusters the resulting centroids using agglomerative hierarchical clustering. The methods are demonstrated by using the `RLdata500` data set by the following commands:

```
>data(RLdata500)
>xpairs=compare.dedup(RLdata500,identity=identity.RLdata500,strcmp=T);
>outkmeans=classifyUnsup(xpairs, method="kmeans")
>outbclust=classifyUnsup(xpairs, method="bclust")
>summary(outkmeans)
>summary(outbclust)
```

which returns the data presented in *Table A.2*. Compared with the supervised methods, shown in *Table A.1*, it is not unexpected that the error rates are higher and accuracy lower for the unsupervised methods. The performance of the bagged clustering is even worse than for the k-means method.

Method	α-error	β-error	Accuracy
k-means	0.000	0.695	0.305
Bagged clustering	0.020	0.894	0.106

Table A.2: The error rates and accuracy of two different unsupervised methods tested on the data set RLdata500.

B R-implementations

This section contains a implementation of the WHIRL-metrics and some implementations to compare entries in the citation data from Johansson *et al.* (2011). The implementations requires the libraries `RecordLinkage`, `lsa` and `igraph` available from the CRAN library using the commands presented in the tutorial of the Record Linkage-package, found in the previous appendix.

B.1 WHIRL metrics

Record Linkage (and Python) lacks an implementation of the WHIRL-metrics. This is a problem as these token-based similarity metrics are quite useful to compare longer strings. Therefore, a somewhat large effort has been devoted to implement these metrics in R (original, soft and q-gram based) and the resulting function is presented in this section.

