



المدرسة الوطنية المتعددة التقنيات  
Ecole Nationale Polytechnique



Departement : Génie Industriel

## Graduation Thesis

To obtain the Industrial Engineering diploma

Option : Data Science and Artificial Intelligence

---

# Entity Resolution in Large Bibliographic Databases : Case of Author Name Disambiguation

---

*Written by :*  
M. CHAALAL Mohamed Elmondhir

*Supervised by :*  
M. ARKI Oussama (ENP)  
M. DA SILVEIRA Marcos  
(LIST)

*Defended on September 21, 2023, before a jury composed of :*

Mme. BOUCHAFAA, Bahia	:	President	Professor	ENP
Mme. AIT BOUAZZA, Sofia	:	Examiner	MAA	ENP
M. ARKI Oussama	:	Promotor	MCB	ENP





المدرسة الوطنية المتعددة التقنيات  
Ecole Nationale Polytechnique



Departement : Génie Industriel

## Graduation Thesis

To obtain the Industrial Engineering diploma

Option : Data Science and Artificial Intelligence

---

# Entity Resolution in Large Bibliographic Databases : Case of Author Name Disambiguation

---

*Written by :*  
M. CHAALAL Mohamed Elmondhir

*Supervised by :*  
M. ARKI Oussama (ENP)  
M. DA SILVEIRA Marcos  
(LIST)

*Defended on September 21, 2023, before a jury composed of :*

Mme. BOUCHAFAA, Bahia	:	President	Professor	ENP
Mme. AIT BOUAZZA, Sofia	:	Examiner	MAA	ENP
M. ARKI Oussama	:	Promotor	MCB	ENP



المدرسة الوطنية المتعددة التقنيات  
Ecole Nationale Polytechnique



Département : Génie Industriel

**Mémoire de Projet de Fin d'Études**

Pour l'obtention du diplôme d'ingénieur en génie industriel

**Option : Data Science et Intelligence Artificielle**

---

# Résolution d'entités dans les grandes bases de données bibliographiques : Cas de la désambiguïsation des noms d'auteurs

---

*Réalisé par :*  
M. CHAALAL Mohamed Elmondhir

*Encadré par :*  
M. ARKI Oussama (ENP)  
M. DA SILVEIRA Marcos  
(LIST)

*Soutenu le 21 Septembre 2023, Devant le jury composé de :*

Mme. BOUCHAFAA, Bahia	:	President	Professor	ENP
Mme. AIT BOUAZZA, Sofia	:	Examiner	MAA	ENP
M. ARKI Oussama	:	Promotor	MCB	ENP

# A Thank You

“

*I would like to express my sincere gratitude to Mr. Da Silveira, for his unwavering support, wise counsel, and invaluable contributions throughout the completion of this master thesis. Your expertise and mentorship were instrumental in the success of this research.*

*I would also like to express my deep gratitude to Mr. Arki, Ms. Bouchaffa and Ms. Ait Bouazza, for their time, dedication, and meticulous review of this work. Your constructive feedback and commitment to academic excellence helped refine the quality of this work.*

*In addition, I would like to acknowledge the collaborative spirit and contributions of my work colleagues at LIST, Jader Martins and Ben Graffet. Your teamwork, brainstorming sessions and camaraderie created a stimulating environment for research and personal growth.*

*To all those mentioned here and to all the others who have supported me on this academic journey, I express my deepest gratitude. Your encouragement and help have been essential to the completion of this work. Thank you for being an essential part of my academic journey.*

”

- *Elmondhir*

# Dedicate

First of all, I thank Almighty Allah for giving me the courage and patience to bring this work to a successful conclusion.

To my parents, my guides and role models, no words could express the love and respect I have always had for you. Your encouragement, sacrifices and unwavering love have been my driving force throughout my academic career, and it is you who have made me the person I am today.

To my brother, the support and comfort you have given me have given me undeniable courage and strength.

To my sister I know very well how important my success is to you, you guided me through all my most crucial decisions. Your journey and your path were an example to follow and my source of inspiration.

To my sister, who was there for me from the beginning, who supported me when my challenges seemed insurmountable, I am deeply grateful for your advice and constant encouragement. Thank you for being my emotional pillar throughout my entire journey.

To my brother who was my companion, my confidant and my first encourager your presence and efforts played a huge role in my academic success thank you for being my source of resilience.

# ملخص

يهدف هذا العمل إلى اكتشاف تمييز أسماء الكتّاب في قواعد بيانات مكتبية ضخمة وشاملة. إنها تتعامل مع تحدٍ يتمثل في تحديد وتمييز الكتّاب بدقة، والذين يشتركون في أسماء مشوشة ضمن مجموعة بيانات واسعة ومتنوعة. تقترح النهج المقترح اتباع نموذج مكون من ست مراحل يشمل استخدام تقنيات التعلم الآلي وتحليل الشبكات لزيادة دقة عملية التمييز. كما يتم مناقشة الاعتبارات العملية لتنفيذ هذه الحلول في قواعد البيانات الكبيرة. هذا العمل يساهم في تحسين إدارة المعلومات الأكاديمية بشكل أكثر موثوقية وكفاءة.

## كلمات مفتاحية :

تمييز الكيانات, تمييز أسماء الكتّاب, التعلم الآلي, التصنيف, سبارك, (Spark), نيوفورج. (Neo4j).

## Résumé

Cette mémoire explore la désambiguïsation des noms d'auteurs dans de vastes bases de données bibliographiques. Elle aborde le défi d'identifier et de distinguer avec précision les auteurs partageant un nom ambigu au sein d'un ensemble de données vaste et diversifié.

L'approche proposée suit un modèle en six phases qui implique l'utilisation de techniques d'apprentissage automatique et d'analyse de réseau pour améliorer la précision de la désambiguïsation. Des considérations pratiques pour la mise en œuvre de ces solutions dans de grandes bases de données sont également abordées. Ce travail contribue à une gestion de l'information universitaire plus fiable et plus efficace.

**Mots clés :** Résolution d'Entités, Désambiguïsation des Noms d'Auteurs, Apprentissage Automatique, Classification, Spark, Neo4j.

# Abstract

This Master thesis explores author name disambiguation in Large extensive bibliographic databases. It tackles the challenge of accurately identifying and distinguishing authors who share an ambiguous name within a vast and diverse dataset.

The approach proposes following a 6 phases model involving the use of machine learning and network analysis techniques to improve disambiguation accuracy. Practical considerations for implementing these solutions in large databases are also discussed. This work contributes to more reliable and efficient scholarly information management.

---

**Keywords :**

Entity Resolution, Author Name Disambiguation, Machine learning, Classification, Spark, Neo4j.

---



# Table of contents

List of Tables

List of Figures

Code Listings

List of Abbreviations

<b>General Introduction</b>	<b>14</b>
<b>1 Preliminary Study</b>	<b>16</b>
1.1 Presenting Luxembourg Institute of Science & Technology	16
1.2 Responsible Data Science & Analytics Systems	17
1.3 Knowledge Sharing Platform	18
1.4 Knowledge Graphs: Bridging the Information Gap	19
1.5 Architecture of KSP Platform	19
1.6 Neo4j: Connecting the Dots	20
1.7 The Platform's Knowledge Graph	21
1.8 Entity Resolution Preliminaries	22
1.9 Different Application Fields of Entity Resolution	23
<b>2 State of the Art</b>	<b>24</b>
2.1 Entity Resolution in Bibliographical Databases	24
2.2 Challenges of Entity Resolution in Bibliographic Databases	25
2.3 Data science, Machine Learning and Other Fundamental Notions	25
2.3.1 Data Science	25
2.3.2 Machine Learning	26
2.3.3 Supervised Learning	26
2.3.4 Unsupervised Learning	33
2.4 Graph Clustering Techniques	36
<b>3 The Entity Resolution Framework</b>	<b>40</b>
3.1 Phase 1: Data Pre-processing	41
3.2 Phase 2: Blocking Scheme	41
3.2.1 Standard Blocking	42
3.2.2 Sorted Neighborhood Blocking	42

3.2.3	Q-gram Based blocking	43
3.2.4	Suffix Array Blocking	44
3.2.5	Block Filtering	45
3.2.6	Evaluation	45
3.3	Phase 3: Pairwise Comparison	46
3.3.1	Levenshtein Similarity	47
3.3.2	Longest Common Substring Similarity	47
3.3.3	Q-gram Based Similarity	48
3.3.4	Jaro-Winkler Similarity	49
3.3.5	Monge-Elkan Similarity	49
3.4	Phase 4: Classification	50
3.4.1	Probabilistic Classification	50
3.4.2	Rule Based Classification	50
3.4.3	Learning-Based Classification	51
3.5	Phase 5: Clustering as a Post-processing Step	52
3.6	Phase 6: Evaluation	53
<b>4</b>	<b>Proposed Solution</b>	<b>55</b>
4.1	Problem Definition	55
4.2	Data Comprehension	56
4.2.1	Data Collection	56
4.2.2	Data Analysis	56
4.3	Data Preparation	58
4.3.1	Data Cleaning	58
4.3.2	Data Transformations	60
4.3.3	Reference Dataset	62
4.4	Blocking Scheme	64
4.4.1	Evaluation	64
4.4.2	Block Refinement	64
4.5	Pairwise Comparison	65
4.6	Classification	67
4.7	Clustering	68
4.8	Model Deployment	69
4.9	Step-by-Step Workflow	72
	<b>General Conclusion</b>	<b>76</b>
	<b>References</b>	<b>78</b>
	<b>Appendix A</b>	<b>81</b>
	<b>Appendix B</b>	<b>88</b>

