Multiple Entity Reconciliation

LAVINIA ANDREEA SAMOILĂ

Master's Degree Project
Stockholm, Sweden September 2015

KTH Royal Institute of Technology
VionLabs, Stockholm

Master Thesis

Multiple entity reconciliation

Lavinia Andreea Samoilă

Supervisors at VionLabs:
Alden Coots
Chang Gao

Examiner, KTH:
Prof. Mihhail Matskin
Supervisor, KTH:
Prof. Anne Håkansson

Stockholm, September, 2015

Abstract

Living in the age of "Big Data" is both a blessing and a curse. On the one hand, the raw data can be analysed and then used for weather predictions, user recommendations, targeted advertising and more. On the other hand, when data is aggregated from multiple sources, there is no guarantee that each source has stored the data in a standardized or even compatible format to what is required by the application. So there is a need to parse the available data and convert it to the desired form. Here is where the problems start to arise: often the correspondences are not quite so straightforward between data instances that belong to the same domain, but come from different sources. For example, in the film industry, information about movies (cast, characters, ratings etc.) can be found on numerous websites such as IMDb or Rotten Tomatoes. Finding and matching all the data referring to the same movie is a challenge. The aim of this project is to select the most efficient algorithm to correlate movie related information gathered from various websites automatically. We have implemented a flexible application that allows us to make the performance comparison of multiple algorithms based on machine learning techniques. According to our experimental results, a well chosen set of rules is on par with the results from a neural network, these two proving to be the most effective classifiers for records with movie information as content.

Keywords – entity matching, data linkage, data quality, machine learning, text processing

Acknowledgements

I would like to express my deepest gratitude to the VionLabs team, who have supported and advised me throughout the entire work for this thesis, and also for providing the invaluable datasets. I am particularly grateful to professor Mihhail Matskin for his feedback and suggestions. Last but not least, I would like to thank my family and my friends for being there for me, regardless of time and place.

Contents

Acronyms

ACE Automatic Content Extraction. 28

ANN Artificial Neural Network. 20, 25

CoNLL Conference on Natural Language Learning. 28

CRF Conditional Random Field. 9

ECI European Corpus Initiative. 8

ERP Enterprise Resource Planning. 22

HMM Hidden Markov Model. 23

MUC Message Understanding Conferences. 9, 28

NER Named Entity Recognition. vi, 8–10, 28

NLP Natural Language Processing. 10

PCA Principal Component Analysis. 10

RBF Radial Basis Function. 20, 43, 44

RDF Resource Description Framework. 23

RP Random Projection. 10

SVM Support Vector Machine. 8, 9, 19, 20, 25, 32, 33, 35, 41, 44, 46

TF/IDF Term Frequency/ Inverse Document Frequency. 11, 12

List of figures

List of tables

1 Introduction

"The purpose of computing is insight, not numbers."

Richard Hamming

1.1 Background

Entity reconciliation is the process of taking multiple pieces of data, analysing them and identifying those that refer to the same real-world object. An entity can be viewed as a collection of key-value pairs describing an object. In related literature, entity reconciliation is also referred to as data matching, object matching, record linkage, entity resolution, or identity uncertainty. Reconciliation is used for example when we want to match profiles from different social websites to a person, or when multiple databases are merged together and there are inconsistencies in how the data was originally stored. Practical applications and research related to data matching can be found in various domains such as health, crime and fraud detection, online libraries, geocode matching[1] and more.

This thesis is oriented towards the more practical aspects of reconciliation: how to employ reconciliation techniques on domain-specific data. We are going to look at different data matching algorithms and see which ones have the highest accuracy when applied on movie related information.

IMDb[2], the most popular online movie database, contains information about more than 6 million people (actors, producers, directors, etc) and more than 3 million titles, including tv episodes [1]. Information is constantly being added, the number of titles having doubled in the last 4 years. Besides IMDb, there are numerous other sources for movie information online. It is obvious that not all of them have the same content. Their focus can be on basic information such as cast and release year, on user reviews, on financial aspects such as budget and gross revenues, on audio and video analysis results of a movie, or on a combination of the items listed previously.

Among the advantages of reconciliation are the improvement of data quality, the removal of duplicates in a dataset, and the assembly of heterogeneous datasets, thus obtaining information which would otherwise not be available.

1.2 Problem

If we try to gather and organize in one place all the available movie-related information, we will face several distinct difficulties. Besides crawling, parsing and storing issues, one important problem we need to solve is uniquely identifying the movie a piece of data is referring to. Although it may seem trivial at

[1] Matching an address to geographic coordinates.
[2] The Internet Movie Database, http://www.imdb.com

first, when we examine the problem deeper, we see that there are many cases in which the solution is not so obvious.

The main reasons why matching movie information can be hard are the following:

- *incomplete information* - not all data sources have information on all the fields describing a movie or the access is limited e.g. some may not have the release year, or the release year is not available in the publicly released information;
- *language* - data sources can be written in various languages;
- *ambiguity* - not all data sources structure the information in the same way. For example, some may include the release year as part of the title, others may have a specific place-holder for the year;
- *alternate names* - the use of abbreviations or alternate movie titles can be problematic;
- *spelling and grammar mistakes* - movie information is edited by people, which means that it is prone to linguistic errors.

To illustrate these reasons, we will use an example. Assume we have stored in a database a list of all the movies, each movie being described by: IMDb id, title, cast and people in other roles (e.g. director). We have also gathered movie information from SF Anytime[3], which gives us title, cast, release year, genre, and a short plot description. We know that the IMDb ids are unique, but movie titles are not. So the goal is to match the information from *SF Anytime* to the corresponding unique identifier (IMDb id in this case). The catch is that *SF Anytime* is not available in English, so the title may have been modified, and also, the plot description is in Swedish or one of the other Nordic languages. So we cannot completely rely on matching the title: it might be the same, it might not, depends on the movie. We need to find alternative comparisons to identify the correct IMDb id.

1.3 Purpose

This thesis aims to investigate the effectiveness of machine learning reconciliation techniques applied in a movie context. We will decide which algorithm performs better when applied on movie-related data, and examine what domain-specific adjustments are necessary to improve the results.

1.4 Goal

The goal of this thesis is to match movie information gathered from various online sources against the known, annotated data existing in our database. The current reconciliation system at VionLabs has a precision of 60%, so we aim to improve this number. We will implement several machine learning classification algorithms and compare them against a baseline of hand crafted rules. This

[3]Nordic video-on-demand service, http://sfanytime.com

will allow us to decide whether it is worth to change the existing reconciliation system with a more complex one.

We will be working with heterogeneous data sources, each one providing different types of information, or similar types, but in a different format. To manage these data inconsistencies, we will design a functional architecture which will be easily extended to accept different data formats.

1.5 Method

In this thesis, we seek to answer the following research questions:

- How effective are the machine learning techniques when applied to the movie reconciliation problem described previously?
- How to design an extensible reconciliation system that allows various data sources as input?

The research process can be divided into three main phases, which also correspond to the structure of this report: literature study, design and implementation, testing and evaluation.

Because we work with large amounts of data (movie databases, training samples, crawled movie information), we have adopted the quantitative research method [2]. As philosophical assumption, naturally, we have chosen the positivism: we can quantify the variables we are dealing with. Experimental research is our research strategy. Using experiments, we collect data and try to decide which machine learning algorithms are more suitable for movie reconciliation and what can be done to adapt them for better results. To analyse the obtained data, we use techniques borrowed from statistics such as mean and standard deviation. As for quality assurance, we make sure its principles are respected as follows:

- *validity* - we have developed various tests to check the accuracy of our results;
- *reliability* - when running the tests multiple times, we get similar results, confirmed by computing the standard deviation;
- *replicability* - if the same tests are repeated, the same results will be obtained;

1.6 Ethics

An important area where reconciliation has a crucial role is people profiles linking. Whether being in a medical or a social context, the lack of unique identifiers is there for a reason: user privacy and data confidentiality. However, we are dealing with publicly available movie information, thus in our case, there are no concerns regarding privacy.

1.7 Delimitations

This work will not present details on how the data was obtained. We focus on the matching of data instances, not on how they have been acquired.

Both accuracy and fast execution are important concerns, and we try to improve both as much as possible. However, when faced with a choice, we will prioritize accuracy over speed. The parallelization of the application is a possible future development.

Though in the reconciliation domain, both graph-oriented and machine learning oriented approaches have been employed, we will concentrate here only on machine learning techniques since they are more suitable for raw text analysis.

1.8 Outline

The layout of this report is organized as follows: Section 2 provides an overview of entity matching techniques and text feature comparison methods. In section 3, the application architecture and the reconciliation methods we have chosen to compare are described. Section 4 shows the obtained results and their interpretation. Sections 5 and 6 present the conclusions drawn and outline possible future work directions.

2 Theoretical Background

"*Can machines think?...* The new form of the problem can be described in terms of a game which we call *the imitation game.*"

Alan Turing

2.1 Reconciliation

The first mention of record linkage in a computer science research paper was in 1959, by Newcombe et al. [3], back when punch cards were still being used in programming. The term *record linkage* had previously been used to define "the bringing together of two or more separately recorded pieces of information concerning a particular individual or family" [3], so only information pertaining directly to a person. Their paper is part of a research study concerning the dependency between fertility issues and hereditary diseases. As such, they needed a way to automatically connect birth records to marriage records. The difficulties they encountered are similar to what we are facing in movie matching: misspellings, duplicates, inconsistencies in name format, incorrect or altogether missing information. In order to achieve their goal, they are comparing birth records to a filtered set of marriage records, each compared characteristics pair being given a weight in the computed match probability. The marriage records are filtered by the husband's last name and the wife's maiden name to reduce the search space, and the characteristics being compared for each record include name, birthplace, age. They report that using this procedure enabled them to discover 98.3% of all potential $\langle$ birth record, marriage record $\rangle$ pairs.

Later, the record linkage problem has been expressed in mathematical terms by Fellegi and Sunter [4]. They extend the record linkage definition from persons, to objects and events as well. Afterwards, the reconciliation problem has been researched in various contexts: administration - records matching, medical and genetic research, databases, artificial intelligence.

Definition 1. *Given 2 sets of records, S_1 and S_2, by* ***reconciliation*** *we seek to find all correspondences (A, B), with $A \in S_1$, $B \in S_2$, referring to the same real-world object.*

Definition 2. *Each set of records, S_1 and S_2, has a number of* ***characteristics****,*

$$\lambda_1 = \left[\alpha_{11}\alpha_{12}\dots\alpha_{1n}\right], \lambda_2 = \left[\alpha_{21}\alpha_{22}\dots\alpha_{2m}\right]$$

, and for each characteristic, each record has a ***value****, which may be null or not:*

$$\lambda_{1A} = \left[a_{11_A}a_{12_A}\dots a_{1n_A}\right], \lambda_{2B} = \left[a_{21_B}a_{22_B}\dots a_{2m_B}\right], A \in S_1, B \in S_2.$$

Note that the characteristics of the two sets do not have to be identical.

Definition 3. *To compare two records, we define a* ***similarities array****, also referred to as weight vector in other papers,*

$$\phi_{AB} = \left[\theta_{AB_1}\theta_{AB_2}\dots\theta_{AB_p}\right], A \in S_1, B \in S_2.$$

We will call an element of the similarities array a ***feature****. Thus, a feature is the result of a function $f(\lambda)$, which takes 2 or more characteristics as input parameters and returns a numerical value representing the degree of closeness between the given characteristics.*

Example. Table 1 contains two possible records representing two user profiles from two different websites. The results of reconciliation would tell us the probability that the two profiles belong to the same person.

Characteristic	**Value**
Username	john23
Birthdate	10.10.1990
City	Stockholm
Gender	Male
Hobbies	Movies, music

Characteristic	**Value**
Nickname	john
Birth day	10
Birth month	October
Birth year	1990
Gender	Male
Likes	Tennis, movies

Table 1: ***User profiles reconciliation example.***

The similarities array can contain the following features:

- Levenshtein distance [5] between *Username* and *Nickname*;
- Birthday match - the function will have as parameters the *Birthdate*, *Birth day*, *Birth month*, *Birth year* and it will have to manage the two different date formats;
- Gender match;
- Degree of similarity between *Hobbies* and *Likes*.

The *City* characteristic will have to be ignored, since the second record does not contain any information related to location.

Based on the similarities array, and previous knowledge, we apply an algorithm and obtain the probability that the two records represent the same real-world object. In the user profiles case we presented, we should obtain a moderately high probability that the two records are indeed referring to the same person. We will see in the following sections possible models we can use to decide if there is a match or not. Given the computed probability, and the selected threshold, we can decide whether a pair of records represents a match (D_{YES}) or not (D_{NO}).

2.2 Feature extraction

No matter the algorithm we choose to use, we will need methods of quantifying how likely it is that two records, two pieces of structured text, refer to the

same movie instance. This means that we need to find metrics that express how similar certain characteristics are. We will present here the more complex techniques used when processing text information, and in the following sections we will describe exactly how we apply these techniques on our data.

2.2.1 Edit distance

To compare two strings, the most widely used edit distance metric is the Levenshtein distance [5]. It computes the minimum number of modifications needed to transform two strings from one to the other. The allowed modifications are insertions, deletions and substitutions, each of them having a unit cost. If the two strings are completely different, the value of the edit distance will be the length of the longest string. Otherwise, if the strings are identical, the distance will be 0. Using a dynamic programming implementation, the complexity of this algorithm is $O(length\_of\_string1 \times length\_of\_string2)$. Thus, the algorithm is computationally intensive and its use is feasible only on rather short strings.

It is important to know the language of a text in order to know whether the results from certain comparison methods are relevant or not. For example, it is not useful to know the edit distance between two titles, one written in English, the other in Swedish. Whether they represent the same movie or not, they will be different anyway, so this score should not have a large weight in the final decision.

2.2.2 Language identification

A simple way of identifying the language a text is written in is using *stop words*. In each language, there are common words like prepositions and conjunctions which can have no meaning by themselves, but are used in all phrases as connectors. If no list of stop words is available, there are automatic methods to generate it from documents written in the required language. For instance, one such method would be to take the available documents, tokenize them into words and count each individual word. To get the stop words, the complete word list is filtered by word length and frequency. However, this solution does not take into account spelling mistakes.