The input is a vector of strings and the output is a matrix where the element found in row i and column j corresponds to the similarity between words i and j in the input vector. The function is called with an additional parameter called `method` which determines the version of the WHIRL metric used: `original` for the basic version of the metric, `qgram` for the version using q-grams instead of words, and `soft` for the version using the Jaro-Winkler metric to determine if words are close enough to be compared.

```
1  ## ------------------------------------------------
2  ## WHIRL implementations
3  ## by Johan Dahlin (2011-07-12)
4  ##
5  ## Input:       A directory of text files
6  ##              A WHIRL-version
7  ## Output:      A n x n-matrix with similarities
8  ##              between strings i and j in element ij.
9  ## Comment: An implementation of Term-Frequency-
10 ##              Inverse-Document-Frequency discussed in
11 ##              e.g. Cohen (1998).
12 ##
13 ## ------------------------------------------------
14
15 ## ---------------------------------------
16 ## Initialization and parameters
17 ## ---------------------------------------
18
19 ## Load the libraries needed
20 library(lsa)
21 library(RecordLinkage)
22 library(tau)
23
24 ## ---------------------------------
25 ## Parameters
26 ## ---------------------------------
```

```
27
28  # the method to use (original, qgrams, or soft)
29  method="soft"
30
31  # the directory containing text files with author names
32  directory="M://fusionstrasket/authors";
33
34  # the length of the q-grams generated
35  q=3;
36
37  # the limit for the J-W for close words in the soft metric.
38  simlimit=0.9;
39
40  ## ------------------------------------
41  ## Subroutines
42  ## ------------------------------------
43
44  ## ------------------------------------
45  ## Soft-WHIRL implementation
46  ## Adds similarity of strings similar by
47  ## the Jaro-Winkler metric
48  ## ------------------------------------
49
50  softWHIRL <- function(zdata,ydata,simlimit=0.9,output) {
51          extrasim=0;
52
53          # Find all words in each string that are not exactly the same
54          commonwords=union(which(zdata!=0),which(ydata!=0));
55          zdata[commonwords]=0; ydata[commonwords]=0;
56          wordsinA=which(zdata!=0); wordsinB=which(ydata!=0);
57
58          # Do only if there exists any more words
59          if ((length(wordsinA) > 0) & (length(wordsinB) > 0)) {
60
61                  # Calculate the Jaro-Winkler metric for every pair of words
62                  simweight=matrix(0,nrow=length(wordsinA),ncol=length(wordsinB))
63                  for (k in 1:length(wordsinA)) {
64                          simweight[k,]=jarowinkler(names(wordsinA[k]),names(wordsinB)
                                );
65                  }
66
67                  # Calculate the extra similarity as the product of the weights
68                  # and the maximum similarity
69                  termstosum=which(simweight >= simlimit,arr.ind=T);
70                  extrasim=sum(xdata[termstosum[,1]]*ydata[termstosum[,2]])*max(
                        simweight)
71          }
72
73          # Output the old simularity with the new added weight
74          output=output+extrasim;
75  }
76
77  ## ------------------------------------
78  ## Q-gram-WHIRL implementation
79  ## Uses similarity between q-grams instead
80  ## of words.
```

```
81  ## ------------------------------------
82
83  qgramWHIRL <- function(directorytoread,q) {
84
85          # Create a directory for temporary files
86          td = tempfile(); dir.create(td);
87           filestoread =dir(directorytoread, full .names=T);
88           Nfiles =length(filestoread)
89
90          # For all text  files , find the q-grams for all the strings
91          for (i in 1:length(filestoread)) {
92                  temp=gsub("\t"," ",read.table(filestoread[i],sep="\n",stringsAsFactors
                        =F));
93                  qgrams=textcnt(temp,method="ngram",n=q);
94                  qgrams=qgrams[which(str_length(names(qgrams))==q)];
95                  qgramsT=c();
96                  for (j in 1:length(qgrams))
97                          qgramsT=c(qgramsT,rep(names(qgrams[j]),qgrams[j]));
98                  write(qgramsT,file=paste(td, i,sep="/"))
99          }
100
101         # Create a TD-matrix using q-grams instead of words
102         xdata=textmatrix("td",stemming=FALSE,stopwords=NULL,minWordLength=1);
103 }
104
105 ## ------------------------------------
106 ## - Main program -
107 ## Primary WHIRL implementation
108 ## ------------------------------------
109
110 ## Import all abstracts into a Term-Document matrix and weight it with the
111 ## TF-IDF method to get the weights to use in the following method.
112
113 if (method=="qgrams") {
114                 # Create TD-matrix with qgrams
115                 xdata=qgramWHIRL(directory,q=3)
116         } else {
117                 # Create TD-matrix with words
118                 xdata=textmatrix(directory,stemming=FALSE,stopwords=NULL,
                        minWordLength=1);
119 }
120
121 # Weight the TD-matrix by the TF-IDF method
122 xdata=lw_logtf(xdata)*gw_idf(xdata);
123
124 ## Calculate cosine measure for two strings using the weights as similarity
125 NDocs=dim(xdata)[2];
126  similarity =matrix(nrow=NDocs,ncol=NDocs);
127 for (i in 1:NDocs) {
128         for (j in i:NDocs) {
129                  similarity [i,j]=sum(xdata[,i]*xdata[,j])/sqrt(sum(xdata[,i]^2)*sum(
                        xdata[,j]^2));
130
131                 # add similarity of close words if the soft metric is used
132                 if (method=="soft") similarity[i,j]=softWHIRL(xdata[,i],xdata[,j],
                        simlimit, similarity [i,j]) ;
```

```
133             }
134             similarity [i, i]=0;
135     }
136
137     ## Find elements with high similarity and print the author names in a table
138     maxsims=which(similarity>0.8,arr.ind=T);
139     maxsims=cbind(colnames(xdata)[maxsims[,1]],colnames(xdata)[maxsims[,2]])
140
141     # Get the author names from the text files
142     simauthornamesA=c(); simauthornamesB=c();
143     for (i in 1:dim(maxsims)[1]) {
144                         simauthornamesA[i]=gsub("\t",", ",read.table(paste("M://
                                fusionstrasket/authors/",
145                                     maxsims[i,1],sep=""),sep="\n",stringsAsFactors=F));
146                         simauthornamesB[i]=gsub("\t",", ",read.table(paste("M://
                                fusionstrasket/authors/",
147                                     maxsims[i,2],sep=""),sep="\n",stringsAsFactors=F));
148     }
149
150     # Create a nicer table and present for user
151     yy=cbind(simauthornamesA,simauthornamesB)
152     yy=yy[-which(yy[,1]==yy[,2]),];
153
154     ## ------------------------------------
155     ## End of file
156     ## ------------------------------------
```

B.2 Blocking strategy

The implementation using the Fellegi-Sunter method with the EM-algorithm and requiring equal last names as a blocking strategy. This code uses the output from the python-file in *Appendix C.1* as an input and returns a list a pairs of name that are candidates of being duplicates.

```
1   ## ------------------------------------------------
2   ## Fellegi-Sunter method with EM-algorithm
3   ## using last name as a blocking strategy
4   ## by Johan Dahlin (2011-06-29)
5   ##
6   ## ------------------------------------------------
7
8   # Load library and text file with names
9   library(RecordLinkage);
10  pathtofile ="C://Documents and Settings/johdah/My Documents/My Dropbox/work/
        entity matching/fusionstrasket/";
11  xdata=read.table(paste(pathtofile,"IFochFUSION2002till2010filtred.doc",sep=""),sep="
        \t",stringsAsFactors=F);
12
13  # Create pairs to compare
14  xpairs=compare.dedup(xdata,blockfld=1,strcmp=T);
15  summary(xpairs)
16
17  # Find weights to pairs by the EM-algorithm and print the most similar pairs.
18  xweighted=emWeights(xpairs);
```

```
19 emresult=emClassify(xweighted);
20 summary(emresult)
21 getPairs(emresult,min.weight=0)
22
23 ## ------------------------------------------------
24 ## End of file
25 ## ------------------------------------------------
```