# List of Tables

Table 2.1 : Confusion Matrix as an evaluation metric	32
Table 4.1 : Blocking Phase results	64
Table 4.2 : Blocking Phase results (after block refinement)	65
Table 4.3 : Class distribution of pairs before sampling	66
Table 4.4 : Class distribution of pairs after sampling	66
Table 4.5 : Classification results	67
Table 4.6 : Classification results (after parameters tuning)	67
Table 4.7 : Clustering results	69

# List of Figures

Figure 1.1 : ESRIC's Knowledge sharing platform	18
Figure 1.2 : Key numbers of the platform	19
Figure 1.3 : Nodes and Edges are the building block of KGs	19
Figure 1.4 : High level design of the Platform [1]	20
Figure 1.5 : Graph databases hold the advantage in the case of connected data	21
Figure 1.6 : Knowledge Graph containing different entities and their relationships	21
Figure 1.7 : Example of duplication in customer purchase data	22
Figure 2.1 : Example of a support vector machines margins	29
Figure 2.2 : Example of a decision tree to classify potential purchasers	30
Figure 2.3 : Example of a Random forest	31
Figure 2.4 : Different types of linkages	35
Figure 2.5 : Example of (CC) clustering	37
Figure 2.6 : Nodes, Edges and the Adjacency Matrix	38
Figure 2.7 : Example of Louvain Algorithm steps	39
Figure 3.1 : A high level refrence model of an ER process [17]	40
Figure 3.2 : Result of standard blocking on Last Name	42
Figure 3.3 : Concept of the Sorted-Neighborhood Method [18]	43
Figure 3.4 : Example of Q-gram based blocking with Surnames as BKV [20]	44
Figure 3.5 : Example of suffix array based blocking with minimim length $l_{min} = 4$	44
Figure 3.6 : The Block Filtering Approach	45
Figure 3.7 : Example of applying CENTER to partition the subgraph of matches, to solve intransitivity	53
Figure 4.1 : For more information check: <a href="https://dev.elsevier.com/">https://dev.elsevier.com/</a>	56
Figure 4.2 : Percentage of missing values in some of the datasets	57
Figure 4.3 : YAKE workflow pipeline	59
Figure 4.4 : Example of rows in the refrence dataset	63
Figure 4.5 : The Gitlab repository containing the code	70
Figure 4.6 : The components of Apache Spark	70
Figure 4.7 : The timeline of the spark job execution	74
Figure 4.8 : Resultat de regroupement graphic	75
Figure 4.9 : Example of Intransitive closer in a graph (nodes a, and e)	81
Figure 4.10 : The difference between under and over fitting	82

Figure 4.11 : k-folds cross validation	83
Figure 4.12 : Example of the semantic distance between two analogies	84
Figure 4.13 : Architecture of skip vs CBOW	85
Figure 4.14 : Gradient Descent illustrated in a two dimensional example	86
Figure 4.15 : GridSearch parameter tuning	87
Figure 4.16 : Deduplicator's python modules	88

# Code Listings

Code 4.1	Function to preprocess the dataframe	59
Code 4.2	Function to apply data transformations	62
Code 4.3	Function to create embeddings	66
Code 4.4	Function to tune model's hyperparameters	67
Code 4.5	Function to read the .csv files	88
Code 4.6	Function to perform prefix array blocking	89
Code 4.7	Function to perform sorted neighborhood blocking	90
Code 4.8	Function to perform Block Filtering	91
Code 4.9	Function to apply similarity comparison	93
Code 4.10	Function to calculate pairwise similarities between candidate records	94
Code 4.11	Function to generate the pairs after blocking	95

# List of Abbreviations

<b>AND</b>	Author Name Disambiguation
<b>BERT</b>	Bidirectional Encoder Representations from Transformers
<b>BKV</b>	Blocking Key Value
<b>CC</b>	Connected Components
<b>ESRIC</b>	European Space Resources Innovation Centre
<b>ER</b>	Entity Resolution
<b>HAC</b>	Hierarchical Agglomerative Clustering
<b>KG</b>	Knowledge Graph
<b>KSP</b>	Knowledge Sharing Platform
<b>LIST</b>	Luxembourg Institute of Science and Technology
<b>NB</b>	Naive Bayes
<b>PC</b>	Pairs Completeness
<b>PQ</b>	Pairs Quality
<b>RF</b>	Random Forest
<b>RR</b>	Reduction Ratio
<b>SNM</b>	Sorted Neighborhood Method
<b>SVM</b>	Support Vector Machine
<b>UDF</b>	User Defined Function
<b>YAKE</b>	Yet Another Keyword Extractor

# General Introduction

With the proliferation of digital devices and the internet, we are generating data at an unprecedented scale. According to a report by IBM [2], we generate 2.5 quintillion bytes of data every day, and this number is only expected to increase as more devices and systems come online. This data comes from a wide variety of sources, such as social media, sensors, transactions, and more. As a result, many organizations are struggling to manage this large and diverse data, and are looking for ways to extract value from it. Among the numerous challenges organizations face, one prominent issue is dealing with duplicate or inconsistent records in their datasets. For instance, a customer might have different names or addresses listed across multiple databases, or a patient might possess multiple medical records scattered across various hospitals. Such discrepancies can lead to inaccurate analytics, and flawed decision-making. To address these challenges, organizations need the ability to identify and link records that pertain to the same real-world entity, regardless of the variations present in the data.

This process is commonly known as Entity Resolution (ER), or alternatively referred to as record linkage or deduplication. As the volume of data generated and collected continues to escalate, ER has become increasingly crucial. While it is not a novel problem, ER has received considerable attention in domains like government agencies, healthcare, and finance [3]. In these sectors, accurately identifying entities is paramount to combat fraud, maintain precise records, and ensure public safety. However, in recent years, ER has gained prominence across various industries due to the exponential growth of data and the pressing need to link and match records effectively.

Entity resolution assumes particular significance in vast bibliographic databases. These repositories house millions of publication records and their respective authors. The primary objective of entity resolution in this context is to identify all publications attributed to a specific researcher, even if those publications are listed under slightly different names or affiliations. Unfortunately, author names can be highly ambiguous, with many authors sharing the same or similar names or using different variations of their names across different publications. Consequently, accurately identifying all of a researcher's publications becomes a challenging endeavor.

The Luxembourg Institute of Science and Technology (LIST), in collaboration with the European Space Agency (ESA), is currently undertaking a crucial project to construct a comprehensive space resources knowledge graph encompassing scientific papers and citations. This endeavor entails collecting and analyzing vast amounts of data from multiple sources, which often refer



to the same entity using different identifiers or representations. By implementing entity resolution techniques, LIST can effectively identify and link records associated with the same entity across multiple sources. This process enhances the accuracy and completeness of the knowledge graph, enabling researchers to derive more precise conclusions and make better-informed decisions regarding space resource exploration and utilization.

# Chapter 1

## Preliminary Study

In this chapter, our primary focus will be directed towards the host company, the Luxembourg Institute of Science and Technology (LIST). We will delve into the organizational structure of LIST, exploring its various groups and units, each contributing to the institution's diverse portfolio of research and innovation. Furthermore, we will shine a spotlight on one of LIST's most prominent endeavors: its collaboration with the Luxembourg Space Agency and the European Space Agency to develop a centralized space knowledge sharing platform. A platform aims to revolutionize how researchers access and query scientific documents.

### 1.1 Presenting Luxembourg Institute of Science & Technology

The Luxembourg Institute of Science and Technology (LIST) is a mission-driven Research and Technology Organization (RTO) that develops competitive and market-oriented product/service prototypes for public and private stakeholders. With its 600 employees, 75% of whom are researchers or innovation experts from all around the world, LIST is active in the fields of informatics, materials and environment and works across the entire innovation chain: fundamental/applied research, incubation and transfer of technologies.

#### **Environment**

The Environmental Research and Innovation (ERIN) department, made up of 200 life science, environmental science and information technology researchers and engineers, provides the interdisciplinary knowledge, expertise and technologies to lead solutions including the major environmental challenges facing society, such as climate change mitigation, ecosystem resilience, sustainable energy systems, efficient use of renewable resources, and environmental pollution prevention and control. The department relies on two cutting-edge platforms, the Biotechnologies and Environmental Analytics Platform and the Observatory for the Climate and the Environment, and the GreenTech Innovation Centre (GTIC): a one-stop-shop for the complete development of bio-based products and processes.