Language identification can be considered a subclass of a larger problem: text categorization. Cavnar and Trenkle [6] tackle this text categorization problem and propose an *N-gram-based approach*. They compute the N-gram frequency profile for each language and then compare this profile to that of the target text - the text for which we want to recognize the language. N-grams are contiguous word substrings of N characters or start/end of word markers, and these character associations are language-specific. They tested the system on more than 3400 articles extracted from Usenet[4]. The results varied with how many N-grams are available in the profile of each language, and stabilized after around 400, with 99.8% accuracy. Anomalies occurred when the target text contained words from more than one language. The advantage of this technique lies in its speed and robustness.

[4]Usenet is a communications network, functional since before the World Wide Web. http://www.usenet.org/

Grefenstette [7] has used the ECI corpus data[5] to compare the stop words technique against the tri-gram method. According to the results, the longer the target text, the better the accuracy is in both cases. This makes sense because longer text increases the chances of recognizing both tri-grams and stop words, thus leading to a higher confidence when deciding the language. For one to three word sentences, the tri-gram method outperforms the stop words method, since it is less likely that shorter sentences contain stop words. For sentences with more than six words, the accuracy in recognizing the correct language is over 95% in both cases.

Computing and comparing the N-gram profiles for large texts takes time and considerable processing power. To minimize the feature space, Teytaud and Jalam [8] have suggested the use of SVMs with a radial kernel for the language identification problem.

The previous methods consider that the language uses space-separated words. However, that is not always true, like in the case of Chinese and Thai. To accommodate this type of languages, Baldwin and Lui [9] use byte N-grams and codepoint N-grams, thus ignoring any spacing. In their experiments, they compare three techniques: *nearest neighbour* - with cosine similarity, skew divergence and term frequency as distance measures, *naive Bayes* and *SVMs* with linear kernels. The conclusion is that SVMs and nearest-neighbour with cosine similarity perform best. However, when there is a mix of languages in a document, the results deteriorate.

2.2.3 Proper nouns extraction

Nouns that are used to identify a particular location or person are called proper nouns, like *John* and *Nevada*. In scientific papers, distinguishing proper nouns from other words is referred to as *named entity recognition* (NER). At first, researchers oriented towards rule-based approaches, like Rau [10] who used prepositions, conjunctions and letter cases to identify the company names in a series of financial news articles. However, rule-based systems have the disadvantage of not being portable across domains, and also, maintaining the rules up-to-date can be expensive.

Nadeau and Sekine [11] divide the possible features into three categories:

- word-level features - word case, punctuation, length, character types, suffixes and prefixes;
- list lookup features - general dictionary, common words in organisation names (e.g. *Inc*), abbreviations and so on;
- document and corpus features - repeated occurrences, tags and labels, page headers, position in sentence.

The above mentioned features can be considered as differentiating criteria in a rule-based system or, as has recently been the case, they can be used as input parameters in machine learning systems, summarized in Figure 1.

[5]European Corpus Initiative Multilingual Corpus, http://www.elsnet.org/eci.html

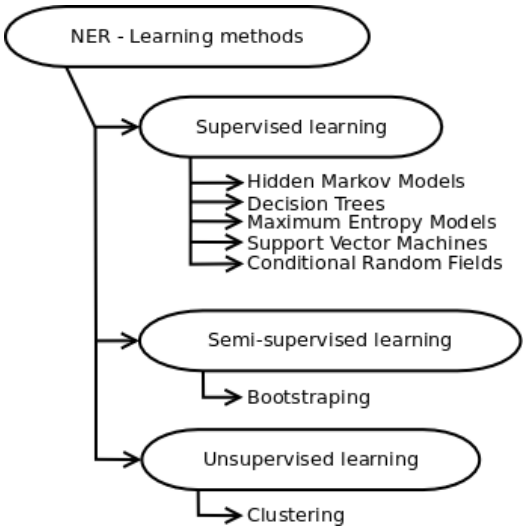

Figure 1: ***NER learning techniques.***

Bikel et al. [12] apply a variant of the hidden Markov models on the MUC-6[6] dataset and obtain over 90% precision for English and Spanish. Sekine [13] labels named entities from Japanese texts using C4.5 decision trees, dictionaries and a part-of-speech tagger. More recently, but for the same problem, Asahara and Matsumoto [14] have chosen to use a SVM-based classifier, with character-level features as input (position of character within the word, character type, part-of-speech obtained from a morphological analyser), thus correctly identifying more than three quarters of the named entities present in the test documents. Conditional random fields (CRF) are a generalized form of hidden Markov models, allowing powerful feature functions which can refer not only to previously observed words, but to any input word combination. Combining CRFs with a vast lexicon obtained by scouring the Web, McCallum and Li [15] have obtained decent results on documents written in English and German.

Even though supervised learning dominates the research done in this field, the disadvantage is that it requires a labelled training set which is often not available, or not large enough. In such cases, semi-supervised and unsupervised approaches have been investigated, where extrapolation and clustering are the main techniques.

At Google, Pasca et al. [16] attempt to generalize the extraction of factual knowledge using patterns. The tests are executed on 300 million Web pages written in English, which have been stripped from HTML tags and then split into sentences. In their experiments, they start from 10 seed items in the form *(Bob Dylan, 1941)*, representing the year a person is born in. From the sentences which contain one of the input facts, basic patterns are extracted in the form *(Prefix, Infix, Postfix)*. Using pre-computed word classes (e.g. names of months, jobs, nationalities), the patterns are modified into more general ones and duplicates are removed. Using the obtained patterns, more candidate seeds are extracted and the top results are added to the initial seed set for a new iteration. With this method, more than 90% of the 1 million extracted facts have the full name and the accurate year.

A problem is establishing the boundaries of named entities, especially difficult in the case of book and movie titles which do not contain proper names, like

[6]The 6^{th} Message Understanding Conference, http://cs.nyu.edu/faculty/grishman/muc6.html

The Good, The Bad and The Ugly. Downey et al. [17] employ an unsupervised learning method: they assume that all contiguous sequences of capitalized words are named entities if they appear in the same formation often enough in the document. The method requires a large corpus for the results to be relevant, but the data does not have to be annotated, so this condition can be easily fulfilled.

Instead of using only one classifier, Saha and Ekbal [18] leverage between the decisions from multiple NER systems mentioned above, so as to achieve a better precision. The classifier ensemble was tested on documents in Bengali, Hindi and Telugu and according to the results, associating weights to each classifier is more effective than a voting scheme.

2.2.4 Feature selection

There are many text comparison techniques encountered in the research done in various NLP sub-fields. It is important to note that, the more features the decision algorithm has to consider, the more computation power is needed. Also, it may be the case that the algorithm requires independent features, like the naive Bayes classifier, and with a high number of features, it is unlikely that there are no dependencies among them. So, having a large number of similarity metrics means that we need to consider dimensionality reduction either by selecting the most relevant, or finding a way to combine several features into one without information loss. We define $F_{k\times N}$ as the **input dataset**, where k is the number of features, and N is the number of samples. The purpose of feature reduction is to reduce k, even N if possible.

Usually in text classification tasks, the features are the words composing the documents, thus the feature space can reach a high dimensionality, a high k. In such cases, there are several techniques that can be used, as described by Taşcı and Güngör [19]. These techniques are based on statistics such as information gain, document frequency, chi-square[7]. Dasgupta et al. [20] propose a different algorithm which assigns weights to features, depending on the results obtained when the classification algorithm is run with various smaller sets of features.

Principal component analysis (PCA) [21, chap. 1, 4] is another way to reduce dimensionality while preserving as much as possible of the original information, with focus on the variance. The process involves computing the covariance matrix of the data, identifying the eigenvalues and the corresponding eigenvectors, which are then orthogonalized and normalized. The vectors obtained are the so-called *principal components* of the dataset, they can be considered a summary of the input dataset. A method less computationally expensive is random projection (RP) [22], which involves generating a random matrix $R_{d\times k}, d \ll k$ and multiplying it with $F_{k\times N}$, thus lowering the number of features while maintaining the distance between the samples.

[7]Independence test between two random variables.

Choosing the right features has a significant influence on how a learning algorithm works. Having too many features, or irrelevant ones, can lead not only to an increased complexity, but also to a tendency to overfit. Overfitting occurs when the learning algorithm attempts to match the training data as close as possible, which results in a poor generalisation of the data and so, to a poor performance on the test data. It is also important that the training data has a balanced number of samples for each feature in order to minimize the risk of overfitting.

2.3 Blocking techniques

Given 2 sets of records, S_1 and S_2, there are $|S_1| * |S_2|$ possible record pairs to evaluate. Or if the task is to remove the duplicates from S_1, then there are $\frac{|S_1|}{2} * (|S_1| - 1)$ possible pairs. For each of these pairs, the similarities array has to be computed. Regardless of the decision model used, this computation is the most time and resource consuming from the entire reconciliation process, especially problematic when dealing with large record sets. That is why usually a filter method is used to reduce the number of pairs that need to be evaluated.

Standard blocking is an intuitive method where the filtering is done by certain key fields. For instance, in personal profile records, key fields could be considered the first and last name. The records that have exactly the same value, or a value within an accepted error boundary, are selected. These key fields should be chosen so that the resulting selection is not too small that is does not contain the matches, nor too large that the evaluation of candidate pairs still takes too long. *Sorted neighbourhood* is a variant of standard blocking, except that the records are first sorted by the key fields, and then from among w consecutive records, candidate pairs are generated. This *window* is then moved to the next records in the list and the process is repeated. However, true matches may be skipped if more than w records match a key field value, so this parameter should be chosen carefully.

Bi-gram indexing, or more general, *k-gram indexing*, is another blocking technique more suitable for phrases, especially when the phrases contain common words, and for incomplete or incorrectly-spelled queries. The query terms are split into a list of k-grams[8]. A number of permutations are generated from these k-grams, and then inserted into an inverted index. A record containing the initial query term will be added to all the keys of the inverted index. Taking the records in this index two by two, we obtain our list of candidate pairs. With this method, more candidate pairs will be evaluated than in standard blocking. *Canopy clustering with TF/IDF* splits the entire dataset by taking random *seed* records and grouping them with their nearest neighbours in terms of their TF/IDF[9] distance metric. Once a record has been included into a group, it will no longer participate in the selection process.

[8] K-gram - a series of k consecutive characters in a word or phrase.

[9] TF/IDF - a numerical measure indicating the relevance of a word in a document relative to the collection of documents to which it belongs. A high word frequency in a document is balanced out by the overall collection frequency, thus giving an informed estimate of the word's significance.

Baxter et al. [23] have studied the effects in results quality of the blocking techniques we have presented previously. They compare the classical methods (standard blocking and sorted neighbourhood), to the newer ones: bi-gram indexing and TF/IDF clustering. They have used as dataset a mailing list created automatically using specialized software, ensuring that each email instance has at least one duplicate. This allows them to accurately measure the performance of each algorithm in terms of precision and computation reduction. According to their experiments, with certain parameter adjustments, bi-gram indexing and TF/IDF indexing perform better than the classical methods. It remains to be seen whether the same improvements occur with a real-world dataset as well.

McCallum et al. [24] study the clustering of large data sets with the practical application tested on a set bibliographic references from computer science papers. Their goal is to identify the citations referring to the same paper. The approach they have chosen has points in common with the TF/IDF clustering, but it is slightly more complex, requiring two steps:

- first, the dataset is divided into overlapping groups, also called *canopies*, according to a computationally cheap distance measure. Note that here the groups can have elements in common, and the distance measure is computed between all possible data point pairs, unlike in the Baxter canopy method;
- secondly, a traditional clustering approach (e.g. k-means), with a more expensive distance metric (e.g. string edit distance) is used to identify the duplicates within a canopy. The computational effort is reduced because no comparisons are made between elements belonging to completely different canopies.

To create the canopies, two distance thresholds are used, T_1, T_2 with $T_1 > T_2$. A point X from the dataset is selected, the distance to all the others is computed, and the first canopy contains the points within T_1 radius from X. The closest points to X, those within T_2 radius, are removed from the dataset. The process is repeated until the dataset is empty. This model can be adapted to be employed as a hierarchical filter, if we adjust the second step and just use it as a modality to further split and decrease the search space.

2.4 Reconciliation approaches

2.4.1 Rule-based approaches

If all the data was clearly and uniquely labelled, reconciliation would only be a matter of matching equivalent identifiers and merging the information. For instance, the *Jinni*[10] movie descriptions can be easily matched to the corresponding IMDb page since they contain the IMDb id. However, this is just a happy coincidence. Often there is missing or incorrect information and more complex techniques are required to find the right correspondences.

Deterministic linking is a straightforward method, also referred to as rule-based linking. It performs best when the sets to be reconciled have characteristics in common that can be used to compare two records. A pair is considered

[10] Jinni - a movie recommendations engine, http://www.jinni.com

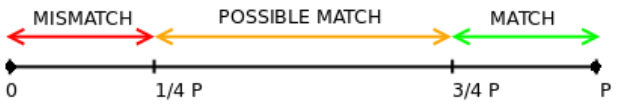

Figure 2: ***Example of linking thresholds.*** *Assuming a number of P comparison methods which return a value in the $[0, 1]$ interval and a unitary weight vector, pair records with a similarities array sum lower than $\frac{P}{4}$ are classified as a non-match. If the sum is higher than $\frac{3P}{4}$, then the pair is considered a match. Otherwise, it cannot be said whether the record pair is a match or not. Image adapted from [25].*

a match if all or a certain subset of characteristics correspond. With this type of algorithm, the result is dichotomous: either there is a match or not.

In the case of probabilistic linking, weights are associated to the elements of the similarities array. Each element will have a different degree of influence on the final output. The values in the similarities array are proportioned to the assigned weight, summed and then, according to a threshold, the pair is classified as a match or as a mismatch. If two thresholds are used, then another classification is possible: possible match, as we can see in the example in Figure 2. With probabilistic linking, we obtain not just a classification, but the likelihood of that classification, which allows adjusting the thresholds according to what is the main interest of the user: increasing the amount of true matches identified or decreasing the amount of false links.