B.3 Graph-based methods

The implementation to find similar authors by the co-author network returned by the BibExcel application. The neighborhood of each node is found and the Jaccard, Dice and inverse-log weighted similarity of each pairs of nodes are calculated. Pairs of nodes with similarity greater than or equal to 0.9 are considered similar. The output of the three measures are combined using a simple majority voting rule.

```
1  ## ------------------------------------------------
2  ## Graph similarity to find duplicate nodes
3  ## by Johan Dahlin (2011-06-29)
4  ##
5  ## Input:      A pajek-network file (.net)
6  ## Output:     A list of nodes that have the same
7  ##             neighbor structure and may therefore
8  ##             be duplicates.
9  ##
10 ## ------------------------------------------------
11
12 # Parameters: Limit values for when nodes are considered similar
13 highJac=0.9;            # limit for the Jaccard measure
14 highDic=0.9;            # limit for the Dice measure
15 highInv=0.9;            # limit for the inverse log measure
16
17 freqlimit=2;            # how many methods need to find that pairs
18                                 # of nodes are similar before the ensemble
19                                 # thinks that the pair is similar.
20
21 # Load the library and the network data file generated for pajek
22 # change path to your file.
23 library(igraph)
24 pathtofile="C://Documents and Settings/johdah/My Documents/My Dropbox/work/
       entity matching/fusionstrasket/";
25 nameoffile="IFOCHFUSION2002TILL2010.net";
26 G=read.graph(paste(pathtofile,nameoffile,sep=""),"pajek")
27
28 # Locate the larget component and delete all other nodes and nodes with degree one
29 # Save the old labels
30 largestsub=as.numeric(names(which.max(table(clusters(G)$membership))));
31 removed=union(which(clusters(G,)$membership!=largestsub),which(degree(G)==1));
32 V(G)$label=V(G);
33 G=delete.vertices(G,(removed-1));
34
35 # Calculate the similarity for each pair of nodes using:
36 # Jaccard, dice and inverse weighted log similarites.
```

```
# See report for details
GsimJac=similarity.jaccard(G,mode="all");
GsimDic=similarity.dice(G,mode="all");
GsimInv=similarity.invlogweighted(G,mode="all");

# Find pairs of nodes with high similarity in each method
highsimJac=t(apply(which(GsimJac>highJac,arr.ind=T),1,sort));
highsimDic=t(apply(which(GsimDic>highDic,arr.ind=T),1,sort));
highsimInv=t(apply(which(GsimInv>highInv,arr.ind=T),1,sort));

# Find pairs of nodes that are similar in all three measures
# count the frequency of being considered similar and return it
simfreq=rep(0,dim(highsimJac)[1]);
for (i in 1:(dim(highsimJac)[1])) {
        simnode1=which(highsimDic[,1]==highsimJac[i,1]);
        simnode2=which(highsimDic[,2]==highsimJac[i,2]);
        simfreq[i]=simfreq[i]+length(intersect(simnode1,simnode2))

        simnode1=which(highsimInv[,1]==highsimJac[i,1]);
        simnode2=which(highsimInv[,2]==highsimJac[i,2]);
        simfreq[i]=simfreq[i]+length(intersect(simnode1,simnode2))
}

highsimJac=cbind(highsimJac,simfreq);

# Find candidate pairs of nodes that are found similar by
# at least freqlimit no. methods.
candidates=highsimJac[which(highsimJac[,3]>(freqlimit-1),arr.ind=T),1:2]
candidates=cbind(candidates[,1],candidates[,2])
candidates=candidates[-which(candidates[,1]==candidates[,2]),]

# Return a list of names of the authors that are considered similar
candidatenames=cbind(V(G)$id[candidates[,1]],V(G)$id[candidates[,2]])
print(candidatenames);

## ------------------------------------------------
## End of file
## ------------------------------------------------
```