## **Informatics**

The IT for Innovative Services (ITIS) department, with its 100 researchers and engineers, focuses on the digital transformation of operations in organizations with traditional environments and digital ecosystems, with the aim of improving their performance and innovation capacity. The common thread throughout ITIS is to develop the most efficient use of big data to ensure the most appropriate decision-making processes. The department relies on the Data Analytics Platform: a hybrid infrastructure covering the entire range of data analytics activities. The platform is based on three pillars: a high-performance computing (HPC) infrastructure, a cognitive analytics pillar and an interactive visualization wall (Viswall).

## **Materials**

Through its research into advanced materials and processes, the “Materials Research and Technology” (MRT) department, with its 200 researchers and engineers, contributes to the emergence of enabling technologies that underpin the innovation processes of local and international industry. MRT’s activities hinge on four thematic pillars: nanomaterials and nanotechnology, scientific instrumentation and process technology, structural composites, and functional polymers. The department also includes four high-tech platforms, focusing on composites, prototyping, characterization and testing. These platforms serve both LIST research staff, and other stakeholders in Luxembourg.

## **Space**

Unique of its kind, the European Space Resources Innovation Centre (ESRIC) is a joint initiative between LIST, the Luxembourg Space Agency (LSA) and the European Space Agency (ESA). As a new department within LIST, it aims to become the internationally recognised centre of expertise for scientific, technical, business and economic aspects related to the use of space resources for human and robotic exploration, as well as for a future in-space economy. At its core, ESRIC provides access to unique research facilities and expertise, enabling ground-based R&D around prospecting, extracting, processing, storing, supplying and using space resources.

## **1.2 Responsible Data Science & Analytics Systems**

The Responsible Data Science & Analytics Systems Unit, within list supports the digital transformation of our society and our economy by focusing on the interplay between people, data, computers and the physical world. RDSA carries out impact-driven research by combining the power of computers and human capabilities to take better, faster, more robust, fair and trustworthy decisions in our complex and changing world. By studying Cyber-Physical-Social Systems (CPSS) to imagine, design, test and develop the next generation of technologies, where computers and humans are smoothly and fairly working together to tackle challenging issues.

The unit is composed of around 40 scientists, engineers, post-docs and PhD candidates with complementary scientific and technological expertise, structured in 3 groups.

- **Trustworthy AI:** Focuses on understanding AI in terms that connect to human-established concepts and expertise.
- **Visualisation & interaction:** Focuses on novel visualisation & interactive methods and technologies that support human-in-the-loop decision-making and collaboration, and improve human immersion, engagement and involvement in tasks.
- **Human Modelling:** Focuses on human-aware cognitive IT technologies combining human modelling with AI to support learning and decision making, perform machine-reasoning and implementing explainable AI.

### 1.3 Knowledge Sharing Platform

In May 2022, ESRIC launched its knowledge sharing platform for the space resources community. This data visualisation tool integrates about 1,000 scientific publications, complemented by news, patents, books, press articles, legislative documents, or social media posts. It is freely accessible to the space resources community on a registration basis.

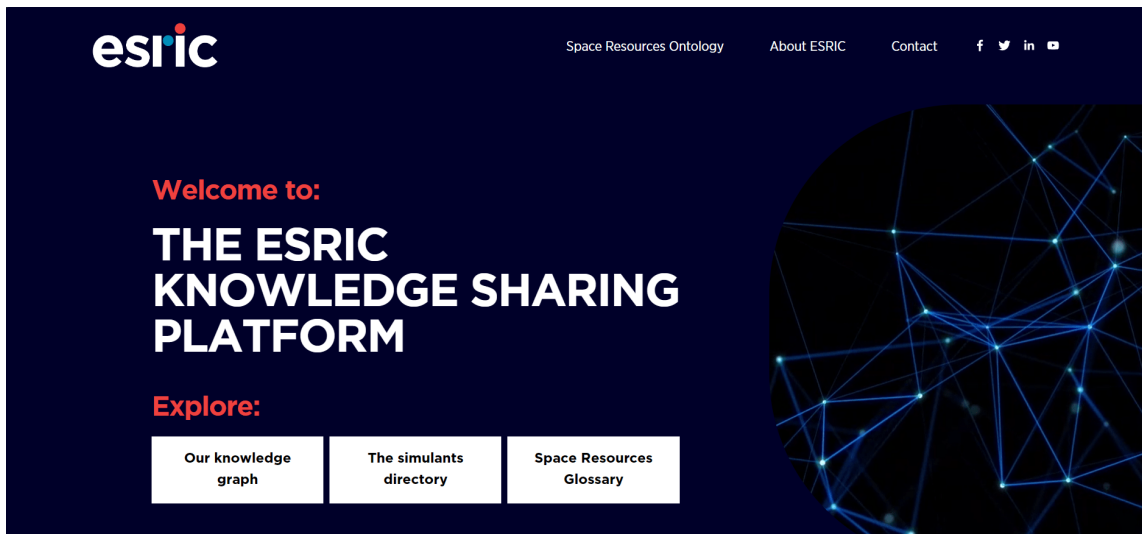


Figure 1.1: ESRIC’s Knowledge sharing platform

The knowledge-sharing platform responds to the motivations and expectations of the community and follows a survey conducted between April and July 2021 which revealed the great diversity of the space sector, as well as its economic potential, linked to terrestrial sectors such as mining, energy or construction.

## Key numbers:



Figure 1.2: Key numbers of the platform

## 1.4 Knowledge Graphs: Bridging the Information Gap

Knowledge Graphs (KGs) are a sophisticated data structure designed to capture, represent, and harness complex relationships within information ecosystems. They enable organizations to represent complex relationships within data, offering a dynamic framework for understanding and querying interconnected information.

At the heart of a KG are nodes and edges. Nodes represent entities or concepts, such as people, places, or things, while edges symbolize the relationships between these entities. Each node and edge can hold attributes or metadata, enriching the graph with additional context.

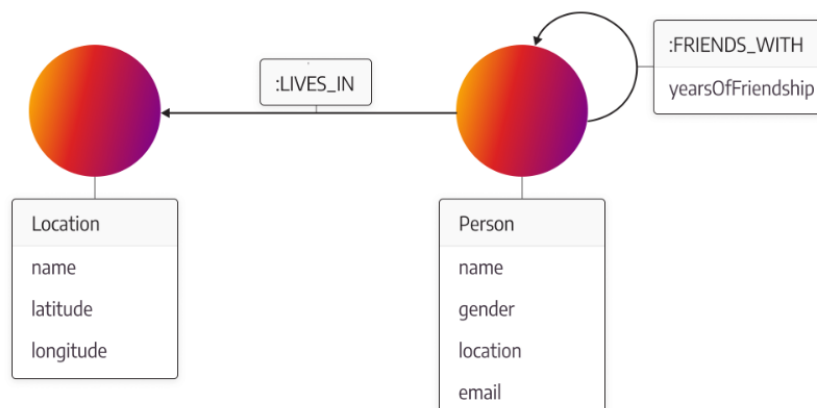


Figure 1.3: Nodes and Edges are the building block of KGs

As KGs grow in complexity and size, scalability and performance become critical considerations. Distributed database systems, graph databases, and optimized storage solutions are employed to ensure fast and efficient access to graph data.

## 1.5 Architecture of KSP Platform

The platform's architecture was built with the aim to respond to the space communities needs, most notably:

- Extracting Pertinent information from reliable sources

- Formalize this information and bring out its semantic meaning in order to transform it into into knowledge
- Exploit this knowledge through a range of appropriate services

To this end, the architects of the platforms focused on using Semantic web technologies that offer standirzed ways of describing the semantics of domain concepts using ontologies. They explain at further details their choices in their paper [1].