The weights in probabilistic linking are computed as a function of the actual data, using either a simple three-step iterative process: *apply* the rules, *evaluate* the results, *refine* the weights and/or thresholds, or more complex methods based on statistic models like maximum likelihood estimation. As opposed to that, in deterministic linking, the discriminative criteria have been pre-determined, based on domain knowledge, not on the available data.

The Link King[11] is a public domain software used for linking and removing duplicates from administrative datasets. It provides the users with the option to choose which algorithm fits them best. With deterministic linking, the users define which combination of fields must match, and whether partial matches are acceptable or not. The probabilistic algorithm relies on research done by Whalen et al. [26] for the Substance Abuse and Mental Health Services Administration's (SAMHSA) Integrated Database Project. In their study, the prior probabilities needed to compute the weights are calculated on a dataset obtained using deterministic rules. Also, to each field value, a scale factor is associated to express the impact of agreement on a certain variable e.g. name agreement on 'Smith' is not as important as name agreement on 'Henderson'.

These two linking approaches, deterministic and probabilistic, have been compared on a dataset containing patient information, so fields such as date of birth, postcode, gender and hospital code [27]. Using the number of false matches and

[11]The Link King, http://www.the-link-king.com/

false mismatches as an evaluation criteria, the authors show that in datasets with noticeable error rates (amount of typing errors), the probabilistic approach clearly outperforms the deterministic one. With very small error rates in the dataset, both techniques have similar results. The deterministic method simply does not have enough discriminating power, while using the probabilistic method, the weights of each variable are fitted according to the available dataset. The probabilistic strategy works better, but there is a price to pay: higher complexity, thus longer computation times.

In account reconciliation, Chew and Robinson [28] use a more complex approach, which combines deterministic and probabilistic techniques, in order to match receipts to disbursements. These records consist of date, description and amount, so the key information in the description must be identified in order to find the correct pairs. First, the terms from each description are ranked according to their *distinctiveness* using the PMI measure[12]. Then, the Pearson product-moment correlation coefficient[13] is computed on the term PMI vectors for each transaction pair and if the value is high enough, the pair is declared a match. For their particular set of data containing tens of thousands of transactions, they obtain near perfect precision and recall. However, it should be noted that one-to-many and many-to-many matches have not been dealt with in this paper.

As we have seen, the rules-based approach is relatively simple and easy to understand. Moreover, there is no need for labelled data, since no training is required. However, it has several disadvantages: the rules can become exceedingly complex and thus difficult to maintain. Also, the rules are specific to one dataset and cannot be migrated to another application.

2.4.2 Graph-oriented approaches

Movies are connected to each other through cast, crew, sometimes even characters in the case of prequels/sequels. We could organize the available information in the form of a graph and apply graph-related algorithms in order to identify which nodes we can join together, thus making use of the connections between the entities, rather than just characteristic similarities.

This type of approach has been used to identify aliases belonging to the same person by analysing the content posted by those aliases. Modelling social networks and trying to map the nodes to each other across multiple networks is similar to the graph isomorphism problem. Unfortunately, no efficient algorithm has been found yet for this problem for all types of graphs [29]. However, in social graphs, we do not expect an exact mapping and some of the work has already been done by the users who link their accounts.

Graph algorithms are based on the topology of the network, like the one proposed by Narayanan and Shmatikov [30]: the algorithm starts from *seed*

[12]Pointwise Mutual Information - indicates how likely is a term to occur in a document.

[13]Pearson product-moment correlation coefficient - index used in statistics as a measure of the linear dependence between two variables.

nodes, which are known in both the anonymous target graph and in the graph containing the auxiliary information, and maps them accordingly. The next step is the *propagation* phase, in which the mapping obtained previously is extended by relying on the topology of the network. The *propagation* is an incremental process which takes place until the entire graph has been explored.

Yartseva and Grossglauser [31] suggest a different approach based on percolation theory, the study of clusters in random graphs. It works by matching two nodes if those nodes already have a certain number of neighbours already matched (controlled by a parameter r). Besides the algorithm itself, they also show how to compute the optimal size for the initial set based on the network parameters and r. The algorithm works like this: for each vertex, a count is kept. Each round a pair we know is correctly matched is chosen, and the count for each of the neighbours is increased. Then a vertex whose count is bigger than r is chosen and added to the list of matched pairs. This process is continued until there are no more changes.

Korula and Lattanzi's algorithm [32] combines both techniques. Initially, according to the available information, nodes are linked with certain probabilities. Like in percolation graph matching, for each vertex, a count of the matched neighbours is kept. Each round, the two nodes with the highest count in each of the graphs are matched. To improve the error rate, nodes with higher degree are preferred. More specifically, each round, only nodes with $degree > \frac{D}{2^i}$ can be matched, where D is highest degree in the graph, and i the round number. So the degrees of the nodes allowed to be matched decrease as the algorithm progresses.

2.4.3 Machine learning oriented approaches

Machine learning oriented approaches require training data. However, it is possible that training data is not available or is in insufficient quantity and difficult to produce. Christen [33][25] proposes a way to automatically generate training data, without human supervision. The algorithm relies on the assumption that two records that have only high similarity values refer to the same entity, and the opposite in the case of low similarity values. So these two types of records are selected, then used as training set for a supervised algorithm so that the most likely matches computed can be appended to the training set. The selection can be made using thresholds, but better results were obtained with a nearest-based criterion: only the weight vectors closest in terms of Manhattan distance[14] to the ideal match, respectively mismatch vector are chosen. In their experiments, this approach outperforms using an unsupervised k-means algorithm.

In machine learning, the first and simplest algorithms taught are variations on *Nearest Neighbour*, like *k-NN*. This is what researchers call a *lazy learner*: it stores in memory the entire training dataset, and every time a query to classify an instance is run, it iterates through the entire dataset to reach a decision.

[14]Manhattan distance - also known as taxicab metric, it is the sum of absolute differences between two vectors.

On the other hand, *eager learner* algorithms try to fit the dataset to a model, thus not being necessary to retain in memory the initial dataset. Also, the classification process is much faster since it is not required to evaluate each entry in the training set for every query. Examples of eager learner approaches include naive Bayes, neural networks and decision trees.

Naive Bayes

A model widely used in text classification is naive Bayes [44, Ch. 4]. In all Bayes-derived methods, the following theorem is crucial:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

, which shows how to compute the *posterior* probability of an event A, knowing that event B took place, in terms of the *likelihood* of event B given event A, the *prior* probability of event A and the *evidence* of event B.

The theorem can be generalized, since the evidence can refer to several features, not just one:

$$P(A|B_1, B_2, ..., B_n) = \frac{P(B_1, B_2, ..., B_n|A)P(A)}{P(B_1, B_2, ..., B_n)}$$

Naive Bayes is a probabilistic model that makes the assumption that these features are independent. This assumption is made in order to simplify the computation of the denominator, bringing it to the form:

$$P(A|B_1, B_2, ..., B_n) = \frac{P(A)\prod_{i=1}^{n} P(B_i|A)}{P(B_1, B_2, ..., B_n)}$$

The event priors and the likelihoods for each feature are computed based on the training set, generally assuming a Gaussian distribution of the samples.

$$P(B_i|A = a) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left(-\frac{(B_i - \mu_a)^2}{2\sigma_a^2}\right)$$

, where μ_a - the mean of the samples belonging to event a, and σ_a - the standard deviation.

To find which event A has occurred, given the feature vector B, maximum likelihood estimates are used. The probabilities for every possible event A are computed, and the largest one is chosen:

$$\arg\max_a P(A = a) \prod_{i=1}^{n} P(B_i|A = a)$$

The advantages of naive Bayes lie in its simplicity and its scalability to large number of features. The algorithm is known for being a decent classifier, especially in spam filtering, but the computed probabilities do not necessarily reflect the reality [34]. Also, the independence assumption is often not quite accurate. Moreover, a Gaussian distribution may not reflect how the data looks like, which in turn may lead to errors in parameter estimation.

In the legal domain, Dozier et al. [35] have applied the Bayes theory in order to create a database of expert witnesses profiles by gathering data from several sources. To merge the duplicates, 6 pieces of evidence are used to asses the match probability between two profiles. These pieces of evidence are the results of the comparison between the various profile fields, such as name, location, area of expertise. In this scenario, the variables obtained are considered, and are indeed, independent. The comparison yields only five possible results: exact, strong fuzzy, weak fuzzy, mismatch, unknown - for missing information. The probability that two profiles are a match given the evidence is computed according to the Bayes equation. The prior probabilities are given values based on estimates from experience and observation. To obtain the conditional probabilities, a manually created subset of profiles and the corresponding pairs has been created.

Decision Trees

In a decision tree [44, Ch. 6], each internal node represents a decision point, and each leaf represents a class or an event or a final decision. Given the input, the tree is traversed starting from the root. At each internal node, a method evaluates the input and decides which branch will be followed next. The method can be a simple comparison of an input field with a fixed value, or it can be a more complex function, not necessarily linear, taking several parameters into account. If a leaf node has been reached, then the input sample that reached the node is associated its label.

The trees can be created manually, method very close to the rule estimation, since a path from root to leaf represents a rule. Or they can be created automatically, based on a training set. To be noted that not all attributes must necessarily take part in the decision process, some may be considered irrelevant by the tree creation algorithm. To create a tree, the training set is evaluated and the feature considered to have the most discriminative power is chosen for the decision at root level. Then the set is split, and the process is repeated for each of the new nodes until there are no more splits possible, meaning all samples in a subset belong to the same class. This is known as the ID3 algorithm [36], but it has some limitations which are addressed by the C4.5 algorithm [37]:

- ID3 deals only with discrete attributes. C4.5 can process continuous attributes using thresholds to create value intervals;
- C4.5 accepts data with missing attribute values - if the sample set has no values for the attribute checked by an internal node, that node becomes a leaf with the label of the most frequent class;
- ID3 does not guarantee the creation of the smallest possible tree. Thus, after the tree has been created, C4.5 prunes it by removing redundant branches and turns them into leafs.

If the created tree does not classify the data well enough, a subset of the samples which have been misclassified can be added to the training set and the process is repeated until we obtain a satisfactory tree.

To determine which attribute to use as set division criterion, we need to sort the attributes by their power to distinguish samples. In order to do that, first we quantify the impurity or the entropy of the data. The entropy of a dataset S is a numerical indicator of the diversity of S and is defined mathematically:

$$Entropy(S) = -\sum_i p_i \log_2 p_i$$

, where p_i is the proportion of samples belonging to class i. In a binary classification problem, an entropy of 1 means that the samples are equally divided between the two classes. When the set is partitioned by the attribute, we expect a reduction in the impurity of the subsets. Logically, we choose the attribute A with the highest such reduction, also referred to as information gain:

$$Gain(S, A) = Entropy(S) - \sum_{k \in values(A)} \frac{|S_k|}{|S|} Entropy(S_k)$$

, where S_k is the subset of samples with value k for attribute A. Another set diversity indicator that can be used is the Gini impurity [38]:

$$Gini(S) = \sum_i p_i(1 - p_i)$$

With Gini splits, the goal is to have nodes with samples belonging to the same class, while entropy attempts to divide the sample set equally between the child nodes.

The worst case scenario is when each sample reaches a different leaf, meaning that there as many leaves as there are samples. This is a clear example of overfitting. If it is not controlled, the tree can become extremely large, leading to a deterioration of the results. Possible solutions are tree pruning and limiting how much the tree can grow. For pruning, each internal node in turn is considered a leaf (with the dominant class as label) and if the new tree is just as reliable as the initial one on a validation set, then the node remains a leaf. However, this is a costly operation since it requires the evaluation of multiple trees.

Decision trees have proven to be effective as part of a system designed to identify record duplicates in a dataset containing student and university staff information [39]. The records hold information such as first and last name, home address and telephone number. The authors of the paper have used an error and duplicate generating software to render the dataset unreliable. To identify the duplicates, a 4-step process is proposed:

- clean the data and convert it to a standard format;
- sort the records according to a pre-determined attribute. Domain specific information is required here. Using a sliding window, generate pairs of match candidate record pairs;
- apply on the candidate pairs a clustering algorithm to split the pairs into matches and mismatches;
- use the previously obtained labelled data as training set to create a decision tree, then prune it if necessary.

By combining clustering and classification procedures, they have identified 98% of the duplicates.

SVMs

A Support Vector Machine (SVM) [40] is a supervised binary classification technique. Its origins lie with the *maximal margin classifier*, which improved, became the *support vector classifier*, which was further extended to the *support vector machine*.

A hyperplane is a p dimensional subspace splitting a space one dimension higher. The hyperplane is defined as:

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0$$

. It looks quite similar to the probabilistic approach if we consider the β parameters as the weights, and X as the feature vector. If we find those β parameters that can split the feature set into those samples for which the equation above is positive, and negative for the rest, then we can say we have trained the classifier. Depending on the data, there can be multiple such parameter choices. A *maximal margin classifier* looks for the plane which maximizes the smallest distance from the training samples to it. The closest samples are considered the *support vectors*, since they determine the hyperplane, as we can see in Figure 3. Should they be modified, the hyperplane would also change.

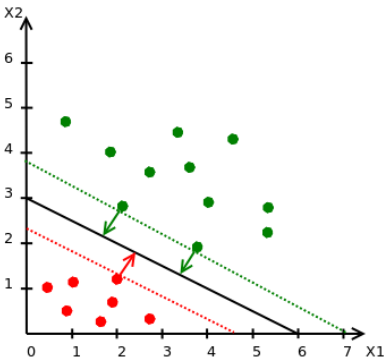

Figure 3: ***Maximal margin hyperplane.*** *Here is an example of a 2D space split by a line. The green points can be separated from the red ones using the hyperplane represented by the black solid line. The margin is the distance from the plane to the dashed lines which go through the points closest to the hyperplane. Image adapted from [40].*

The maximal margin classifier only works if the samples of the two classes are linearly separable. If the samples are interlocked, then no separating hyperplane will be found. Also, the presence of outlier samples can have a strong influence on the hyperplane, which is not desirable. The *support vector classifier* resolves these issues by introducing the concept of *soft margin*. Its goal is still finding the largest margin, but it accepts a certain amount of samples to be misclassified.