B.4 Text mining-based methods

The implementation to find similar authors by the text-mining methods, grouping similar abstracts together. The names of the authors of each group of papers are compared using standard record linkage methods. The input data is found using the Python script discussed in *Appendix C.2*, which creates three directories containing the abstracts, author names and titles each in a separate text file. See the Python and R codes for more details regarding the format of the input. The implementation uses LSA to find a truncated SVD of the Term Document-matrix and the documents are clustered by the k-means algorithm into as many clusters as there are terms in the truncated decomposition. This code returns the groupings of documents and the authors in each group, which can be analyzed using the Fellegi-Sunter method outlined in the tutorial.

```
1  ## ------------------------------------------------------------
2  ## Text similarity to find duplicate nodes
3  ## by Johan Dahlin (2011-06-29)
4  ##
5  ## Input:       A directory of abstracts, authors and titles
6  ## Output:      The authors in clusters of similar abstracts
7  ##
8  ## ------------------------------------------------------------
9
10 ## ------------------------------------------------------------
11 ## Latent Semantic Analysis
12 ## ------------------------------------------------------------
13
14 # Load LSA library and stop words
15 library(lsa)
16 data(stopwords=stopwords_en);
17
18 # Import all abstracts into a Term-Document matrix and weight it with the
19 # TF-IDF method, then calculate the full SVD of the TD-matrix
20 xdata=textmatrix("M://fusionstrasket/abstracts",stopwords=stopwords_en,minGlobFreq
       =10);
21 xdata1=lw_logtf(xdata)*gw_idf(xdata);
22 xdata1LSA=lsa(xdata1, dims=dimcalc_raw())
23
24 # Plot a scree plot for identification of the optimal number of singular values
25  fulleigenvalues =xdata1LSA$sk;
26 plot(xdata1LSA$sk,type="l"); identify(xdata1LSA$sk);
27 optimaldims=50
28
29 # Perform a truncated SVD with the selected number of singular values
30 xdata1LSA=lsa(xdata1,dims=optimaldims)
31 xdata1LSAtext=as.textmatrix(xdata1LSA)
32
33 # Calculate the cosine measure of all the abstracts and cluster
34 # the abstract using k-means with the cosine measure and the selected
35 # number of clusters from the scree plot
36 xdata1cossim=cosine(xdata1LSAtext);
37 xdata1clustkmeans=kmeans(xdata1cossim,optimaldims);
38
39 ## ------------------------------------------------------------
40 ## Extract similar authors
41 ## ------------------------------------------------------------
42
43 # Preallocate matrices for authornames
44  filesauthors =matrix(0,nrow=optimaldims,ncol=500);
45 filenamesauthors=matrix(0,nrow=optimaldims,ncol=500);
46
47 # Extract the authors included in each cluster
48 similarauthors=matrix(0,nrow=optimaldims,ncol=1000);
49  similartitels =matrix(0,nrow=optimaldims,ncol=1000);
50
51 for (i in 1:optimaldims) {
52           temp=which(xdata1clustkmeans$cluster==i);
53            filesauthors [i ,1:length(temp)]=temp;
54           filenamesauthors[i ,1:length(temp)]=names(temp);
55            authorsincluster=c(); titels =c(); titelsincluster =c();
```

```
56          for (k in names(temp)) {
57                  ktransformed=gsub("abs","authors",k,fixed=T);
58                  temp2=read.table(paste("M://fusionstrasket/authors/",ktransformed,
                        sep=""),
59                      sep="\t",stringsAsFactors=F)
60                  authorsincluster=c(authorsincluster,temp2);
61                  ktransformed=gsub("abs","titels",k,fixed=T);
62                  temp2=read.table(paste("M://fusionstrasket/titels/",ktransformed,sep
                        =""),
63                      sep="\t",stringsAsFactors=F)
64                   titelsincluster =c( titelsincluster ,temp2);
65          }
66          names(authorsincluster) <- NULL
67          names(titelsincluster) <- NULL
68          authorsincluster=unlist(authorsincluster);
69           titelsincluster =unlist( titelsincluster );
70          if (length(authorsincluster) > 0)
71                  similarauthors[i,(1:length(authorsincluster))]=authorsincluster;
72          if (length( titelsincluster ) > 0)
73                   similartitels [i,(1:length( titelsincluster ))]= titelsincluster ;
74  }
75
76  # Present the authors found in each cluster
77  for (i in 1:optimaldims) print(c(i,similarauthors[i,which(similarauthors[i,] !=0 )]))
78
79  # For some cluster i=7, find all the authors and create a table of their splitted names
80  i=7;
81  print(c(i,similarauthors[i,which(similarauthors[i,] !=0 )]))
82  print(c(i, similartitels [i,which(similartitels[i,] !=0 )]))
83  ioo=similarauthors[i,which(similarauthors[i,] !=0 )]
84
85  comparablenames=matrix(nrow=length(ioo),ncol=2);
86  for (i in 1:length(ioo)) {
87          temp=unlist(strsplit(ioo[i]," ")) ;;
88          if (length(temp)>2) comparablenames[i,]=temp[2:3];
89          if (length(temp)==2) comparablenames[i,]=temp;
90          if (length(temp)==1) comparablenames[i,1]=temp;
91  }
92  ## ------------------------------------------------
93  ## Use the RecordLinkage package to find possible matches
94  ## ------------------------------------------------
95
96  library(RecordLinkage)
97
98  xcompairs=compare.dedup(comparablenames,strcmp=T)
99  xemweights=emWeights(xcompairs); summary(xemweights);
100
101 xpairs=emClassify(xemweights); summary(xpairs);
102 getPairs(xpairs,min.weight=2,max.weight=4)
103
104 ## ------------------------------------------------
105 ## Create Heat plots of the cosine similarity for each
106 ## cluster
107 ## ------------------------------------------------
108
109 names(xdata1clustkmeans$cluster[which(xdata1clustkmeans$cluster==19)])
```

```
110
111 require(lattice);
112 simmatrix=array(0,dim=c(optimaldims,100,100));
113 for (k in 1:optimaldims) {
114         temp=filesauthors[k,which(filesauthors[k,] !=0)]
115         for (i in 1:length(temp)) {
116                 for (j in 1:length(temp)) {
117                         simmatrix[k,i,j]=cosine(xdata1LSAtext[,temp[i]],xdata1LSAtext[,
                                temp[j]])
118                 }
119         }
120 }
121
122 i=10; nonzerocoords=sum(simmatrix[i,1,]!=0);
123 levelplot (simmatrix[i,1:nonzerocoords,1:nonzerocoords])
124
125 levelplot (xdata1cossim[which(xdata1clustkmeans$cluster==7),which(xdata1clustkmeans
      $cluster==7)])
126
127 hh=c(); for (i in 1:optimaldims) hh[i]=length(which(xdata1clustkmeans$cluster==i))
128
129 ## ------------------------------------------------
130 ## End of file
131 ## ------------------------------------------------
```