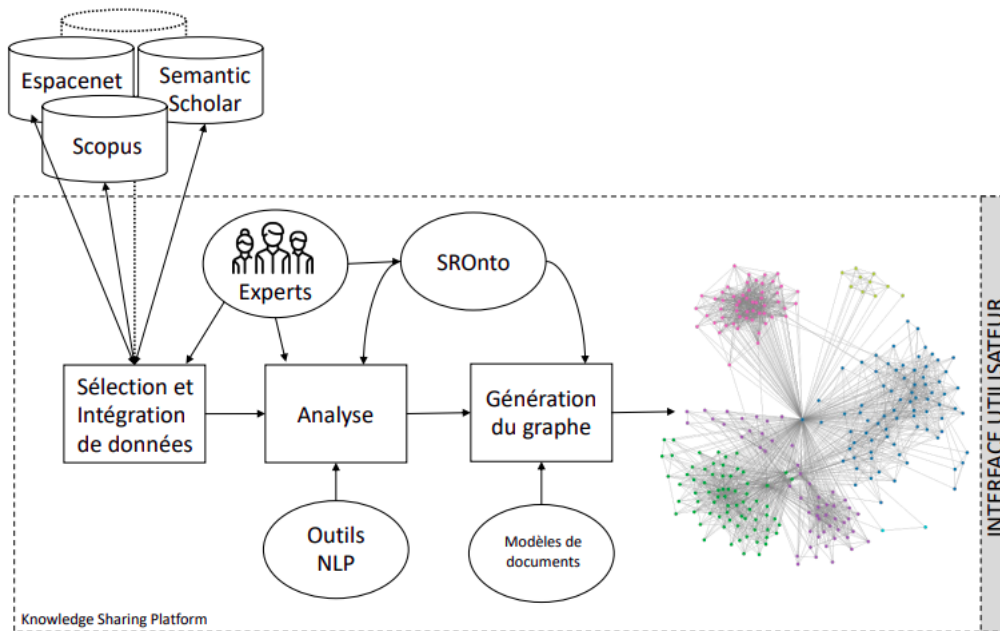


Figure 1.4: High level design of the Platform [1]

## 1.6 Neo4j: Connecting the Dots

In the intricate landscape of Knowledge Graphs, selecting the right database management system is paramount. Among the array of choices, Neo4j stands out as the preferred choice for organizations and researchers aiming to harness the full potential of Knowledge Graphs

Graph databases, like Neo4j, depart from the traditional relational database model by focusing on the relationships between data points as first-class citizens. In a graph database, data is organized into nodes, which represent entities, and relationships, which represent connections between entities. This structure enables Neo4j to excel in scenarios where the relationships between data points are as important as the data itself.

This means that Graph databases far outperform traditional databases when it comes to querying highly connected data. The latter necessitates multiple extensive join operations, while in the case of graph databases, traversing the graph's path is all that's needed.

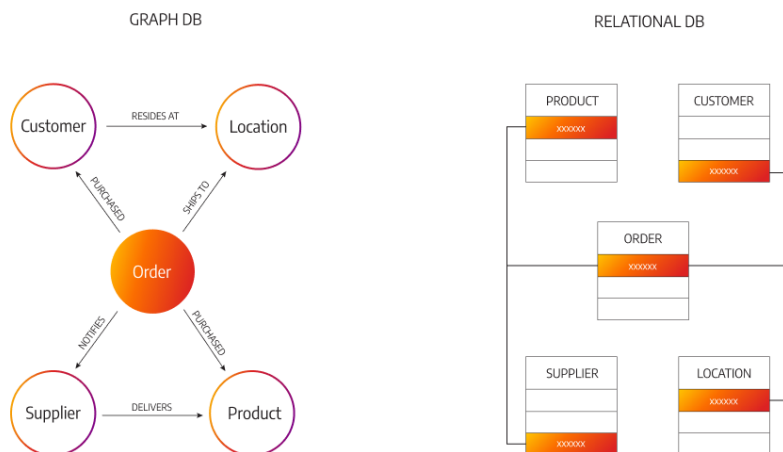


Figure 1.5: Graph databases hold the advantage in the case of connected data

## 1.7 The Platform’s Knowledge Graph

The ESRIC Knowledge Sharing Platform utilizes a robust knowledge graph that encompasses a vast number of nodes and relationships. This knowledge graph encompasses a rich collection of information, including hundreds of thousands of nodes representing various entities such as documents, authors, institutions, and the intricate relationships between them, such as affiliations and authorship.

The objective of this work is to contribute to the development of methods that enhance the quality and accuracy of the knowledge graph’s content. This involves tackling challenges such as identifying and removing duplicate records and resolving entities with similar or ambiguous representations. By addressing these challenges, the aim is to optimize the knowledge graph’s capability to effectively respond to user queries and provide reliable and comprehensive information.

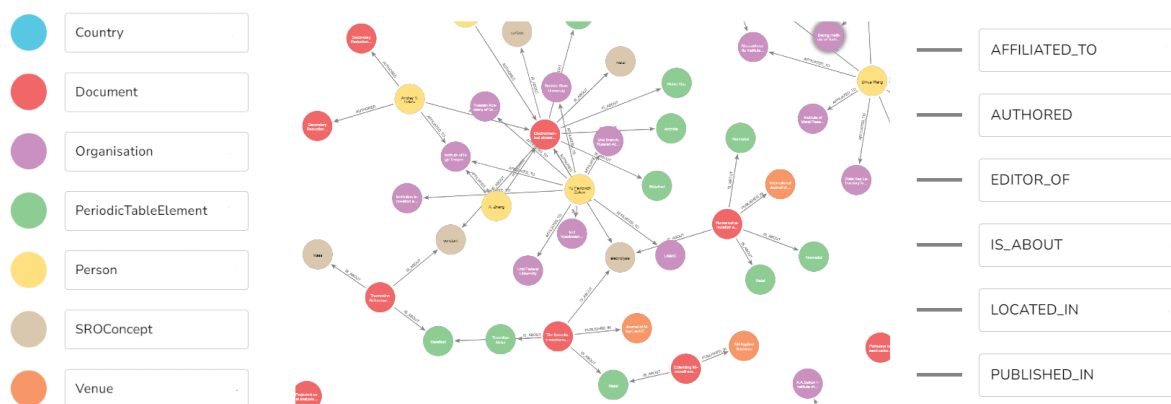


Figure 1.6: Knowledge Graph containing different entities and their relationships

## 1.8 Entity Resolution Preliminaries

Entity resolution research has a long history, dating back to the mid-20th century [4], and has garnered significant attention over the years. Depending on the specific application domain, entity resolution may be referred to by different names, such as record linkage [5] [6], de-duplication [7], or even the merge/perge problem [8].

At first glance, entity resolution may appear straightforward since records with similar attributes are likely to pertain to the same entity. However, as we will explain later, various challenges emerge during the process. To illustrate the concept of entity resolution further, let's consider the following example:

Imagine we are an online retail business currently developing a new recommendation engine that utilizes customer information to offer personalized product recommendations. Our dataset comprises a purchase log where customer details are represented by attributes like names, addresses, emails, and phone numbers. Each customer interaction is logged with a unique ID.

As we have many returning customers, effectively building a recommender system requires aggregating the order history for each individual customer. However, this task becomes challenging when customer data exhibits inconsistencies across different rows associated with the same customer's purchases. Let's examine a table to illustrate this scenario:

ID	Name	Phone	Address	Email	Items Purchased
148	Chris J. Smith	334-555-0178	111 Crackow St, Washington DC	c.jamesmith@gmail.com	1124, 1789, 2335
201	Anna Walker	202-555-5475	342 Hope Blvd, Alabama	walker.an2@gmail.com	1786, 2748
207	C.J. Smith	334-555-0178	111 Crackow St, Washington DC	jsmith.c@yahoo.com	2134
298	Annabelle Walker			anna.walker@yahoo.com	1786
324	Carl Smith	275-266-7541	4141 Baker Avenue, Texas	carlos_smith@outlook.com	1124

Figure 1.7: Example of duplication in customer purchase data

Now, the question arises: how can we aggregate data for each unique customer rather than just their individual purchase data? Which functions should we employ? Moreover, how do we handle slight variations in attributes, such as names that might include nicknames, initials, or misspellings?

This is precisely where entity resolution plays a crucial role. By leveraging existing customer attributes and extracting relevant features, a robust entity resolution process can identify, for example, that purchase records "148" and "207" likely correspond to the same individual, namely "Chris J. Smith."



## 1.9 Different Application Fields of Entity Resolution

Before moving into the challenges associated with this task, it is important to highlight the broad applicability of ER across various sectors and fields:

- In healthcare, ER is instrumental in linking patient records from diverse sources, creating comprehensive profiles that aid in informed decision-making and enhanced patient care. By identifying and combining records across multiple healthcare facilities, ER ensures a holistic view of a patient's medical history, resulting in improved accuracy and better healthcare outcomes.
- ER plays a vital role in the finance sector by identifying and linking financial transactions associated with the same entity. This capability helps detect and prevent fraudulent activities and money laundering. By identifying suspicious transactions across multiple accounts, ER assists in investigating potential criminal organizations and safeguarding financial systems.
- In the insurance industry, ER enables the linkage of policyholder records sourced from various channels, including claims data and customer interactions. This linkage enables insurers to develop comprehensive profiles of policyholders, facilitating more accurate risk assessments. By identifying patterns across multiple claims, insurers can better understand policyholders' risk levels and make informed decisions.

## Chapter 2

# State of the Art

In this chapter, we will explore the fundamental theoretical concepts that lay the groundwork for understanding the problem at hand and the proposed solution. Our goal is to provide readers with a clear understanding of these concepts, which will be further explored in the remainder of this work.

To begin, we will establish an understanding of entity resolution usage in bibliographic databases. We will discuss the specific challenges that arise when performing entity resolution tasks, particularly in our context. Additionally, we will delve into other fundamental notions, such as Data Science and Machine Learning, which have become synonymous with the task of entity resolution.

### 2.1 Entity Resolution in Bibliographical Databases

Bibliographic databases like PubMed, Scopus, and Web of Science store a vast number of records representing scholarly articles and research publications. However, errors, inconsistent data entry, and citation variations can lead to duplicate or similar records in these databases. The process of entity resolution involves identifying and linking records that refer to the same publication, author, or research entity to ensure data accuracy.