If the classes do not respect linear boundaries, even the support vector classifier will fail. To handle these cases, SVMs have been developed. SVMs allow enlarging the feature space and make the computation feasible using functions referred to as *kernels*. The kernels are an expression of the similarity between two vectors and require the inner product between all pairs of vectors . The linear kernel, $K(v_1, v_2) = 1 + v_1^t v_2$ is equivalent to using the support vector classifier. Other kernels allow the creation of boundaries of different shapes. Among the most frequently used kernels, we mention:

- the *polynomial* kernel: $K(v_1, v_2) = (v_1^t v_2 + 1)^d$, where d - the degree of the polynomial;
- the *radial basis function* kernel (RBF): $K(v_1, v_2) = \exp\left(-\frac{\langle v_1, v_2\rangle^2}{2\sigma^2}\right)$, where σ - tuning parameter, $\langle v_1, v_2\rangle$ - the Euclidean distance between two vectors;
- the *sigmoid* kernel: $K(v_1, v_2) = \tanh(\gamma\langle v_1, v_2\rangle + r)$, where γ and r - tuning parameters.

SVMs have been employed in a wide range of applications, including duplicate detection. Su et al. [41] tackle the problem of removing duplicate records from query results over multiple Web databases. Since the records from one source typically have the same format, first duplicates from one set are removed. Then, this set is used for the training of two classifiers, a SVM with a linear kernel and a probabilistic linker. The results obtained when applying the trained classifiers on the data are added to the training set and the process is repeated until no more duplicates are found. They have chosen the SVM because it works well with a limited amount of training samples and it is not sensitive to the positive/negative samples ratio. The advantage of the proposed technique is that it does not require manually verified labelled data. The authors report a performance similar to the one of other supervised techniques when applied on datasets with information about paper citations, books, hotels and movies.

Another example of SVM usage is the work of Bilenko and Mooney [42], who go beyond the standard string edit distance and cosine similarity. Depending on the domain, the relevancy of the string similarities can be improved by taking into account sensible domain characteristics. For example, the noun "street" is not as important when comparing addresses as it is when comparing people names. So, to escape the deficiencies of the standard comparison methods, they train two classifiers, one using expectation-maximization for shorter strings based on character-level features, and an SVM for longer strings with the vector-space model of the text as input.

Neural Networks

Even though it is still not entirely clear how the human brain works, what we do know is that the electrical signals travel along paths formed between neurons, and depending on the experiences of each person, some paths are more travelled than others, some things are more familiar to us than others. Attempts to emulate this process have lead to the emergence of artificial neural networks, ANNs [43]. They are considered simplified versions of the brain, partly due to a lack of understanding of the brain, partly due to a lack of computation power.

A neural network consists of simple processing elements called *neurons*, connected to each other through *links* characterized by *weights*. The neurons work in a parallel and distributed fashion. They learn by automatically adjusting the weights. An ANN is characterized by:

- learning method - *supervised*, *unsupervised*, *reinforcement*;

- direction of information flow:
  - *feed-forward networks* - information flows one way only;
  - *feedback networks* - information flows both ways.
- weight update methods - usually a form of gradient descent with error backpropagation;
- activation function - most common being the step and sigmoid functions;
- number of layers and neurons per layer.

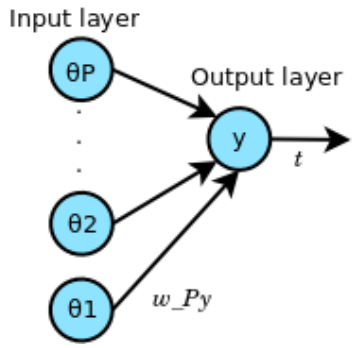

Figure 4: ***Basic neural network model.*** *The neuron shown here has a bias value of t and P incoming connections, associated to the w_{Py} weights. Image adapted from [44, Ch. 7.3].*

In Figure 4 we have represented the basic form of the neural network, also known as a perceptron, which takes P inputs and returns one value as output, y. The output is computed according to the following equation:

$$y(\theta) = f\left(\sum_{i=1}^{P} w_{Py_i}\theta_i + t\right)$$

, where t is the bias value, and f is the activation function. Usually, for a more concise expression, the bias is included in the sum by adding an extra input, $\theta_0 = 1$ and prepending the bias to the weight vector:

$$y(\theta) = f\left(\sum_{i=0}^{P} w_{Py_i}\theta_i\right)$$

.

Similarly, for the network in Figure 5, the overall network function will look like this:

$$y(\theta) = f_1\left(\sum_{j=0}^{M} w_{yM_j} f_2\left(\sum_{i=0}^{P} w_{MP_i}\theta_i\right)\right)$$

, with f_1, f_2 - activation functions.

The computation done by a perceptron can be seen as a linear separation of the input space, but also as a search in the weight vector space. During learning, the weights are updated each iteration for each sample s according to the following gradient descent rule:

$$w_{Py_i}(k+1) = w_{Py_i}(k) + \eta(d_k - y^s)\theta_i^s, i = 0..P$$

, where k is the iteration number, η the learning rate, y^s the correct output from sample s, and d_k the output from the previous iteration. So the weights are adjusted proportionally to the output error, the learning rate determining the influence degree of the error. It should be noted that there are more efficient methods than the gradient descent like conjugate gradients and quasi-Newton methods, as shown by Nocedal and Wright [45].

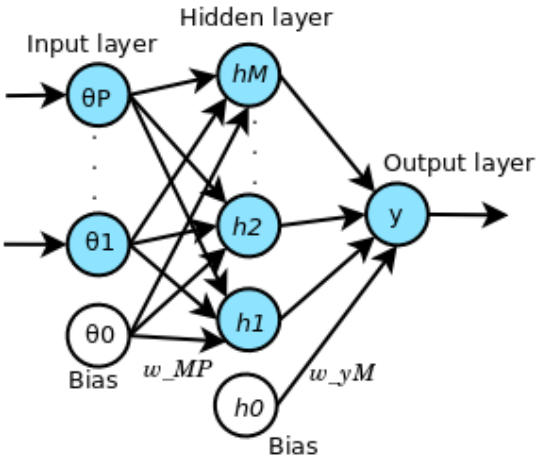

Figure 5: ***Neural network with one hidden layer.*** *The feed-forward network in this example consists of P input features, 1 hidden layer with M neurons and 1 output value. The θ_0 and h_0 neurons are used as threshold equivalents. w_{MP} and w_{yM} represent the vectors of computed weights. Image adapted from [43].*

Neural networks have been applied to the problem of identifying the fields which represent the same information across several different databases (attribute correspondences) [46]. As input, schema information (column names, data types, constraints) and statistics obtained from the database content are used. All this information needs to be converted to a numerical format. There are transformations depending on the type of fields. For example, binary values are converted to 0/1, numerical values are normalized to the [0..1] range using a sigmoid function (not a linear transformation in order to avoid false matches and false drops), and for string fields, statistics such as field length are used. The reconciliation algorithm proposed follows these steps:

- parse information obtained from database;
- apply classifier to decide which fields in a database represent the same information, so that when the neural network is applied, a field in database X is not mapped to two fields in database Y. Moreover, it leads to lower running time and reduced complexity for the neural network;
- use a neural network to learn which fields across multiple databases represent the same information.

Another example of neural networks in reconciliation problems is their application in enterprise resource planning (ERP) to find duplicate records [47], occurrences due to the lack of unique identifiers. Instead of relying on string comparisons to determine record similarities, the authors use a semantic approach instead. Vectors containing discriminating keywords are deduced from the records, and then a neural network establishes the similarity degree. They show superior results to other methods based on computing string edit distances differences.

2.5 Movie-context related work

For ontologies reconciliation, Huang et al. [48] propose an algorithm which leverages techniques from the rule-based and learning-based approaches. Both these approaches have their own disadvantages: rule-based algorithms are fast, but the rules are set in stone. They do not adapt to changes in data, and more than that, the rules do not always take into consideration all the relevant discriminative factors. Learning-based algorithms are slower and may need large quantities of learning data. An ontology is considered to have the following components: name, properties and relationships. These aspects will form the discriminant criteria for the proposed algorithm, Superconcept Formation System (SFS). SFS uses a three neuron network, one neuron for each of the three component of an ontology concept. Initial weights are assigned to each neuron and then gradient descent is used to update the rules, based on the training samples. The input data for the neurons represents the results from similarity functions for each ontology concept aspect. To evaluate their algorithm, they take two real world ontologies and compare their result with that of human experts and report to have identified 85% correct predictions.

LinkedMDB[15] is a project aiming to connect several databases with information pertaining to the film universe. It uses several web pages like *Freebase* and *IMDb*, as sources for movie information, information which is enriched by linking it to other pages such as *DBpedia/YAGO* for persons, *MusicBrainz* for soundtracks, *Flickr* for movie posters. The links are automatically generated using string matching algorithms: Jaccard, Levenshtein, HMM and not only. The tuples obtained are stored using the RDF model for clarity and ease of comprehension. Hassanzadeh and Consens [49] report the weighted Jaccard similarity[16] as the edit distance to provide the highest number of correct links. Choosing a threshold to determine when the similarity score is high enough and the link is deemed as valid, is a compromise between the amount of discovered links and the how many of them are correct. However, it must be noted that because of data copyright issues and crawling limitations, only 40k movie entities are processed and 10k links to *DBpedia/YAGO* are discovered with 95% confidence.

[15] Linked Movie Database, http://linkedmdb.org

[16] Weighted Jaccard similarity - a measure of the amount of common N-grams between two documents, taking into account the frequency of the N-grams.

3 Practical Application in Movies Domain

"Simplicity is prerequisite for reliability."

Edsger Dijkstra

3.1 Application requirements

Given a set of records containing movie-related information, we want to group together those records that refer to the same movie instance. We are going to achieve this in a pairwise fashion, comparing the records two by two and checking if they describe two different movies or not.

In the following sections, we will use Police Academy 4 as an example, to better illustrate our reasoning. Tables 2 and 3 show two records that we have on Police Academy 4. The purpose of the application is to verify and give a confidence score on whether both records contain information about Police Academy 4 or not.

The application should fulfil the following requirements:

- allow various formats of input data, with little or no code alterations;
- allow addition of multiple reference databases, with little or no code alterations;
- no dependencies between multiple running application instances so as to allow process parallelism;
- accuracy higher than 80%;
- little or no human intervention necessary;
- allow easy addition of new comparison methods;
- allow multiple decision models;

3.2 Tools used

For the implementation of the application, we have chosen *Python 3* as the main programming language. The reasons behind this choice are several, first and foremost being its popularity and the availability of numerous useful libraries. Secondly, its syntax simplicity and a vast support community allow a rapid prototype development. An added advantage is its availability on all platforms (unix, windows and mac). External modules we have used include:

- *guess_language*[17] - to identify the language a text is written in;
- *nltk*[18] - for text processing purposes, like computing the edit distance between two strings;

[17]The *guess_language* Python module, https://bitbucket.org/spirit/guess_language
[18]The Natural Language Toolkit, http://www.nltk.org/

Characteristic	Value
Name	Police Academy 4: Citizens on Patrol
People	**Cast Roles** Bobcat Goldthwait → Zed Tab Thacker → Tommy "House" Conklin Leslie Easterbrook → Debbie Callahan G.W. Bailey → Thadeus Harris Billie Bird → Mrs. Feldman George Gaines → Eric Lassard David Graf → Eugene Tackleberry Bubba Smith → Moses Hightower Michael Winslow → Larvell Jones Derek McGrath → Butterworth Steve Guttenberg → Cadet Carey Mahoney Tim Kazurinsky → Cadet Sweetchuck Lance Kinsey → Lt. Proctor Marion Ramsey → Sgt. Lavern Hooks Scott Thomson → Sgt. Chad Copeland **Crew Roles** David Rawlins → Editor Gene Quintano → Screenwriter Pat Proft → Screenwriter Neal Israel → Screenwriter Paul Maslansky → Producer Robert Saad → Cinematographer Jim Drake → Director
IMDb id	tt0093756

Table 2: *Record 1* on Police Academy 4.

- *roman*[19] - to translate from roman to arabic numerals. To be noted that it is limited to numerals lower than 5000;
- *scikit-learn*[20] - for machine learning algorithms like decision trees and SVMs;

For the neural network, we have used the FANN[21] implementation, written in C. It provides a flexible and easy to use multilayer feed-forward neural network, and allows the user to adjust parameters which control the number of layers of the ANN, the number of neurons per layer, the density of the ANN and the activation functions.

Since the data we had was available in JSON format, we have decided to use *elasticsearch*[22] for storage and information retrieval purposes. Elasticsearch is

[19]The *roman* Python module, https://pypi.python.org/pypi/roman
[20]Machine Learning in Python, http://scikit-learn.org/stable/
[21]Fast Artificial Neural Network Library, http://leenissen.dk/fann/wp/
[22]Elasticsearch, https://www.elastic.co/products/elasticsearch

Characteristic	Value
Name	Polisskolan 4: Kvarterspatrullen
Year	1987
Length	5280
Genres	Komedi
Directors	Jim Drake
Actors	David Graf Steve Guttenberg Michael Winslow Bubba Smith
Description	Rektor Lassard på Polisskolan har ett nytt projekt. Han vill träna ett antal civila frivilliga till att ingå i sitt nya projekt för att hålla ordning och reda i samhället – kvarterspatrullen. Han tar in sina gamla elever för att utföra uppgiften. Trots att projektet har starkt stöd från myndigheterna så har den rivaliserande kapten Harris hellre lust att se projektet misslyckas...

Table 3: *Record 2* on Police Academy 4.

a full-text search engine built on Apache Lucene[23] for applications dealing with complex search queries and large amounts of data.

We will present more details on how all these tools have been used in the following sections.

3.3 Datasets

3.3.1 Data formats

The datasets used in the experiments have been provided by the VionLabs team. Like we mentioned before, rather than keeping them in text files in a JSON format, we have preferred to store them using *elasticsearch*.