B.5 Ensemble methods

The implementation of the subroutines for the ensemble method. The three functions contained in the R-file are the following:

- `coauthornetworksim`: calculates the vertex similarity (Jaccard and Dice) and also returns all the author names.
- `abstacttextanalysis`: calculates the maximum cosine similarity between abstracts written by the two authors using LSA.
- `ensembleentitymatching`: script to calculate the different similarities and return them for evaluation.

The input to the script is a matrix of names where each row contain the two author names to be investigated. The script can also use the LSA-matrix created by the abstracts as an input, this is due to the problem with 32-bit computers and the amount of abstracts to be analyzed, only 64-bit computer can do this due to memory restrictions. Therefore it is efficient to construct the matrix and then save it for further use, as it contains all the abstracts. The script calls the two subroutines in the correct sequence and returns a matrix with the similarities calculated for each pair of author names.

```
1 ## ------------------------------------------------
2 ## Ensemble Entity matching methods
3 ## by Johan Dahlin (2011-08-09)
4 ##
```

```
## Input:       A matrix of names
## Output:      A matrix of names and similarity measures
##
## ------------------------------------------------------------

##------------------------------
## Co-author network similarity
##------------------------------

coauthornetworksim <- function(pathtofiles,name1,name2,plotting=FALSE,authornames
    =NULL) {

## Construct a matrix with all authors of all papers
filestoopen=dir(pathtofiles); NDocs=length(filestoopen);

if (is.null(authornames)) {
        authornames=matrix(nrow=NDocs,ncol=20)
        for (nn in 1:NDocs) {
                temp2=unlist(read.table(paste(pathtofiles,"authors",nn,".txt",sep=""),
                    sep="\t",stringsAsFactors=F));
                if (length(temp2) > 0) authornames[nn,1:length(temp2)]=temp2;
        }

        authornames=str_trim(authornames)
}

## Find the paper number where the name string is found as an author
name1indices=arrayInd(which(name1==authornames),dim(authornames));
name2indices=arrayInd(which(name2==authornames),dim(authornames));

## Construct a co-author network
coauthorstoname1=matrix(nrow=dim(name1indices)[1],ncol=19);
coauthorstoname2=matrix(nrow=dim(name2indices)[1],ncol=19);

# Find the names of all co-authors
for (i in 1:dim(name1indices)[1]) coauthorstoname1[i,]=authornames[name1indices[i,1],-
    name1indices[i,2]]
for (i in 1:dim(name2indices)[1]) coauthorstoname2[i,]=authornames[name2indices[i,1],-
    name2indices[i,2]]

if ((dim(coauthorstoname1)[1]==0) | (dim(coauthorstoname2)[1]==0)) print("ERROR:
    Cannot find author name!!")

# Find unique co-authors and merge the two vectors with the author names included
coauthors1=unique(coauthorstoname1[-which(is.na(coauthorstoname1))])
coauthors2=unique(coauthorstoname2[-which(is.na(coauthorstoname2))])
allcoauthors=union(coauthors1,c(coauthors2,name1,name2));

## Construct an co-author network adjancency matrix
coauthoradjmatrix=matrix(0,nrow=length(allcoauthors),ncol=length(allcoauthors));
colnames(coauthoradjmatrix) <- allcoauthors;
rownames(coauthoradjmatrix) <- allcoauthors;

# Add all connections with author 1
indexname1=which(name1==allcoauthors)
for (i in 1:dim(name1indices)[1]) {
```

```
56          uniquecoauthorofpaperi=coauthorstoname1[i,-which(is.na(coauthorstoname1[i,]))
                ]
57          if (length(uniquecoauthorofpaperi) > 0) {
58                  for (j in 1:length(uniquecoauthorofpaperi)) {
59                          index2=which(uniquecoauthorofpaperi[j]==allcoauthors);
60                          coauthoradjmatrix[indexname1,index2]=coauthoradjmatrix[
                                indexname1,index2]+1;
61                          coauthoradjmatrix[index2,indexname1]=coauthoradjmatrix[index2,
                                indexname1]+1;
62                  }
63          }
64  }
65
66  # Add all connections with author 2
67  indexname2=which(name2==allcoauthors)
68  for (i in 1:dim(name2indices)[1]) {
69          uniquecoauthorofpaperi=coauthorstoname2[i,-which(is.na(coauthorstoname2[i,]))
                ]
70          if (length(uniquecoauthorofpaperi) > 0) {
71                  for (j in 1:length(uniquecoauthorofpaperi)) {
72                          index2=which(uniquecoauthorofpaperi[j]==allcoauthors);
73                          coauthoradjmatrix[indexname2,index2]=coauthoradjmatrix[
                                indexname2,index2]+1;
74                          coauthoradjmatrix[index2,indexname2]=coauthoradjmatrix[index2,
                                indexname2]+1;
75                  }
76          }
77  }
78
79  xgraph=graph.adjacency(coauthoradjmatrix,mode="undirected",add.rownames=T,
        weighted=T)
80
81  # Plot the co-author network with green color indicating the authors of interest
82  if (plotting) {
83          l <- layout.fruchterman.reingold(xgraph,weights=E(xgraph)$weights); l <-
                layout.norm(l, -1,1, -1,1) ;
84          vertexcolor=rep("grey",length(allcoauthors));
85          vertexcolor[indexname1]="darkgreen"; vertexcolor[indexname2]="darkgreen";
86          vertexlabel=matrix(nrow=length(allcoauthors),ncol=1);
87          vertexlabel[indexname1]=V(xgraph)$name[indexname1]; vertexlabel[indexname2