Entity resolution in bibliographic databases is an ongoing and iterative process. It involves regularly reviewing and improving algorithms and techniques as new publications are added and existing records are updated.

Accurate entity resolution is crucial for researchers, scholars, and information retrieval systems. It provides reliable citation data for research, facilitates citation analysis, supports evaluation based on citations, and contributes to the discovery and sharing of scholarly knowledge.

## 2.2 Challenges of Entity Resolution in Bibliographic Databases

Entity resolution in bibliographic databases comes with several challenges that researchers and data professionals encounter. These challenges stem from the unique characteristics of the data and the complexities of the domain. Here are some key challenges:

- **Data Quality:** Inconsistent and incomplete information in bibliographic databases can hinder accurate entity resolution. Missing data, variations in formatting, and errors in data entry are common issues. Robust techniques like data cleaning, imputation, and leveraging external sources are needed to address data quality problems.
- **Integration of Multiple Data Sources:** Bibliographic databases gather data from diverse sources, including different publishers and disciplines. Integrating data from heterogeneous sources and resolving entities across them can be challenging due to variations in data formats, inconsistent naming conventions, and disparate data quality.
- **Scalability:** Managing the large volume of records in bibliographic databases poses computational and storage challenges. Efficient algorithms and indexing techniques are necessary to handle the scale of data and ensure timely resolution of entities.
- **Handling Different Entity Types:** In the research domain, there are different types of entities, like authors, publications, and institutions. Each type requires specific techniques to match and link related records accurately. For example, matching authors might involve dealing with variations in their names and affiliations, while publications could be matched based on titles and dates.

## 2.3 Data science, Machine Learning and Other Fundamental Notions

In this section, we will explore some concepts related to entity resolution, including the domains of data science and machine learning. We will also delve into popular techniques used in machine learning, such as supervised and unsupervised learning.

### 2.3.1 Data Science

Data science is an interdisciplinary field that involves extracting insights, knowledge, and actionable information from structured and unstructured data. It combines elements of statistics, mathematics, computer science, and domain expertise to analyze large and complex datasets and uncover patterns, trends, and correlations that can be used for decision-making and problem-solving.

Data science encompasses a range of activities, including data collection, data cleaning and pre-processing, exploratory data analysis, feature engineering, predictive modeling, and evaluation

of models. It also involves interpreting and communicating the findings to stakeholders in a clear and understandable manner.

### 2.3.2 Machine Learning

Machine learning, as a subfield of data science, encompasses the utilization of mathematical and statistical methods to construct algorithms and models that possess the ability to autonomously learn from data and subsequently make predictions or informed decisions. This field relies on the examination and analysis of datasets to identify underlying patterns and relationships, which are then leveraged to train machine learning models.

These trained models often aim to emulate specific facets of human cognition, including perception, reasoning, and decision-making. While machine learning does not strive to replicate the entirety of human intelligence, it endeavors to mimic certain aspects of human-like behavior and performance

Machine learning encompasses various approaches, in this work, we will focus on two fundamental categories: supervised learning and unsupervised learning. Each category has its own distinct objectives and techniques, which we will explore in the following sections.

### 2.3.3 Supervised Learning

Supervised learning is a machine learning paradigm that involves training models using labeled data. Labeled data consists of input features along with their corresponding target labels. The main objective of supervised learning is to learn a mapping function that can accurately predict the labels of unseen data based on their input features.

In the context of supervised learning, we consider a training dataset comprising  $N$  instances, denoted as  $(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)$ . Here,  $X_i$  represents the input feature vector of size  $d$ , and  $y_i$  represents the corresponding target label associated with that input instance. The underlying learning algorithm seeks to find a function  $f : X \rightarrow Y$ , where  $X$  denotes the input space and  $Y$  represents the output space.

The function  $f$  is trained to minimize a cost function, which quantifies the prediction error across all instances in the dataset. This cost function is determined by employing a defined loss function that measures the discrepancy between predicted and actual labels. In supervised learning, there are two main categories of tasks: supervised classification and supervised regression.

**Regression:**

For regression tasks, we aim to learn a function  $f(X; \theta)$  that can predict a continuous target value. Here,  $\theta$  represents the model's parameters that need to be learned. The prediction  $\hat{y}$  for a given instance  $X$  is computed as:

$$\hat{y} = f(X; \theta) \quad (2.1)$$

To quantify the discrepancy between the predicted values  $\hat{y}$  and the true target labels  $y$ , we typically use a loss function. One commonly used loss function is the mean squared error (MSE), defined as:

$$L(\hat{y}, y) = (\hat{y} - y)^2 \quad (2.2)$$

The goal is to minimize the average loss over all instances in the training dataset, which is commonly known as the cost function:

$$MSE(\theta) = \frac{1}{N} \sum_i^N L(\hat{y}_i, y_i) \quad (2.3)$$

The model's parameters  $\theta$  are optimized to minimize the cost function using gradient-based optimization algorithms. The gradients of the cost function with respect to the parameters are computed, and the parameters are updated iteratively to reduce the loss.

**Classification:**

In classification tasks, the goal is to assign instances to different predefined classes based on the input features. That is, by learning a scoring function  $f(x; \theta)$  that maps the input features  $x$  to a predicted score or probability for each class. To obtain class probabilities, we can use the softmax function (or sigmoid in case of binary classification):

$$\hat{y}_j = \frac{e^{f_j}}{\sum_i^K e^{f_i}} \quad (2.4)$$

where  $\hat{y}_j$  represents the predicted probability for class  $j$ ,  $f_j$  is the score for class  $j$ , and the summation is performed over all classes  $k$ . The prediction  $\hat{y}$  for a given instance  $X$  is the class with the highest probability :

$$\hat{y} = \underset{j}{\operatorname{argmax}}(\hat{y}_j) \quad (2.5)$$

To quantify the discrepancy between the predicted probabilities  $\hat{y}$  and the true target labels  $y$ , we define a suitable loss function, denoted as  $L(\hat{y}, y)$ . Commonly used loss functions for classification tasks include the cross-entropy loss, defined as:

$$L(\hat{y}, y) = - \sum_k y_j \log(\hat{y}_j) \quad (2.6)$$

As with regression, the goal is to minimize the average loss over all instances in the training dataset, that is the cost function defined as:

$$J(\theta) = \frac{1}{N} \sum_i^N L(\hat{y}, y) \quad (2.7)$$

To optimize the cost function and learn the model's parameters  $\theta$ , various optimization algorithms can be used, such as gradient descent or its variants. The optimization process involves updating the parameters iteratively based on the gradients of the cost function with respect to

## Common Supervised Learning techniques

There exist various supervised learning algorithms for both classification and regression tasks, However for the sake of brevity we will only talk about common classification algorithms, as regression task, are not of high relevance to our work.

### Logistic Regression

Logistic regression is a statistical model used for binary classification problems. The model uses a sigmoid function, denoted as  $\sigma(z)$ , to transform a linear combination of the features into a probability value between 0 and 1. The sigmoid function is defined as:

$$\sigma(z) = 1/(1 + \exp(-z)) \quad \text{given: } z = w + wx + \dots + w_d x_d \quad (2.8)$$

where  $z$  is the linear combination of the feature vector  $x$  and the corresponding weight vector  $w$ . The logistic regression model computes the probability of an instance belonging to class 1 as:

$$P(Y = 1|x; w) = \sigma(z) \quad (2.9)$$

To estimate the model parameters (weights), logistic regression employs a technique called maximum likelihood estimation (MLE). The goal is to find the optimal values for the weight vector  $w$  that maximize the likelihood of observing the given dataset.