We have worked with three datasets, each containing a different amount of records:

- Dataset 1 - 324518 records with title, cast and crew information;
- Dataset 2 - 86443 records with title, release year, and occasionally cast;
- Dataset 3 - 178870 records with title, cast, description and other information.

All the datasets contain unique identifiers, which has facilitated the evaluation phase of our work. We have used the first two datasets for reference information, and the third as the one to be reconciled. Note that not all records contain all the information advertised, cast information or description or other fields may

[23]Apache Lucene - Java-based information retrieval library, https://lucene.apache.org/

be missing. Also, even though all the datasets respect the JSON format, they do not use a consistent key name scheme from one set to another.

3.3.2 Extracted features

In our work context, features are ways of comparing the available movie information. Depending on for which characteristics we have values, we may have more or less comparison options. In our experiments, we have used the following methods:

- titles comparison;
- language recognition;
- proper nouns extraction from the plot description;
- actors and other people lists match;
- title numerals match;
- release year match.

To compare the titles, we have employed the Levenshtein distance. First we convert both title strings to lowercase and remove any trailing blank characters. Then we compute the edit distance and divide it by the length of the longest string, to get a value in the [0, 1] interval. By division, we make sure the comparison with other pairs will be fair, regardless of absolute value of the edit distance. We want a high value to indicate a good match, so we return 1 minus the previously obtained value. Thus, a value of 1 will indicate a perfect match, while 0 will indicate completely different strings.
Example.
Title 1: "Police Academy 4: Citizens on Patrol"
Title 2: "Polisskolan 4: Kvarterspatrullen"
Levenshtein distance: 23
Title comparison value: 0.361
According to this result, we have a low degree of similarity between the two titles, even though we know that the titles refer to the same movie. Movie titles are usually given alternatives, depending on the country they are playing in. This is the case here: since we have crawled the information from a Swedish website, we get the Swedish version of the title. So, a movie may have different names, which is why the title match criteria alone is not enough to positively identify whether two records represent the same movie.

For the language recognition, we have used the *guess_language* Python package. We have applied the language recognition methods on the title and plot description characteristics of the available records. If the strings given as parameters are too short, we lengthen them by duplication. If one of the two strings is empty or missing, we assume identical language and return 1. The package works by relying on tri-grams to identify a language. Knowing the language of each record is useful because it allows us to assign lower importance to the titles comparison when we have records in different languages. If we have the same language, we know to expect a high title similarity.

Example. We consider the movie titles in the previous example.
Title 1 language: en
Title 2 language: sv
Language comparison value: 0
As we can see, we identify English for the first title, and Swedish for the second one. This language difference explains the low title comparison value obtained earlier.

One other feature is proper nouns match. We have extracted the proper nouns from the movie description and we have compared the obtained nouns with the people names listed as cast or crew. To accomplish this, we have used the NER system available in the *nltk* package. To identify named entities, the NER system uses a classifier trained on data from CoNLL[24], MUC-6 and ACE[25].
Example. To compute the proper nouns match, we check how many of the identified proper nouns can be found in the *Crew Roles* and *Cast Roles* lists.
Extracted proper nouns: Rektor, Lassard, Polisskolan, Harris.
Proper nouns match: 0.5
The nouns *Lassard* and *Harris* can be found in the *Cast Roles* list, so we have a 50% match.

Another comparison we make is between the lists of people existing in the two records. We take each person listed in the unreconciled record, and check if the name is also listed in the other record. The more matches we find, the better the score. If in one of the records there are no people listed, then a score of 0 is returned as an indicator that this feature should be ignored for this pair of records. If a name from the unreconciled record is not found, the overall score is decreased.
Example. In the case of *Record 1* and *Record 2*, we have a 100% people match, since all the names listed in *Record 1* can be found in *Record 2*.

A problem we encountered was that often we were matching a movie to one of its sequels or prequels. In this case, the titles are similar, the cast and crew lists almost identical, so it is difficult to differentiate between the two. In order to avoid this problem, we have added a comparison function which takes any numerals it finds in the titles and checks if they are the same or not. Many movies, like Rocky IV, use roman numerals to indicate the continuation of the series. So, first we translate all numerals we find to arabic numbers, and then we compare them. If the titles are similar and there is a numeral mismatch, then we return a negative score. A positive score is returned otherwise, and 0 if the titles do not contain any numeric values.
Example. For *Police Academy 4: Citizens on Patrol* and *Polisskolan 4: Kvarterspatrullen*, a score of 1 will be returned since both titles contain number *4*.

The release year is an important piece of movie meta-data since it helps discriminate between a remake and the original movie. The year comparison method returns the absolute value of the difference between the release years of the two records. A value of 0 or 1 indicates a match possibility, larger values

[24]The NER task of the CoNLL 2003 conference, http://www.cnts.ua.ac.be/conll2003/ner/
[25]Source: https://catalog.ldc.upenn.edu/LDC2011T02

mean that the two movies are most likely different.
Example.
Release year for Record 1: N/A
Release year for Record 1: 1987
Release year comparison: -1
The value -1 is returned to indicate missing information.

3.4 Movie reconciliation system architecture

3.4.1 General description

The best solutions are those tailored to fit as well as possible the problem at hand. The approach we have chosen for the movie reconciliation uses general techniques, but with carefully customized input. It is a compromise between using ad-hoc heuristics and a generic, off-the-shelf reconciliation software.

The general architecture of the application is split into components, which communicate with one another to generate the end results. Also, each component can be easily altered, thus allowing the users to adapt the application to their requirements.

In order to be able to find the unique identifiers of the unreconciled records, we need a set of labelled records with enough information to help us determine the correct matches. The application allows several labelled datasets, with different formats. The access to the information from these datasets about a movie instance is unified through the *TaggedMovie* class. It must be noted that when adding a new dataset, which can have a different format, this class has to be updated to reflect the new data source.

An unreconciled record is represented by an *UnreconciledRecord* class instance. The reason for using this class and not accessing directly the information from the record itself is to allow an increased modularity. Thus, if at some point we want to change the format of the unreconciled dataset, we just need to modify the *UnreconciledRecord* class, and not references to that record scattered throughout the entire application.

In Figure 6, the main steps of the entire reconciliation process are listed. The application receives as input a record to be reconciled, and multiple labelled datasets, used for reference information. First, from all the pairs possible, the most likely ones are selected by the filter component. Next, for each of the candidate pairs, the similarity measures are computed. Based on these similarity vectors, a classification algorithm is applied which will return the most likely match.

In the detailed description of each component, when we list the parameters, we refer to parts of the application that can be adjusted in order to fit other data environments. We have also highlighted these parameters using the blue elements in Figure 7 . Our application can manage data in other formats than those used during our experiments just by extending the two classes that control

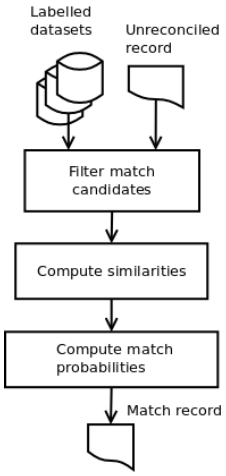

Figure 6: ***Reconciliation process overview.*** *The structure of the application is divided into three main components responsible for the main steps of the reconciliation process: filtering, computation of the similarities array and candidate pair classification.*

the data access: *UnreconciledRecord* and *TaggedMovie*. Obviously, with a different dataset outside the movie domain, the current filter method will no longer be appropriate. All the user has to do is change a variable in the configuration file to point to a different method. Likewise for the similarity measures, more can be added, or they can be changed completely. If more computation power is available, more candidates can be examined, also a parameter in the configuration file. Switching between classification algorithms is also done though the change of a configuration parameter.

3.4.2 Filter component

Parameters:

- number of candidate matches returned, default: 15;
- labelled datasets;
- filter method.

The large amount of entries in the search space does not allow comparing an unreconciled record with each and every labelled entry, since it would not be feasible in terms of computation time. The complexity for such an algorithm would be $O(n \times m \times comparison\_cost)$, with n - the number of tagged records, and m - the number of untagged records. Considering a search space containing 300000 entries and having 200000 records to reconcile, that would mean 6×10^{10} pairs to compare. Assuming 0.5 seconds per pair, the computation would be finished in about 800 years without any parallelization. So in order to avoid such a lengthy execution time, the obvious mismatched pairs will be filtered out.

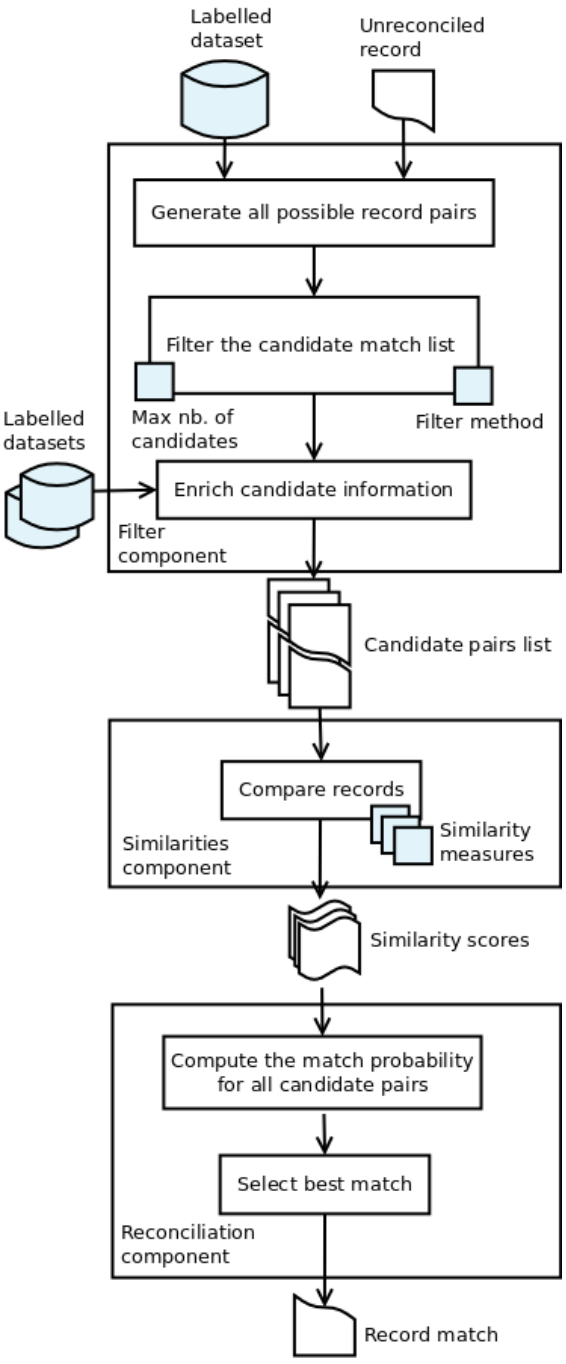

Figure 7: ***Reconciliation process detailed stages.***

The filter component takes as input the record to be matched and one or multiple datasets with already tagged information. The number of tagged datasets depends on the number of distinct movie-related information resources. It is more convenient to keep them separate for usability reasons, in case they are needed for other applications as well.

First, the dataset with the largest number of entries is chosen and from it, a limited amount of entries is selected based on title and actor similarity. The filtering is done using an elasticsearch *match* query. In our implementation, we have set the limit to 15 results in order to have a reasonably fast execution of the algorithm over the entire dataset. It is important to note that we have used an *or* query, and we have assigned to the actor similarity a higher weight than to the title similarity. The reason for this is that the filter operation can introduce bias in the way it selects the results, and leveraging between actors and title should increase the odds of capturing the correct match among the filtered entries, especially when we are searching for non-English titles. Should the user consider this filter method inadequate, then it can be changed by altering the value of *config.FILTER_METHOD* in the configuration file.

We refer to the second operation that the filter component performs as data enrichment. Based on the unique identifiers of the movies returned by the filter operation, we add the corresponding information from the other tagged datasets. This data is exposed to the other components of the application through the *TaggedMovie* class instance. The user can decide which datasets will be considered by modifying the *constants.LABELED_INDEXES* in the configuration file.

3.4.3 Similarities component

Parameters:

- similarity measures.

The similarities component is responsible for computing the correlation between an unreconciled record and a tagged one. It takes as input a list of *TaggedMovie* objects and an *UnreconciledRecord* object. Then, for each $\langle$*UnreconciledRecord*, *TaggedMovie*$\rangle$ pair, it applies a number of methods designed to compare certain features of the two entities. For each pair, it outputs a list of float values, representing the return values for each of the similarity methods applied. We will refer to this list as *sample*.

As we have shown earlier, on the *Police Academy* example, the similarity measures that we have used include the edit distance between titles, language comparison, people match, a comparison of the numerals in the title, the difference between the release years and the proper nouns match. In the *Evaluation* section, we will see which of these measures have the most impact on the match decision.

3.4.4 Reconciliation component

Parameters:

- decision model.

The reconciliation component is responsible for the actual decision on whether the two records are a match or not. This is done in two steps. First, we apply the reconciliation technique on each sample, obtaining a confidence value. Then, in the selection phase, the most likely match for the unreconciled record is chosen and the result is returned.

Each sample represents the similarity measures between two movies. For each sample, the output will be a real number in the $[0, 1]$ interval, representing the confidence that the two samples are about the same movie. We have implemented and compared the following reconciliation techniques: rule-based, naive Bayes, decision trees, SVMs and neural networks, and we can select which one to use through the configuration file by adjusting the *constants.RECONCILIATION_TYPE* parameter.

The selection process means choosing the sample with the highest confidence value. However, if that value is under a certain threshold, it indicates two possibilities: either the correct match is not in the tagged dataset, or the correct match has not been selected by the filter component. Depending on the correct match, the pair will be counted either as a true negative or as a false negative.

3.5 Reconciliation techniques

We have chosen to implement the heuristic and machine learning oriented approaches, disregarding graph-based techniques. The reason for this choice is that in the movie context, the resulting graph would be extremely large, and all of the graph algorithms mentioned in the *Graph-oriented approaches* section have a high computational complexity, and would not be scalable. Unlike social graphs, where an exact match is not always needed, in the movie context we do not want to publish erroneous information. This strict restriction makes these types of algorithm not viable for our application.