                ]=V(xgraph)$name[indexname2];
88          plot(xgraph,vertex.label=vertexlabel,layout=l,vertex.color=vertexcolor,edge.
                width=E(xgraph)$weight)
89  }
90
91  # Calculate some simularity measures
92  XsimJac=similarity.jaccard(xgraph,mode="all")[indexname1,indexname2];
93  XsimDic=similarity.dice(xgraph,mode="all")[indexname1,indexname2];
94  XsimInv=similarity.invlogweighted(xgraph,mode="all")[indexname1,indexname2];
95
96          list(jaccard=XsimJac,dice=XsimDic,inverselog=XsimInv,authornames=
                authornames,
97                  name1indices=name1indices,name2indices=name2indices)
98  }
99
100
```

```
##------------------------------
## Abstract text analysis
##------------------------------
abstacttextanalysis <- function(name1indices,name2indices,xdataLSAtext=NULL) {
        generatedTextmatrix=FALSE;

        if (is.null(xdataLSAtext)) {
                # Load LSA library and stop words
                library(lsa)
                data(stopwords=stopwords_en);

                # Import all abstracts into a Term-Document matrix and weight it with
                      the
                # TF-IDF method, then calculate the full SVD of the TD-matrix
                xdata=textmatrix("M://fusionstrasket/abstracts",stopwords=stopwords_
                      en,minGlobFreq=10);
                xdata=lw_logtf(xdata)*gw_idf(xdata);

                # Perform a truncated SVD with the selected number of singular values
                xdataLSA=lsa(xdata, dims=50)
                xdataLSAtext=as.textmatrix(xdataLSA)

                generatedTextmatrix=TRUE;
        }

        # Find the abstracts written by each author
        abstractfilenames1=paste("abs",name1indices[,1],".txt",sep="");
        abstractfilenames2=paste("abs",name2indices[,1],".txt",sep="");

        # Compare all pair of abstracts and return the cosine measure
        abstractcossim=c(); mm=1;
        for (i in 1:length(abstractfilenames1)) {
                for (j in 1:length(abstractfilenames2)) {
                document1=which(colnames(xdata1LSAtext)==abstractfilenames1[i])
                document2=which(colnames(xdata1LSAtext)==abstractfilenames2[j])
                if (length(document1>0) & length(document2>0))
                        abstractcossim[mm]=cosine(xdata1LSAtext[,document1],
                              xdata1LSAtext[,document2]);
                mm=mm+1;
                }
        }

        # Return the relevant data (maximum cosine similarity) and the LSA-matrix (if
              generated)
        if (length(which(abstractcossim==1))>0) abstractcossim=abstractcossim[-
              which(abstractcossim==1)];
        if (generatedTextmatrix==TRUE) list(maxabssim=max(abstractcossim,na.rm=
              T),xdataLSAtext=xdataLSAtext);
        if (generatedTextmatrix==FALSE) list(maxabssim=max(abstractcossim,na.rm
              =T));
}



##------------------------------
## Ensemble entity matching
```

```
150 ##------------------------------
151 ensembleentitymatching <- function(namevector,xdata1LSAtext=NULL,plotting=
        TRUE) {
152
153         # Prepare window for plotting the network neighborhoods
154         if (plotting) {
155                 Ncomparisons=dim(namevector)[1];
156                 Nrows=floor(sqrt(Ncomparisons));
157                 Ncols=ceiling(Ncomparisons/Nrows);
158                 x11()
159                 par(mfrow=c(Nrows,Ncols));
160         }
161
162         # For each pair of names, do the analysis
163         output=matrix(nrow=dim(namevector)[1],ncol=7)
164         for (i in 1:dim(namevector)[1]) {
165
166                 # Calculate the co-author network and vertex similarity
167                 if (i==1) Xcoauthoroutput=coauthornetworksim("M://fusionstrasket/
                        authors/",
168                         namevector[i,1], namevector[i,2], plotting)
169                 if (i!=1) Xcoauthoroutput=coauthornetworksim("M://fusionstrasket/
                        authors/",
170                         namevector[i,1], namevector[i,2], plotting, authornames=
                                Xcoauthoroutput$authornames)
171
172                 # Analyze the abstracts written by the two authors
173                 Xcoabsoutput=abstacttextanalysis(Xcoauthoroutput$name1indices,
174                         Xcoauthoroutput$name2indices,xdata1LSAtext);
175
176                 # Have the authors written a paper together?
177                 coauthors=0;
178                 if (length(intersect(Xcoauthoroutput$name1indices[,1],
                        Xcoauthoroutput$name2indices[,1]))>0) coauthors=1;
179
180                 X=c();
181                 X[1]=jarowinkler(namevector[i,1],namevector[i,2]);      # Calculate The
                        Jaro-Winkler metric
182                 X[2]=Xcoauthoroutput$jaccard;                                   # Graph
                        -based and text-mining based metrics
183                 X[3]=Xcoauthoroutput$dice;
184                 X[4]=Xcoabsoutput$maxabssim;
185                 X[5]=coauthors;
186                 output[i,]=c(namevector[i,1], namevector[i,2], round(X,3));
187         }
188
189         # Return the table and present it to the user
190         output
191 }
192
193 ## ---------------------------------------------
194 ## End of file
195 ## ---------------------------------------------
```