The likelihood function  $L(w)$  represents the probability of observing the given labels  $y$  (0 or 1) given the input features  $x$  and the parameters  $w$ . It can be defined as:

$$L(w) = \prod_{i=1}^N [\sigma(z_i)]^{y_i} [1 - \sigma(z_i)]^{(1-y_i)} \quad (2.10)$$

To maximize the likelihood, we aim to find the values of  $w$  and  $b$  that maximize  $L(w)$ . However, it is more convenient to minimize the negative log-likelihood (NLL) since it converts the product of probabilities into a sum of log-probabilities, which result into this cost function:

$$J(w) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\sigma(z_i)) + (1 - y_i) \log(1 - \sigma(z_i))] \quad (2.11)$$

## Support Vector Machine

The main goal of an SVM is to find an optimal hyperplane that can separate data points belonging to different classes with the maximum margin (see Fig below). This hyperplane acts as a decision boundary, allowing us to classify new, unseen instances, and is defined by the equation:

$$w^T x + b = 0 \quad (2.12)$$

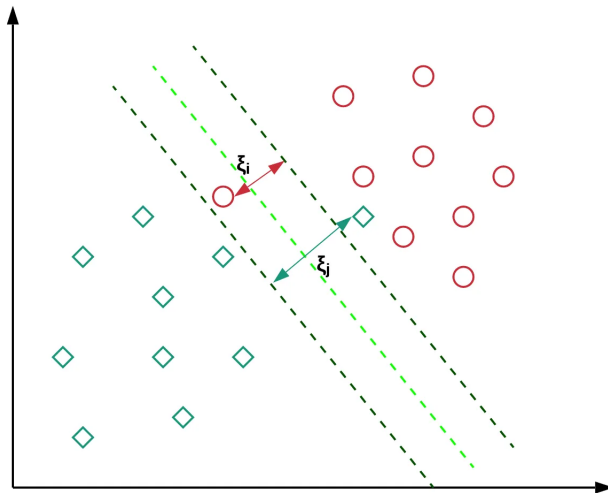


Figure 2.1: Example of a support vector machines margins

To maximize the margin, SVM seeks to solve the following optimization problem:

$$\begin{aligned} \min_{w,b} \frac{1}{2} \|w\|^2 \\ \text{s.c. } y_i(w^T x_i + b) \geq 1 - \epsilon \end{aligned} \quad (2.13)$$

where  $y_i = -1$  for the negative class and  $y_i = 1$  for the positive class. The constraint enforces that each data point is classified correctly with a margin of at least  $\epsilon$ . The parameter  $\epsilon$  allows for some tolerance in the margin to handle noisy or overlapping data points.

## Decision Trees

Decision trees are a popular and intuitive machine learning algorithm used for both classification and regression tasks. They are constructed using a tree-like structure, where each internal node

represents a decision based on a feature, and each leaf node represents a class label or a predicted value, as is shown in Fig below:

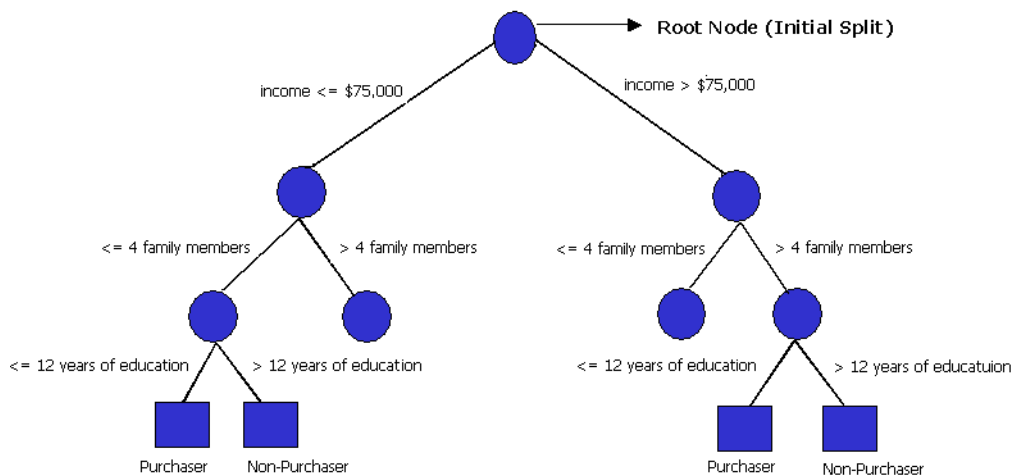


Figure 2.2: Example of a decision tree to classify potential purchasers

The decision tree can be constructed recursively as follows:

- **Selecting the best feature:** At each internal node of the tree, a feature  $f$  is selected based on a measure of impurity, such as Gini impurity or entropy. The feature  $f$  is chosen to maximize the reduction in impurity across the child nodes.
- **Splitting the data:** The selected feature  $f$  is used to partition the data into different subsets based on the feature values. Let  $v$  represent a specific value or threshold for feature  $f$ . The data is split into two branches based on the condition  $x_i, f \leq v$  for one branch and  $x_i, f > v$  for the other branch.
- **Recursion on child nodes:** The above steps are recursively applied to each child node until a stopping criterion is met. This criterion can be a maximum depth limit, a minimum number of samples per leaf, or a minimum decrease in impurity.
- **Assigning labels:** Once the recursion ends and we reach a leaf node, the leaf node is assigned a class label based on the majority class in the corresponding subset of samples.

The Gini impurity is a commonly used cost function in decision tree classification. It quantifies the impurity or uncertainty of a node in terms of class labels. It is defined as the probability of misclassifying a randomly chosen element in the node if it were randomly labeled according to the class distribution of the node  $dist(p_c)$ , it can be calculated as:

$$G(N) = 1 - \sum_{c=1}^C p_c^2 \quad (2.14)$$

## Random Forrest & Ensemble Methods

Ensemble methods are machine learning techniques that combine multiple models to make predictions. The idea behind ensemble methods is to leverage the diversity and collective wisdom



of multiple models to improve the overall performance and robustness.

In Random Forest, an ensemble of decision trees is constructed. Let's denote the ensemble as  $F$ , and each decision tree in the ensemble as  $f_i$ , where  $i$  ranges from 1 to the total number of trees in the forest. Each decision tree is built using a subset of the training data. Given a training dataset  $X$ , where each instance is denoted as  $x_i$ , and the corresponding target variable values are  $y_i$ , the Random Forest algorithm proceeds as follows:

- For each tree  $f_i$  in the ensemble  $F$ : a. Randomly select a subset of the training data with replacement, denoted as  $X_i$ . This is known as bootstrap sampling. b. Construct the decision tree  $f_i$  using  $X_i$ , where the tree is recursively built by selecting the best split at each node based on a selected criterion, such as Gini impurity or information gain.
- Aggregate the predictions of all trees in the ensemble to obtain the final prediction. For classification tasks, this can be done through majority voting, where the class with the most votes is selected as the predicted class. For regression tasks, the predictions of all trees can be averaged.

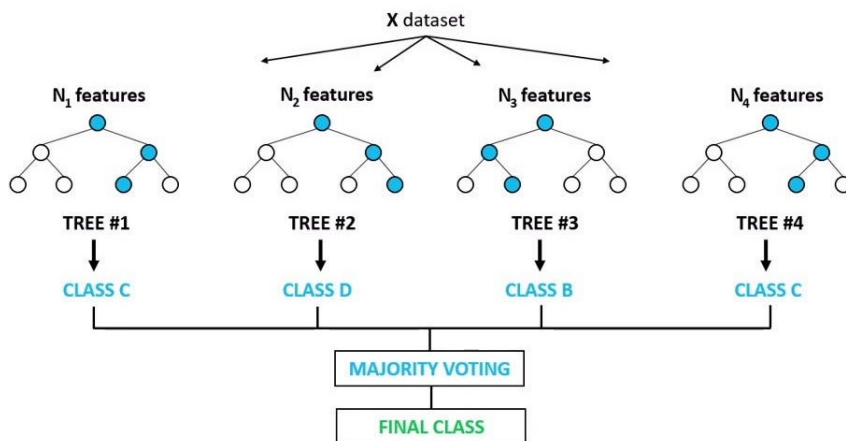


Figure 2.3: Example of a Random forest

The Random Forest algorithm provides several advantages. It helps mitigate overfitting by using random subsets of the data for each tree and combining their predictions. Additionally, it offers robustness to noisy data and can handle high-dimensional feature spaces effectively.

Ensemble methods, including Random Forest, improve the overall performance by leveraging the diversity among individual models. By combining multiple models, the ensemble can capture different aspects of the data and reduce the variance in predictions.

## Evaluation Metrics for Supervised Learning

There exists several evaluation metrics for supervised learning tasks, we will talk about the one used to assess the performance of classification models.

	Predicted Negatives	Predicted Positives
Actual Negatives	TP	FP
Actual Positives	FN	TN

Table 2.1: Confusion Matrix as an evaluation metric

For simplicity let's assume we have a binary classification problem with two classes, denoted as positive (P) and negative (N). Each prediction of the model will fall into one of the four categories.

- True Positives (TP): The number of instances that are correctly predicted as positive
- True Negatives (TN): The number of instances that are correctly predicted as negative
- False Positives (FP): The number of instances that are incorrectly predicted as positive, but are actually negative.
- False Negatives (FN): The number of instances that are incorrectly predicted as negative, but are actually positive.

Based on this, we can define the following metrics:

- Accuracy: measures the proportion of correctly classified samples out of the total number of samples.

$$Acc = \frac{TP + TN}{FP}$$

- Precision: measures the proportion of true positive predictions (correctly predicted positive samples) out of the total predicted positive samples.

$$Acc = \frac{TP + TN}{FP}$$

- Recall: measures the proportion of true positive predictions (correctly predicted positive samples) out of the total actual positive samples.

$$Acc = \frac{TP + TN}{FP}$$

- F-1 score: represents the harmonic mean of precision and recall, providing a balanced measure of a model's performance.

$$Acc = \frac{TP + TN}{FP}$$

- Confusion matrix: while not a scalar metric, the confusion matrix is a tabulation of predicted classes against actual classes, which helps visualize the performance of a classification model, it can be represented as:

### 2.3.4 Unsupervised Learning

Unsupervised learning is a branch of machine learning where the task involves discovering patterns, structures, or relationships in unlabeled data. Unlike supervised learning, unsupervised learning does not rely on predefined target labels or explicit feedback.