Since we do not want to build everything from square one and the focus of our work is on the evaluation of various algorithms that can be used in the movie reconciliation context, we have used external libraries, like the *scikit-learn* Python module for naive Bayes, SVMs and decision trees. In this section, we will provide detailed information on how we have employed these libraries, what parameters we have chosen and the reasoning behind these choices.

For the algorithms that require training, the training data is represented by the matrix ϕ, containing the comparison results between the characteristics from a number of movie information samples, accompanied by the correct match decision $D_i, i \in \{YES, NO\}$.

3.5.1 Rule-based approach

We have implemented 4 intuitive methods, each taking into account a different subset of features. This allows us to comment on the relevance of each feature in particular, and also lets us see how much seemingly similar rules can differ in performance.

When taking into account all the similarities, first we look at the language match. If the two records are written in different languages, then there is a high chance that the titles will not match, and then the title match should not count as much as in the normal case. The same reasoning is applied for the proper nouns match, because often there are multiple names for the same location, depending on the nationality of person referring to it. However, the names of the cast and crew do not change from a language to another, so this is not influenced by the language similarity. So, depending on the language the records are written in, the other parameters mentioned here are assigned different weights.

The similarities relying on numerical features are language-independent, so they have a fixed maximum weight. Regarding release years, equality would be preferred, a difference of one year is also acceptable since it could mean that a movie was published towards the end of the year and online sources have stored it as having been published the following year. A difference larger than two is a strong indicator that the records are about two distinct movies, or a movie and its sequel/prequel. Another sign of distinct movies belonging to the same saga are the numerals present in the title, which accounts for up to one fifth of the final match confidence. On the other hand, a discrepancy in numerals decreases this confidence by up to 30%.

For the other subsets of similarities, we have applied the same reasoning as above, but we have adjusted the weights to reflect the lack of additional attributes. For example, when considering only title and people similarities, each has a 50% stake in the final decision. The rules are simple, easy to understand and fast to execute. The downside is that they are specific to the current dataset and the similarities defined on it, and if new information is added, the rules do not automatically adapt to it. Also, they do not incorporate previous match results, so they do not get better in time.

3.5.2 Naive Bayes

When applying Bayesian learning to our reconciliation problem, we are looking to determine the most probable decision D for a similarities array ϕ_{AB}, given the available training data. In other words, classifying the pair that the similarities array represents into a match or a non-match. The Bayes model relies on the Bayes theorem, which given our problem context, looks like this:

$$P(D|\phi) = \frac{P(\phi|D)P(D)}{P(\phi)}$$

, where

- $P(D)$ - prior probability of match/non-match decisions;
- $P(\phi)$ - prior probability that the similarities array ϕ will be observed;
- $P(\phi|D)$ - probability of observing the similarities array ϕ given the decision D, also known as the *likelihood* of the data;
- $P(D|\phi)$ - what we want to know is the *posterior probability*, which is the probability of decision D, given ϕ.

The naive assumption of feature independence allows us to compute the match probability according to the following formula:

$$decision(\phi) = \arg\max_{d} P(D = d) \prod_{i=1}^{p} P(\theta_i | D = d)$$

For the implementation, we have used the GaussianNB class from the *scikit-learn* library. Thus, the likelihoods of the similarities are computed assuming a normal distribution. The class has methods both for directly predicting the class, and for returning the probabilities for each possible class, thus allowing the user to select an appropriate threshold.

3.5.3 Decision Tree

Using decision trees is similar to the rule estimation method, except that the rules are determined automatically, based on the training set. Having a balanced training set ensures that resulted tree will not be biased towards a certain category. An advantage of the decision trees is that they can be visualized, which makes it easy to understand why a certain decision has been made.

The set from which the decision tree is created consists of the similarity arrays computed by the similarities component. For the implementation of the decision trees, we have also used the *scikit-learn* library. The *DecisionTreeClassifier* class allows customizing settings such as the set diversity metric (entropy or Gini impurity), the maximum depth of the tree or the minimum number of samples required to split a node. We have chosen the entropy as the split quality criterion and compared the performance obtained by trees of various sizes.

3.5.4 SVM

Due to the use of kernels, SVMs perform well even in high dimensional spaces. That means that we can add as many similarity methods as we deem necessary and SVMs will still be effective as long as we provide enough training samples. Their versatility is also owed to the fact that we have the freedom to design custom kernel methods, should we require more than the standard ones.

Applied to the movie reconciliation context, the kernel definitions will use the similarities arrays as the input vectors. In our reconciliation application, we offer the possibility to choose which kernel to use by setting the kernel parameter to *linear*, *poly*, *rbf* or *sigmoid*.

Because training and testing SVMs with a large degree polynomial kernels on large datasets is an expensive process time-wise, we have chosen to use only degree 2 polynomials in our experiments. Moreover, low degree polynomials have been shown to be more effective when dealing with text classification tasks [50] [51]. Higher degree polynomials tend to over-fit the training data, thus yielding poorer results.

For the SVMs implementation we have elected to use the *SVC* class from the *scikit-learn* library. It is a flexible class, granting the user the power to customize to the smallest detail the class instances, by setting parameters such as the number of permitted instances to classify erroneously, the various kernel tuning variables, the tolerance for the stopping criterion and more. By default, an SVM classifier will only return the class predicted for each sample instance. However, the library also provides the option to return the probability that a sample instance belongs to a certain class. Because this is done through five-fold cross validation, it considerably lengthens the execution time of the classifier.

3.5.5 Neural network approach

Given the capacity of neural networks to model extremely complex functions, this approach was the next logical step in our search for the most suitable decision mechanism in the movie-information reconciliation domain. Of course, just to name the neural networks as the best solution to our problem is not enough: we also need to determine the most appropriate values, or at least bounds for the main parameters of the neural network. In order to do that, first we have restricted the design of the network - no backward connections between the neurons, and secondly we have empirically tested multiple network sizes and shapes on our dataset.

Restricting the network design to only feed-forward networks does not affect the results of the experiment, because we are not dealing with a time-dependent problem. According to Hassoun [52], a recurrent architecture is most suitable for time-dependent inputs, like modelling the process of an unknown plant. In the case of a recurrent network, the dependencies between the neurons vary not only according to the input signals, but also according to the output produced by those inputs at a given time instance. However, in our case, there are no time dependencies and thus, a feed-forward network will suffice.

With regards to implementation, we have chosen the *FANN* library for the neural network part of our application. The library allows the user to create a feed-forward neural network, train and execute it through simple function calls. Several different parameters can be adjusted before and during the training phase, the most important being the learning rate and the size and shape of the network. During the experiments, we have decided on 0.7 as the learning rate, with a symmetric sigmoid activation function .

The shape and size of the network matters and greatly affects the results obtained. Should we choose too many layers and neurons, the learning process will be slow and we risk over-fitting the input data, not being able to generalize.

Should we choose too few layers and neurons, the network will not be able to learn the rules and the error rate will most likely oscillate, without reaching low enough levels. Therefore, during the experiments, we have tested several network configurations.

4 Evaluation

"A program that has not been tested does not work."

Bjarne Stroustrup

4.1 Experimental methodology

4.1.1 Hardware

The experiments have been run on a Dell Inspiron N5110, equipped with an Intel® Core^{TM} i7-2630QM CPU with 8 cores at a 2.00GHz frequency, 8 MB L3 cache and 8 GB RAM. The system is running Ubuntu 14.04.2 LTS.

4.1.2 Datasets

The training set includes 1000 samples: 500 positive, 500 negative, so there will be no bias in either direction. In the similarities arrays, duplicates can be found: it is possible for 2 different pairs made of 4 distinct movies to have identical similarities arrays.

The test data is comprised of 178870 records for which we know the correct unique identifier. After the filtering phase, the list of candidates contains the correct match in 173278 of cases, so 96.87%. The classification algorithms have the task of finding the correct match from the list of candidates.

Depending on whether a pair is an actual match or not and on the algorithm decision, there are 4 possible type of results:

- *true positive* (TP) - the pair is an actual match and has been identified as such;
- *true negative* (TN) - the algorithm could not find a match for an unreconciled record, so the pair is not a match and has been identified as such;
- *false positive* (FP) - the pair has been falsely classified as a match;
- *false negative* (FN) - the pair is an actual match, but it has not been recognized by the algorithm.

4.1.3 Evaluation metrics

If this application is to be used as part of a public user service like Vionel, then it is important to know how much we can trust the results, how confident we are that the computed matches are correct and can be published online. In order to evaluate performance of the application, we have considered the following metrics:

- *accuracy* - expresses the percent of correct results, regardless of whether they represent a positive match or not;

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

- *precision* - also known as *positive predictive value*, it shows how much confidence we can have in a positive match, the probability of two records being identified as one movie when, in fact, this is not the case;

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

- *recall* - it indicates what percent of positive matches has been identified from all the positive matches evaluated by the learning algorithm;

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

- *false positive rate* - it expresses how many mismatches have been classified as matches out of all the negative pairs;

$$\frac{FP}{TN + FP}$$

, which have been expressed in percentages in the *Experimental results* section.

A low recall is not what we are looking for, but it would be better than having a low precision. A low precision means that we cannot rely on the matches provided by application and that we would need to manually verify each and every one of them and it is precisely this effort that we wish to avoid. To better illustrate the meaning of these metrics, consider the following fictive example: out of a set of 100 pairs, 80 are true matches. Applying a reconciliation algorithm on this set, 60 pairs of the true matches are discovered as such, but so are 10 other pairs from the non-matches. Therefore, we would obtain these metrics:

- $accuracy = \frac{60+10}{60+10+10+20} = 0.7$;
- $precision = \frac{60}{60+10} \approx 0.85$;
- $recall = \frac{60}{60+20} = 0.75$;
- $false\_positive\_rate = \frac{10}{10+10} = 0.5$.

In order to see how the algorithms behave when using different features, we have selected 4 sets of features on which we have run tests:

- set 1 - S_1: title match, language comparison, people match, numerals match, release year difference, proper nouns match (all features);
- set 2 - S_2: title match, numerals match, people match, release year difference;
- set 3 - S_3: title and people match;
- set 4 - S_4: title match and release year difference.

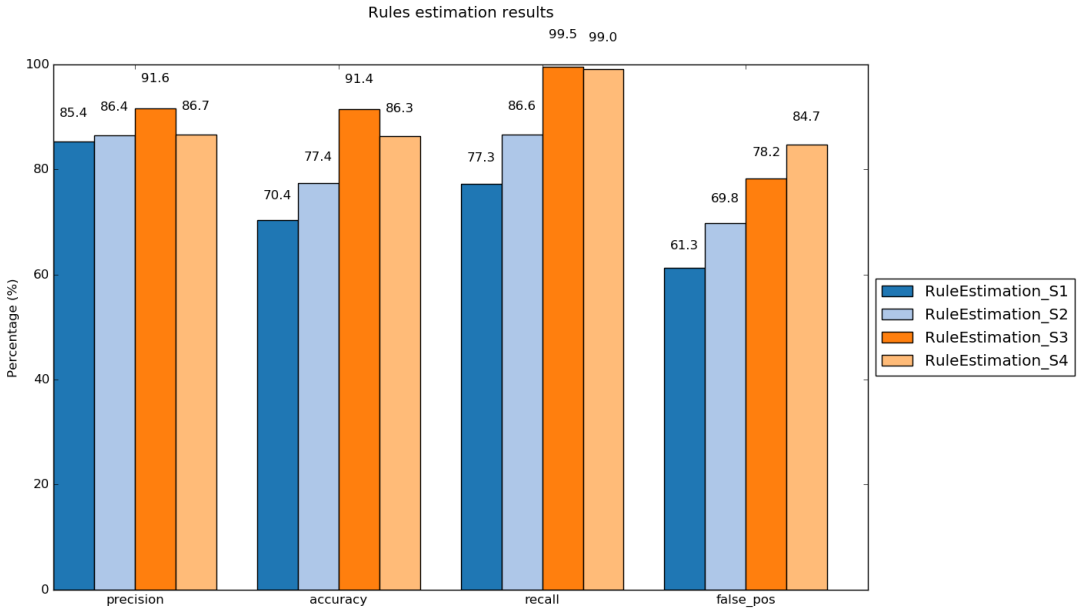

Figure 8: ***Rules estimation results.*** *Comparison of four rule-based methods applied on the four feature subsets.*

4.2 Experimental results

4.2.1 Rule-based approach

In Figure 8 we have plotted the results obtained when applying our rules on the feature subsets. Even though we would expect that more features would give us better results, we can see that in the current scenario, using a subset of four features works best. With the full set of features, the lack of information in certain records leads to low similarity measures which decrease the overall match probability of a pair. Thus, even though we are quite confident in the predicted matches (85.4% precision), there are a lot of true matches that we have missed, fact reflected by the low recall.

With heuristics based on the S_2 feature subset, we obtain not only a high precision, but also a higher recall. We do identify much more of the true matches, but at the cost of more false positives. Using the title and cast similarities, the amount of false positives is still relatively high, but we have a higher confidence that the predicted matches are indeed correct. Moreover, from the true matches in the candidate list, almost all of them are recognized. The release year is not as good a discriminative criterion as the cast and crew list is. With S_4, more true matches are misclassified as negative and we get a much higher false positive rate.