A small script for calling the ensemble method and use the output to train and evaluate a SVM. Please note that the amount of training examples is too

small in this script and a much larger training set is needed to properly train the classifier. This script serves only to demonstrate how to call the subroutines written for the ensemble method.

```
## ------------------------------------------------
## Demostration of
## Ensemble Entity matching methods
## by Johan Dahlin (2011-08-09)
##
## Input:       - (Is is just a script)
## Output:      A matrix of names and similarity measures
##
## ------------------------------------------------

##-----------------------------
## Preliminaries
##-----------------------------

# Load the relevant packages
library(lsa)
library(stringr)
library(igraph)
library(RecordLinkage)

# Load the subroutines
source("M://fusionstrasket/datafusionsubroutines.r");

# So that the matrix is not reconstructed each time
load("M://abstextmatrix.RData");

# Constrcut the matrix of names to be compared
namevector=matrix(nrow=12,ncol=2);
namevector[1,]=c("Hall DL","Hall D")
namevector[2,]=c("Das S","Das Subrata")
namevector[3,]=c("Seo YW","Seo Y-W")
namevector[4,]=c("Fisher JW III","Fisher JW")
namevector[5,]=c("Grindle C","Girindle C")
namevector[6,]=c("Zhang M","Zhang Miao")
namevector[7,]=c("Xu Q","Xu QF")
namevector[8,]=c("Raprey V","Rapley V")
namevector[9,]=c("Rogova G","Rogova GL")
namevector[10,]=c("Hall DL","Hall CM")
namevector[11,]=c("Das S","Das Subrata")
namevector[12,]=c("Powell G","Powell GM")

# Do the analysis
ff=ensembleentitymatching(namevector,xdata1LSAtext,plotting=F)

## Attempt to learn a SVM to determine if matching or unmatching.
gg=c("M","M","M","M","M","M","M","M","M","U","M","U");
ff2=matrix(nrow=dim(ff)[1],ncol=5)
for (i in 1:dim(ff)[1]) ff2[i,]=as.numeric(ff[i,-c(1,2)]);

# Train the SVM and present the predicted labels for the training set
model=svm(gg ~. ,data=as.data.frame(cbind(ff2,gg)))
summary(model)
```

```
predict(model,subset(as.data.frame(cbind(ff2,gg)),select=-gg))

## Evaluate the SVM

# Construct a small evaluation set
namevector2=matrix(nrow=3,ncol=2);
namevector2[1,]=c("Johansson F","Martenson C")
namevector2[2,]=c("Das S","Das Subrata")
namevector2[3,]=c("Powell G","Powell GM")

# Calculate the similarities for these pairs of authors.
xx=ensembleentitymatching(namevector2,xdata1LSAtext,plotting=FALSE)
xx2=matrix(nrow=3,ncol=5)
for (i in 1:dim(xx)[1]) xx2[i,]=as.numeric(xx[i,-c(1,2)]);

# Predict the labels of the traning set
predict(model,as.data.frame(xx2))

## ------------------------------------------------
## End of file
## ------------------------------------------------
```