Unsupervised learning algorithms typically fall into two main categories: clustering and dimensionality reduction.

#### Clustering

The goal clustering algorithms is to group similar instances together based on their patterns or similarities, that is to identify distinct clusters in the data, where instances within the same cluster are more similar to each other than those in different clusters. Let's denote a dataset with N instances as  $(x_1, x_2, \dots, x_N)$ , where each  $x_i$  represents a feature vector.

A clustering algorithm aims to partition the dataset into K clusters, denoted as  $C_1, C_2, \dots, C_k$ , where each  $C_j$  represents a cluster. The objective of clustering is to find an assignment function, denoted as  $g(x)$ , that assigns each instance  $x$  to a cluster  $C_j$ .

$$g(x) : x \rightarrow C_j \quad (2.15)$$

The choice of the assignment function and the objective function of the clustering algorithm depends on the specific algorithm employed. Further details regarding these aspects will be discussed in the subsequent section.

#### Dimensionality Reduction

Dimensionality reduction techniques aim to reduce the number of input features while preserving important information. These techniques help overcome the difficulties encountered with multi-dimensional data, improve computational efficiency, and facilitate data visualization and analysis.

Let's consider a dataset with N instances, denoted as  $(x_1, x_2, \dots, x_n)$ , where each  $x_i$  represents a feature vector of dimension r. The goal of dimensionality reduction is to find a transformation function  $f(x)$  that maps the high-dimensional feature vectors  $x$  to a lower-dimensional representation  $z$ .

$$f(x) : x \rightarrow z \quad x \in R^r, z \in R^d \quad (2.16)$$

The transformed lower-dimensional representation  $z$  typically has a reduced number of dimensions, denoted by d, where  $d \ll r$ . The objective of dimensionality reduction is to minimize the information loss or reconstruction error during the reduction process. This can be achieved by optimizing a specific criterion or objective function, which varies depending on the technique used.

## Common Unsupervised Learning Techniques

There are various unsupervised algorithms for both clustering and dimensionality reduction. However, for the sake of brevity, we will focus solely on discussing common methods used for clustering, as dimensionality reduction is beyond the scope of this work.

### K-Means

In the K-Means algorithm, each cluster is represented by a centroid  $\mu_j$ . The objective is to minimize the sum of squared distances between instances and their assigned centroids. The objective function for K-Means is given as:

$$J(\mu, g) = \sum_i \|x_i - \mu_j\|^2 \quad (2.17)$$

Where  $\mu$  represents the centroids,  $g$  represents the assignment function, and  $\|\cdot\|$  represents the Euclidean distance. The K-Means algorithm proceeds iteratively. It starts by initializing  $K$  centroids randomly or using a heuristic. Then, in each iteration, it performs two steps:

- Assignment step: each data point  $x_i$  is assigned to the nearest centroid  $u_j$  based on the Euclidean distance. the cluster assignment variable is then defined as:

$$r_{ij} = \begin{cases} 1 & \text{if } x_i \text{ is assigned to cluster } j \\ 0 & \text{otherwise} \end{cases} \quad (2.18)$$

- Update step: The centroids are updated by calculating the mean of the data points assigned to each cluster. The updated centroid  $u_j$  is computed as:

$$u_j = \frac{1}{\sum_{i=1}^N r_{ij}} \sum_{i=1}^N r_{ij} x_i \quad (2.19)$$

### Agglomerative Clustering

Hierarchical Agglomerative Clustering (HAC) is a bottom-up hierarchical clustering algorithm that iteratively merges similar data points or clusters to form a hierarchical structure of clusters. It does not require a predefined number of clusters and can be summarized as follows:

- Initialization: Start with  $N$  individual clusters, each containing a single data point from the dataset  $C_i = \{x_i\}$  for  $i = 1, 2, \dots, N$
- Similarity/Dissimilarity Calculation: Compute the similarity or dissimilarity between clusters using a distance metric, such as the Euclidean distance  $d$ .

$$D_{ij} = d(C_i, C_j) \text{ for } i, j = 1, 2, \dots, N$$

- Merge Similar Clusters: Iterate through the clusters and merge the two most similar clusters based on a linkage criterion. The linkage criterion determines the distance or similarity between clusters (see fig below). Single, average and complete linkage are common criteria.

For Single Linkage:  $D(C_i, C_j) = \min\{D_{kl}\}$ , for all  $k \in C_i, l \in C_j$

For Complete Linkage:  $D(C_i, C_j) = \max\{D_{kl}\}$ , for all  $k \in C_i, l \in C_j$

For Complete Linkage:  $D(C_i, C_j) = avg\{D_{kl}\}$ , for all  $k \in C_i, l \in C_j$

- Repeat and update: Continue merging the most similar clusters and updating the similarity/dissimilarity matrix until a stopping criterion is met. This criterion can be a desired number of clusters or a specific threshold for similarity/dissimilarity.

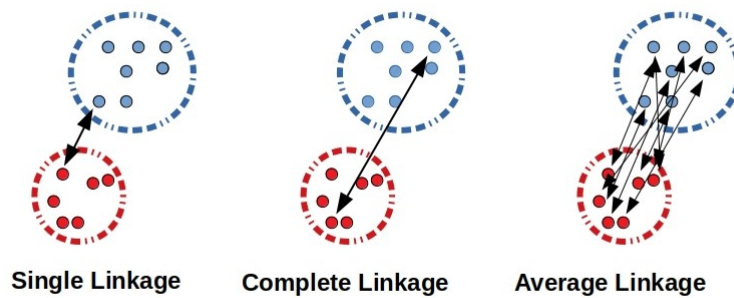


Figure 2.4: Different types of linkages

## Evaluation Metrics

Evaluation metrics for clustering assess the quality and performance of clustering algorithms, can be divided into two categories: intrinsic and extrinsic measures.

Intrinsic measures evaluate the quality of the clustering based solely on the clustering results themselves, without reference to external information. Some intrinsic evaluation metrics are:

- Silhouette Coefficient: measures the quality of a clustering result by computing the average silhouette coefficient for each data point. For a data point  $i$ , the silhouette coefficient is given by:

$$SC(i) = \frac{b(i) - a(i)}{\max}$$

where  $a(i)$  is the average distance between  $i$  and other data points in the same cluster, and  $b(i)$  is the average distance between  $i$  and data points in the nearest neighboring cluster. The overall Silhouette Coefficient is the average of  $SC(i)$  for all data points.

- Dunn Index (DI): is calculated as the ratio between the minimum inter-cluster distance and the maximum intra-cluster distance. Mathematically, the Dunn Index is defined as:

$$D = \frac{\min_{i \neq j} d(i, j)}{\max_k d_{intra}(k)}$$

where  $d(i,j)$  represents the distance between clusters  $i$  and  $j$ , and  $d_{intra}(k)$  represents the intra-cluster distance for cluster  $k$ . The inter-cluster distance is the minimum distance between any two clusters, and the intra-cluster distance is the maximum distance within each cluster.

Extrinsic measures, on the other hand, assess the quality of clustering by comparing the clustering results to externally provided ground truth information, that is the correct clustering of the data points. some common extrinsic evaluation metrics are:

- Rand Index (RI): The Rand Index measures the similarity between two clusterings by comparing pairs of data points and evaluating if they are assigned to the same or different clusters in both clusterings. To Calculate we first define the following.
  - $a$  represents the number of pairs that are assigned to the same cluster in both the true and predicted clusterings.
  - $b$  represents the number of pairs that are assigned to different clusters in both the true and predicted clusterings.
  - $c$  represents the number of pairs that are assigned to the same cluster in the true clustering but to different clusters in the predicted clustering.
  - $d$  represents the number of pairs that are assigned to different clusters in the true clustering but to the same cluster in the predicted clustering.

$$RI = \frac{a + b}{a + b + c + d}$$

- Adjusted Rand Index (ARI): The ARI improves upon the Rand Index by considering the expected agreement by chance, taking into account the cluster sizes and the number of possible agreements. The ARI is calculated using the following notation:

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

Here,  $\max(RI)$  represents the maximum possible value of the Rand Index (equal 1). The ARI ranges from -1 to 1, where a value of 1 indicates a perfect agreement between the clusterings, a value of 0 indicates the agreement expected by chance, and negative values suggest a disagreement beyond what would be expected by chance.

## 2.4 Graph Clustering Techniques

Graph clustering approaches are a category of data analysis methods that aim to group data points or elements based on their relationships within a graphical representation, typically in the form of a network or graph. These approaches are widely used in various fields, including social network analysis, biology, recommendation systems, and more. They leverages the inherent structure of the data to uncover patterns and groupings that might not be apparent through

other methods.

Connected components and the Louvain algorithm are two widely used techniques in the realm of graphical clustering, particularly for community detection within networks.