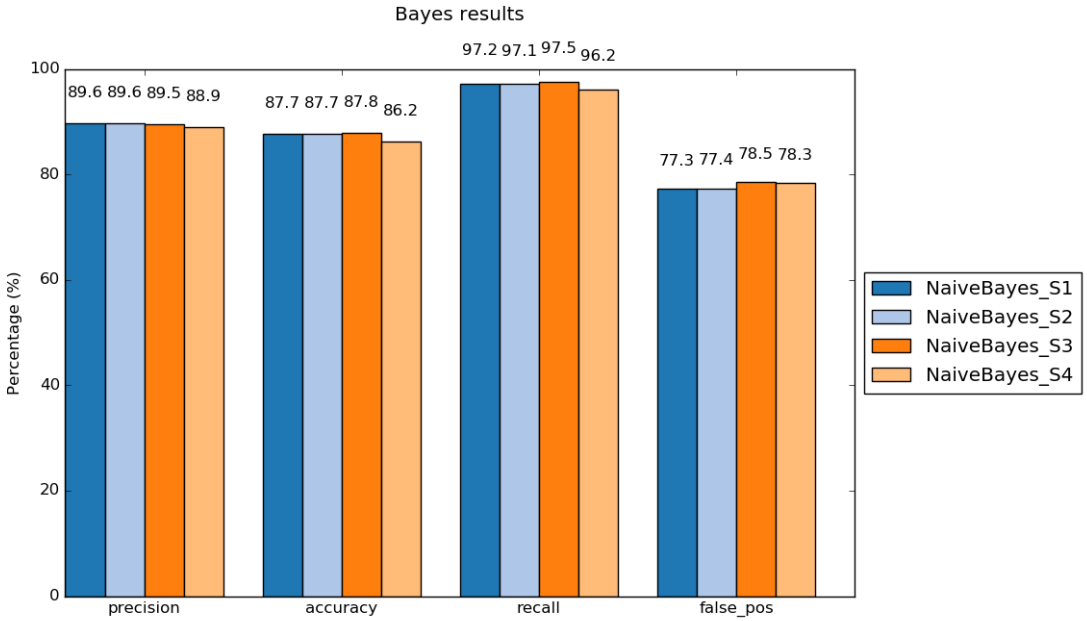

Figure 9: ***Naive Bayes - comparison by the features used.*** *In NaiveBayes_Sx, x denotes the feature set used for the training phase.*

4.2.2 Naive Bayes

As we have mentioned in the description of the system, we have used naive Bayes assuming a normal distribution of the samples, and trained 4 separate instances on the four feature sets. We can observe in Figure 9 that comparable performances are obtained regardless of the combination of features used. However, there are subtle differences which should be noted.

Looking at the set 3 and set 4 results, we can see that knowing the release year is less relevant than knowing the cast and crew, since we have higher percentages for the precision, accuracy and recall. However, there is also a slightly higher number of pairs which have been mistakenly identified as matches. Using more features for the training allows us to reduce this number by almost 1%. The difference between using the full set of features and a partial set of 4 is not very noticeable, indicating that language comparison and the proper nouns match do not contribute much to the classification with the current dataset.

Overall, using more features gives better results in the case of naive Bayes. However, applied on our test set, it only identifies 87.5% of the true matches listed in the candidate lists.

4.2.3 Decision Tree

In the case of decision trees, it is the entropy or the Gini metric that determines which features are most relevant in the classification process, so it is

not necessary to train the decision trees on the feature subsets defined earlier. Instead, we are comparing the metrics obtained by training trees up to several levels of depth. The higher the depth, the higher the risk of over-fitting the training data and not generalising enough.

The over-fitting problem is obvious when we look at the results. In the training phase, the accuracy grows from 98.1% to 98.7%. In contrast, in the testing phase, the accuracy drops drastically to 89.6%, almost 10% smaller than the training one. And as the depth increases, the accuracy keeps decreasing, while the false positives rate is going up. The effect is rather visible for trees with maximum depth 4 and higher.

In Figure 10 we have the output from applying the decision tree classifier on our test set, with entropy as the split criterion. Again, we see very high values in precision, accuracy and especially recall. Confirming the Bayes results, here the title is the first feature to split the set by, followed by the cast and crew match, as we can see in the tree with a maximum depth of 2 in Figure 11.

With a 3 layer tree, we obtain the best precision, 90.7%. The 3 layer tree manages to find a balance between the amount of true positive and false positive pairs, also having the lowest false positive rate. While with the 2 layer tree we obtain the best generalisation (best accuracy - 89.6%) and a higher number of true positive pairs, we also get a higher number of false positives, which we would prefer to avoid.

4.2.4 SVM

Kernel	**TP**	**FP**	**TN**	**FN**	**precision**	**accuracy**	**recall**	**false+ rate**
Linear	153327	17267	4914	3362	89.8	88.4	97.8	77.8
Polynomial, d=2	152095	17592	4946	4237	89.6	87.7	97.2	78.0
RBF, γ=0.5	146802	17885	4856	9327	89.1	84.7	94.0	78.6
RBF, γ=1	145262	18748	4786	10074	88.5	83.8	93.5	79.6
RBF, γ=2	140003	18531	5577	14759	88.3	81.3	90.4	76.8
Sigmoid, r=0.3	90174	72216	3849	12631	55.5	52.5	87.7	94.9
Sigmoid, r=0.5	89852	71671	4037	13310	55.6	52.5	87.0	94.6
Sigmoid, r=1	89104	69751	4511	15504	56.0	52.3	85.1	98.9
Sigmoid, r=2	86553	67326	5462	19529	56.2	51.4	81.5	92.4

Table 4: ***SVM results.*** *The precision, accuracy, recall and the false positive rate are all expressed in percentages.*

In Table 4, we have presented the best results obtained when running experiments with SVMs as the classification algorithm, and varying the kernels and their parameters.

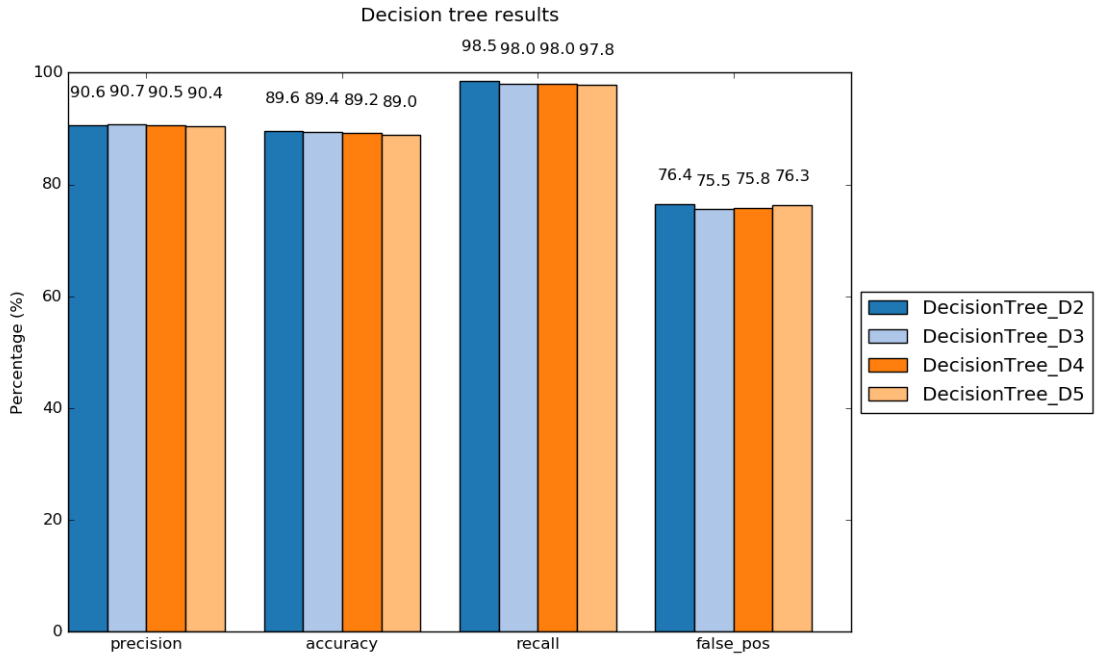

Figure 10: ***Decision trees - comparison by maximum depth.*** *In the name DecisionTree_Dx, x denotes the maximum depth of the tree.*

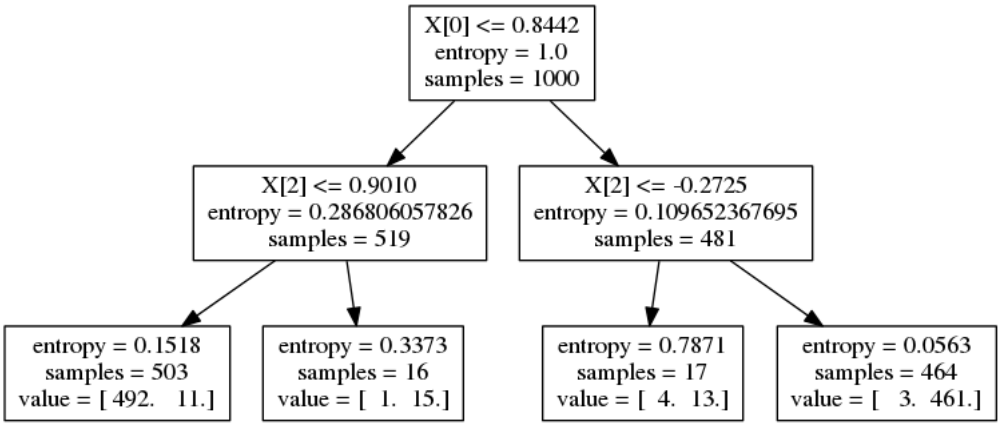

Figure 11: ***Decision tree with max depth 2.*** *The first split criterion is the title match (X[0]), followed by the people match (X[2]).*

We can see that the best performance is obtained with the linear kernel: the least amount of incorrectly classified true matches, and also the least amount of record pairs wrongly predicted as referring to the same movie, leading to a good confidence in the results. A similar performance is obtained in the case of a degree 2 polynomial kernel, but with slightly less true matches identified. With a RBF kernel, as the γ parameter grows, we observe a decrease in the number of predicted true positives and as a result, an increase in the number of false negatives.

While the linear, polynomial and RBF kernels give reasonable results, that is not the case with the sigmoid kernel. More than half of the predicted matches are, in fact, not matches. And compared to the other kernels, we have a much lower recall, a drastic decrease in how many of the candidate true matches are recognized.

4.2.5 Neural network approach

Network size	**TP**	**FP**	**TN**	**FN**	**precision**	**accuracy**	**recall**	**false+ rate**
6x3x1	145173	30192	3099	406	82.783	82.894	99.721	90.691
6x5x1	159075	16493	2846	456	90.606	90.524	99.714	85.284
6x7x1	159612	15956	2913	389	90.912	90.862	99.757	84.562
6x12x1	159236	15973	3233	428	90.883	90.831	99.732	83.167
6x25x1	160159	15105	3297	309	91.382	91.383	99.807	82.083
6x35x1	159700	15161	3499	510	91.330	91.239	99.682	81.249
6x50x1	159796	14455	4022	597	91.704	91.585	99.628	78.232
6x100x1	160440	15097	3009	324	91.400	91.379	99.798	83.381
6x200x1	158763	17617	2275	215	90.012	90.031	99.865	88.563
6x10x25x1	159348	15329	3679	514	91.224	91.143	99.678	80.645
6x10x5x1	156040	18348	3798	684	89.479	89.360	99.564	82.850
6x5x15x1	154393	19610	3886	981	88.730	88.488	99.369	83.461
6x20x7x1	159364	14960	3937	609	91.418	91.296	99.619	79.166
6x10x5x15x1	156587	16598	4374	1311	90.416	89.988	99.170	79.144
6x5x25x5x1	156942	17007	4191	730	90.223	90.084	99.537	80.229

Table 5: ***Neural network results.*** *The precision, accuracy, recall and the false positive rate are all expressed in percentages.*

We present in Table 5 the most significant results we have obtained working with neural networks as the reconciliation algorithm. Regardless of the network size, the performance is very good, with very small differences from one network instance to another. Exceptions occur when we have significantly less neurons in the hidden layer than input features. As we can see when we have a 3 layer network with 3 neurons in the hidden layer, the precision drops from an average of 90% to 82%, and have almost double the number of false positives compared to the other networks.

The 3 layered network with 25 hidden nodes identifies the most of the true pairs present in the candidates list, with a 99.8% recall. The precision is also high, but with 25 more hidden nodes, the count of false positives is smaller. Increasing the network size to 4 layers does not improve the performance, on the contrary, slightly less of the true matches are correctly classified. The same occurs with a 5 layer network as well, so a more complex neural network does not guarantee better results.

4.2.6 Discussion

The false positives rate is always high because this metric compares the number of false positives relative to the amount of true negatives which is always relatively low.

In the plot in Figure 12, we have displayed the results from all the previously mentioned experiments, but in terms of result instance types: TP, TN, FN, FP. We can observe that in general, most of the applied algorithms have a decent performance. The most notable variations occur in the case of the rule-based approaches, where we have a significant increase in the number of pairs not recognized as matches.

An increase in the number of pairs erroneously predicted as true matches is also observed in the case of the neural network with a hidden neuron count lower than the input size. A more drastic surge in false positives also occurs for the SVMs with RBF kernel.

Throughout most of the experiments, true matches are rarely misclassified as incorrect pairs, but even less errors are made by the neural networks in comparison to the other algorithms.

From what we have observed, the main causes for classification errors are:

- a true match is not included in the pair candidate list after the filtering phase;
- the two records are part of a saga, and the title gives no clue to this;
- missing field information - e.g. no release year available;
- the records do not contain enough text for the application to properly identify the language.

Overall, we can say that for our dataset, the best approach is either a wisely designed set of rules, or a neural network - keeping in mind to choose more neurons in the intermediate layers than the number of inputs. The performance of these algorithms is closely followed by the naive Bayes and the decision trees. The SVMs have proven to be the least reliable classifier in the movie reconciliation context.