C Python-scripts

C.1 Extract author names

This script reads the file generated from data taken from Web of Science and extracts author's full names for the papers that this information exists. The names are extracted by identifying the line that begins with `AF` and the script outputs the data in a text-file with one author name at each line.

```
# ------------------------------------------------
# Extracting full author names
# by Johan Dahlin (2011-06-29)
#
# Input:        The citation information from Web of Science
# Output:       The Full names of authors tab seperated
#
# ------------------------------------------------

import os

def rreplace(s, old, new, occurrence):
        li = s.rsplit(old, occurrence)
        return new.join(li)

# Open files
fin=open("IFochFUSION2002till2010.doc");
fout=open("temp.doc","wt");

# Locate author name rows and clean information
for line in fin:
        if (line.find('AF- ') != -1):
            line=line.replace('AF- ','')
            line=line.replace(",","')
            line=line.replace("","')
            line=line.replace('[Anon]','')
            line=line.replace('; ','\n')
            line=line.replace('|','')
            fout.write(line)
fin.close()
fout.close()

# Change the order of the first and last names
fin=open("temp.doc");
fout=open("temp2.doc","wt");
for line in fin:
        line=line.replace(' ','\t',1)
        fout.write(line)
fin.close()
fout.close()

# Remove duplicate lines
fin=open("temp2.doc")
```

```
fout=open("IFochFUSION2002till2010filtred.doc", 'w')
unique = set(fin.read().split("\n"))
fout.write("".join([line + "\n" for line in unique]))
fout.close()

# ------------------------------------------------
# End of file
# ------------------------------------------------
```

C.2 Extract abstracts

This script reads the file generated from data taken from Web of Science and extracts the full abstract, title of the paper, and name of the authors. The output is found in text-files stored in three directories, names of authors are found in `authors`, titles of papers in `titles`, and full abstracts in `abstracts`. Each filename consists of the directory name and a number, this number is the same for each file, i.e. the paper with the title found in `title100.txt` is written by the authors found in file `authors100.txt` with the abstract as in `abstract100.txt`.

```
# ------------------------------------------------
# Extracting abstracts, authors and titels
# by Johan Dahlin (2011-06-29)
#
# Input:        The citation information from Web of Science
# Output:       Directories of abstracts, authors and titels
#
# ------------------------------------------------

import os, random

# Open file
fin=open("IFochFUSION2002till2010.doc");
mm=0
marker=0
for line in fin:
                # First time a new paper is encountered, extract authors
        if (line.find('AU- ') != -1):
                line=line.replace('AU- ','')
                line=line.replace(';','\t')
                line=line.replace('|','')
                line=line.replace('\n','')
                mm=mm+1
                nameoffile="authors/authors"+str(mm)+".txt"
                open(nameoffile,"wt").write(line)
                # Extract title
        if (line.find('T1- ')!= -1):
                line=line.replace('T1- ','')
                line=line.replace('|','')
                nameoffile="titels/titels"+str(mm)+".txt"
                open(nameoffile,"wt").write(line)
        if (line.find('TI- ')!= -1):
                line=line.replace('TI- ','')
```

```
34                      line=line.replace('|',' ')
35                      nameoffile="titels/titels"+str(mm)+".txt"
36                      open(nameoffile,"wt").write(line)
37                      # Stop at end of abstract
38              if ((line.find('ER- ') != -1) and (marker==1)):
39                      marker=0
40                      fout.close()
41              # Extract abstract line by line
42                      if (marker==1):
43                      line=line.replace('|',' ')
44                      fout.write(line)
45                      # Extract the first line of the abstract
46              if ((line.find('N2- ') != -1) and (random.random()>0)):
47                      line=line.replace('N2- ','')
48                      nameoffile="abstracts/abs"+str(mm)+".txt"
49                      fout=open(nameoffile,"wt")
50                      fout.write(line)
51                      marker=1
52              if ((line.find('AB- ') != -1) and (random.random()>0)):
53                      line=line.replace('AB- ','')
54                      nameoffile="abstracts/abs"+str(mm)+".txt"
55                      fout=open(nameoffile,"wt")
56                      fout.write(line)
57                      marker=1
58  fin.close()
59  fout.close()
60
61  # ------------------------------------------------
62  # End of file
63  # ------------------------------------------------
```