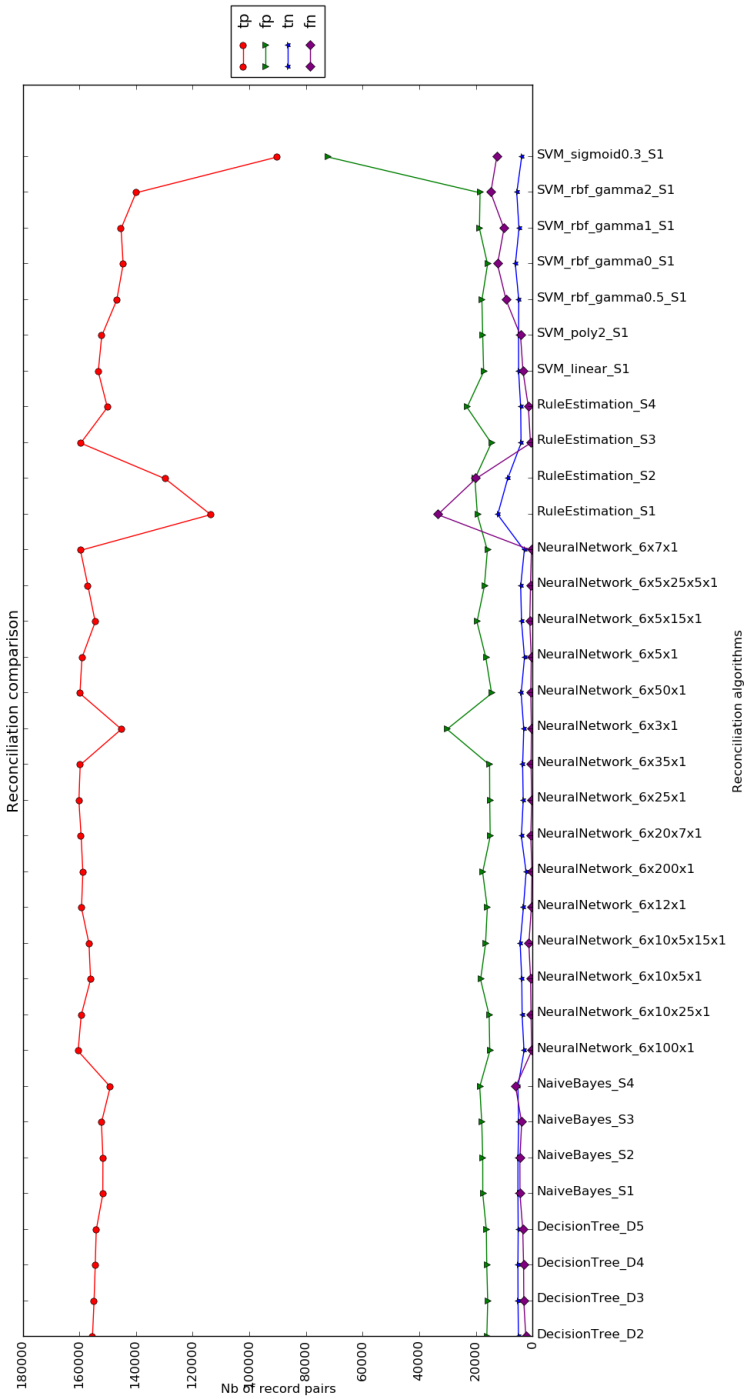

Figure 12: ***Overall reconciliation comparison.***

5 Conclusion

> "Do what you think is interesting, do something that you think is fun and worthwhile, because otherwise you won't do it well anyway."
>
> Brian Kernighan

Reconciliation has become an important research topic due to the existence of numerous heterogeneous information resources. When attempting to collect relevant data in a certain domain, *identification* and *disambiguation* problems arise: dealing with similar records corresponding to the same unique entity and differentiating between seemingly identical records which actually refer to different entities. Further complications include missing values, inconsistent formatting, different languages, large number of records. Performing this process manually is definitely not feasible when handling hundreds of thousands of records.

In this thesis, we have addressed the problem of reconciling movie information gathered from various web resources. We believe we have achieved our goal of creating an application that is suited to the movie context, while at the same time allowing for rapid extension to fit other problem variants.

The application we are proposing works in stages. First, we limit the number of candidate pairs to the most likely ones. Then, considering the data we have had available, both labelled and unlabelled, we have implemented six comparison methods which express the similarity degree for a pair of movie records. Based on these measures, we have compared multiple classification algorithms with various parameters to determine whether the pair in question is a match or not.

Our application can be extended to datasets from other domains easily for the following reasons:

- it can handle input data with other formats, the user just having to extend two classes which define the data accessors;
- should a new labelled dataset become available, its addition to the computation is done simply by adjusting one configuration parameter;
- if it is used in a different context, the similarity component can be instructed to employ a different set of similarity methods;
- the classification algorithm can be selected from the a wide range (naive Bayes, decision tree, SVM, neural network), or a new one can be added.

We have tested our application on a practical problem, the identification of movie record matches. The supervised algorithms have been trained on a set of 1000 samples, the positive samples having an equal share to the negative ones. The test set contains almost 180000 records, uniquely labelled so there was no

need for statistic result estimation, we have shown the actual numbers obtained in the *Evaluation* section. According to our results, we can conclude that the best choice for the classification algorithm would be either a neural network, or a carefully selected set of rules. If we would know that the data available does not change, then a set of rules would be the less computationally intensive and less time consuming. However, if the data is expected to go often through changes, a neural network would be easier to adapt to these changes rather than having to manually modify the classification rules. Should the neural network be considered too complex, naive Bayes and decision trees have also shown a satisfactory performance.

Especially looking at the results using a set of rules and those using neural networks, we have clearly fulfilled our goal of improving the precision from 60% to 90%. However, it should be noted that the initial precision has been computed on different datasets. After discussing the results with the VionLabs team, it has been decided that they will not choose to implement a complex machine learning reconciliation approach, but that they will improve the existing set of rules.

It is our view that there is no unique solution to reconciliation. Data matching is domain and data dependent: it requires extensive domain knowledge, knowledge about data matching techniques, occasionally manual intervention too. And even having all that, the results will still strongly depend on the data available to match against.

6 Further work

> "Thus we may have knowledge of the past but cannot control it; we may control the future but have no knowledge of it."
>
> Claude Elwood Shannon

During this research, we have implemented an application to facilitate the integration of movie records from multiple sources to one database, by determining the proper unique identifier for the movies the records refer to. The application we have developed so far can be improved further by exploring several areas: execution time, results quality, performance evaluation, and user friendliness.

As we have mentioned, the experiments have been run on a single machine. However, the nature of application and the architecture itself allow for parallelization in order to reduce the execution time. One possibility is to split into groups the records that need to be reconciled and run the groups in parallel, as there are no dependencies from a record computation to another. Also, if multiple machine learning algorithms have been selected to be applied on the input data, each one can be run separately and concurrently with the other.

In our application, we have used a standard blocking technique based on the movie titles and actors. Employing other filtering techniques, like those mentioned in [23], [24] might improve the accuracy of the results, by including more true matches in the candidate pairs. Also, to avoid the case when the sorted neighbourhood block technique does not function properly (more records than the window size correspond to a value of the key fields), an idea would be not to use a window size at all, but to take the entire block of records respective to a key field value, plus a number of previous and successive records in the ordered list, and generate candidate pairs among these records. How many extra records are chosen can be a fixed value, or it can be a function of the number of the initial records that matched the key field value.

Since we are using supervised techniques, the training data is of crucial importance. In order to better evaluate the performance of the algorithms and also asses the influence of the training set, more work can be done in the training set creation. More tests can be done on a training set containing various ratios of positive to negative samples, thus allowing us to determine which ratio would give the best results. Moreover, we can be more particular about the samples we want included in the training set.

There are cases in which the characteristics we have compared are not enough to confidently classify a movie record pair, especially if one or more fields are missing. However, other fields may be available, which would permit reaching the correct decision. In order to do that, we would need to add more comparison methods, more elements in the characteristics array, according to the data we

have available. For example, we could compare poster images. Another possibility is text comparison using the Soundex algorithm [53], especially useful in the case of data known to contain spelling mistakes. Keep in mind that this will increase the execution time, so it is best to customize the comparison methods to the data available.

To make our application more user friendly, a graphical interface would facilitate operations such as adding unreconciled records and labelled data, selecting or adding comparison methods, or choosing classification algorithms. Installation and configuration of the application would be easier and more intuitive, requiring less documentation research for the basic uses.

7 References

[1] IMDb.com Inc. Database statistics, 2015.

[2] Anne Håkansson. Portal of Research Methods and Methodologies for Research Projects and Degree Projects. *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering FECS'13*, pages 67–73, 2013.

[3] H.B. Newcombe, J.M. Kennedy, S.J. Axford, and A.P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

[4] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[5] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710, 1966.

[6] William B. Cavnar and John M. Trenkle. N-Gram-Based Text Categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[7] Gregory Grefenstette. Comparing two language identification schemes. *Proceedings of the Third International Conference on the Statistical Analysis of Textual Data (JADT'95)*, pages 263–268, 1995.

[8] Olivier Teytaud and Radwan Jalam. Kernel-based text categorization. *Proceedings of the International Joint Conference on Neural Networks (IJCNN'2001)*, 2001.

[9] Timothy Baldwin and Marco Lui. Language identification: The long and the short of the matter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 229–237. Association for Computational Linguistics, 2010.

[10] Lisa F Rau. Extracting company names from text. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume 1, pages 29–32. IEEE, 1991.

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

[12] Daniel M Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201. Association for Computational Linguistics, 1997.

[13] Satoshi Sekine. NYU: Description of the Japanese NE system used for MET-2. In *Proc. Message Understanding Conference*, 1998.

[14] Masayuki Asahara and Yuji Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 8–15. Association for Computational Linguistics, 2003.

[15] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[16] Marius Pasca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge. In *AAAI*, volume 6, pages 1400–1405, 2006.

[17] Doug Downey, Matthew Broadhead, and Oren Etzioni. Locating Complex Named Entities in Web Text. In *IJCAI*, volume 7, pages 2733–2739, 2007.

[18] Sriparna Saha and Asif Ekbal. Combining multiple classifiers using vote based classifier ensemble technique for named entity recognition. *Data & Knowledge Engineering*, 85:15–39, 2013.

[19] Şerafettin Taşcı and Tunga Güngör. Comparison of text feature selection policies and using an adaptive framework. *Expert Systems with Applications*, 40(12):4871–4886, 2013.

[20] Anirban Dasgupta, Petros Drineas, Boulos Harb, Vanja Josifovski, and Michael W Mahoney. Feature selection methods for text classification. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 230–239. ACM, 2007.

[21] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[22] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001.

[23] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27. Citeseer, 2003.

[24] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM, 2000.

[25] Peter Christen. Automatic training example selection for scalable unsupervised record linkage. In *Advances in Knowledge Discovery and Data Mining*, pages 511–518. Springer, 2008.

[26] Daniel Whalen, Anthony Pepitone, Linda Graver, and Jon D Busch. Linking Client Records from Substance Abuse. *Mental Health and Medicaid State Agencies. SAMHSA Publication No. SMA-01-3500. Rockville, MD: Center for Substance Abuse Treatment and Center for Mental Health Services, Substance Abuse and Mental Health Services Administration*, 2001.

[27] Miranda Tromp, Anita C Ravelli, Gouke J Bonsel, Arie Hasman, and Johannes B Reitsma. Results from simulated data sets: probabilistic record linkage outperforms deterministic record linkage. *Journal of clinical epidemiology*, 64(5):565–572, 2011.

[28] Peter A Chew and David G Robinson. Automated account reconciliation using probabilistic and statistical techniques. *International Journal of Accounting & Information Management*, 20(4):322–334, 2012.

[29] Johannes Kobler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012.

[30] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 173–187. IEEE, 2009.

[31] Lyudmila Yartseva and Matthias Grossglauser. On the performance of percolation graph matching. In *Proceedings of the first ACM conference on Online social networks*, pages 119–130. ACM, 2013.

[32] Nitish Korula and Silvio Lattanzi. An efficient reconciliation algorithm for social networks. *Proceedings of the VLDB Endowment*, 7(5):377–388, 2014.

[33] Peter Christen. A two-step classification approach to unsupervised record linkage. In *Proceedings of the sixth Australasian conference on Data mining and analytics-Volume 70*, pages 111–119. Australian Computer Society, Inc., 2007.

[34] Harry Zhang. The optimality of Naive Bayes. *AA*, 1(2):3, 2004.

[35] Christopher Dozier, Peter Jackson, Xi Guo, Mark Chaudhary, and Yohendran Arumainayagam. Creation of an expert witness database through text mining. In *Proceedings of the 9th international conference on Artificial intelligence and law*, pages 177–184. ACM, 2003.

[36] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[37] J. Ross Quinlan. Improved use of continuous attributes in C4.5. *Journal of artificial intelligence research*, pages 77–90, 1996.

[38] Leo Breiman. Technical note: Some properties of splitting criteria. *Machine Learning*, 24(1):41–47, 1996.

[39] Vassilios S Verykios, Ahmed K Elmagarmid, and Elias N Houstis. Automating the approximate record-matching process. *Information sciences*, 126(1):83–98, 2000.

[40] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[41] Weifeng Su, Jiying Wang, and Frederick H Lochovsky. Record matching over query results from multiple web databases. *Knowledge and Data Engineering, IEEE Transactions on*, 22(4):578–589, 2010.

[42] Mikhail Bilenko and Raymond J Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[43] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[44] M Narasimha Murty and V Susheela Devi. *Pattern recognition: An algorithmic approach*. Springer Science & Business Media, 2011.

[45] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[46] Wen-Syan Li, Chris Clifton, and Shu-Yao Liu. Database integration using neural networks: Implementation and experiences. *Knowledge and Information Systems*, 2(1):73–96, 2000.

[47] Wei Zong, Feng Wu, Lap-Keung Chu, and Domenic Sculli. Identification of approximately duplicate material records in ERP systems. *Enterprise Information Systems*, (ahead-of-print):1–18, 2015.

[48] Jingshan Huang, Jiangbo Dang, José M Vidal, and Michael N Huhns. Ontology matching using an artificial neural network to learn weights. In *Proc. IJCAI Workshop on Semantic Web for Collaborative Knowledge Acquisition (SWeCKa-07), India*, 2007.

[49] Oktie Hassanzadeh and Mariano P Consens. Linked Movie Data Base. In *LDOW*, 2009.

[50] Yoav Goldberg and Michael Elhadad. splitSVM: fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 237–240. Association for Computational Linguistics, 2008.

[51] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear SVM. *The Journal of Machine Learning Research*, 11:1471–1490, 2010.

[52] Mohamad H Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[53] Hema Raghavan and James Allan. Using soundex codes for indexing names in ASR documents. In *Proceedings of the Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval at HLT-NAACL 2004*, pages 22–27. Association for Computational Linguistics, 2004.