### Weakly Connected Components

In the context of graph theory, connected components represent subgraphs within a larger graph where each node is reachable from every other node in the same component by following edges. This concept serves as a building block for clustering and partitioning in graph theory.

Weakly Connected components algorithm leverages that concept to find sets of connected nodes in directed and undirected graphs. It only needs a path to exist between pairs of nodes in one direction, whereas Strongly Connected Components needs a path to exist in both directions.

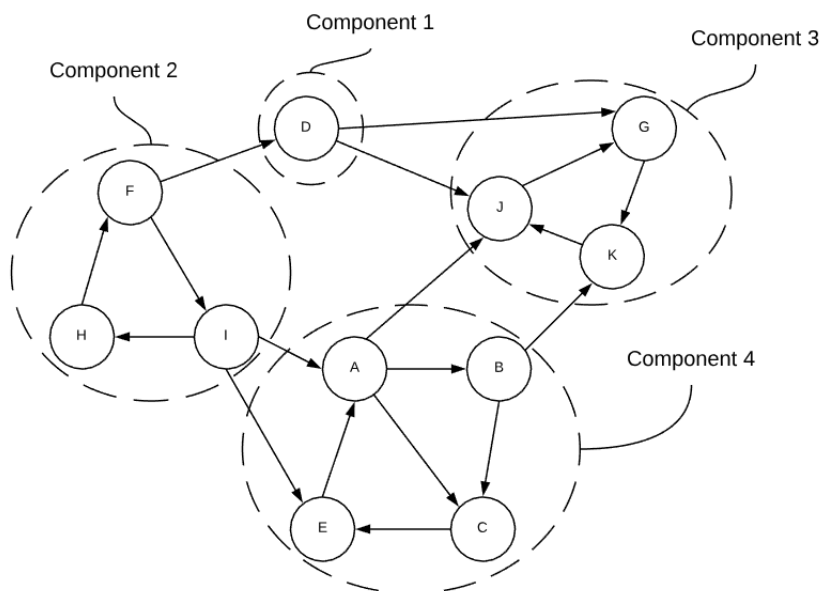


Figure 2.5: Example of (CC) clustering

### Louvain Algorithm

Louvain Algorithm is a sophisticated modularity-based community detection algorithm. It aims to maximize the modularity of a network, a measure that quantifies the quality of the community structure based on the density of connections "within" and "between" clusters.

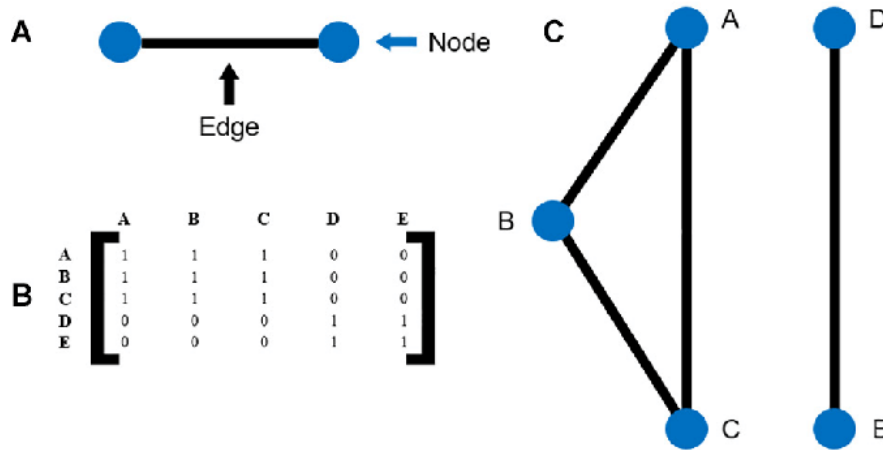


Figure 2.6: Nodes, Edges and the Adjacency Matrix

The Modularity  $Q$  formula:

$$Q = \frac{1}{2m} \sum_{i,j} \left[ A_{ij} - \frac{k_i k_j}{2m} + \sigma(C_i, C_j) \right] \quad (2.20)$$

Where:

$G$  The graph with  $n$  nodes and  $m$  edges.

$A_{ij}$  The adjacency matrix of the graph, representing the connections between nodes  $i$  and  $j$ .

$C_i$  The community to which node  $i$  belongs.

$K_i$  The degree (number of connections) of node  $i$ .

The Louvain algorithm optimizes this modularity  $Q$  by iteratively moving nodes between communities to maximize it. The algorithm consists of two phases:

### Phase 1: Agglomeration Phase

- Start with each node in its own community:  $C_i = i$  for  $i = 1$  to  $n$ .
- For each node  $i$ , compute the change in modularity  $\Delta Q$  by moving  $i$  to the community of its neighbor  $j$  if  $\Delta Q$  is positive. Repeat this for all nodes until no further improvement in modularity can be achieved.
- Aggregate the communities formed in this phase, treating them as nodes in a new network.

### Phase 2: Hierarchical Iteration

- Repeat Phase 1 on the aggregated network until no further improvement in modularity can be achieved.
- This process creates a hierarchy of communities, revealing both fine-grained and coarse-grained structures.



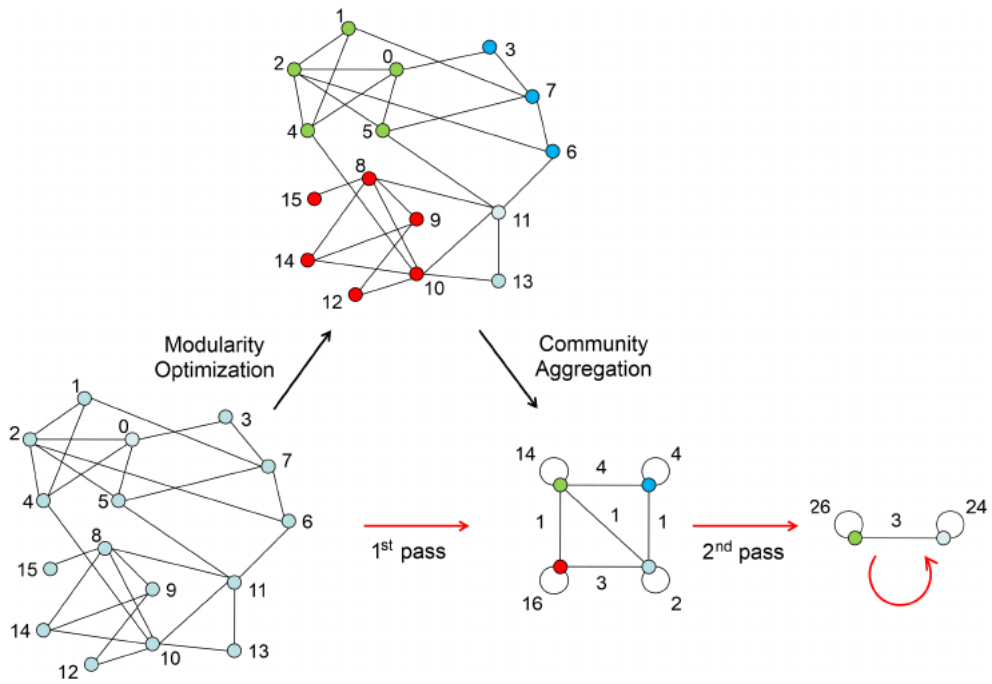


Figure 2.7: Example of Louvain Algorithm steps

## Chapter 3

# The Entity Resolution Framework

Entity resolution (ER) is a multi-step process that tackles the challenges of resolving entities in a systematic manner. It involves a series of subtasks or steps aimed at achieving accurate entity resolution. In this section, we will explore the main steps of the ER task and examine various techniques developed to address each subtask within these steps.

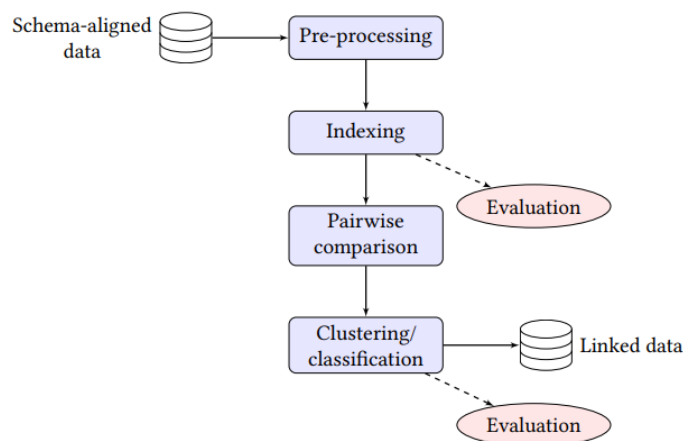


Figure 3.1: A high level reference model of an ER process [17]

To provide a visual representation of the ER process, Figure 4.4 is included as a helpful reference. The process starts with the data pre-processing stage, where inconsistencies and errors are addressed to improve data quality. Next, the data is partitioned into blocks based on specific attributes, reducing the search space for comparisons. Comparisons are then performed using similarity metrics to identify potential matches between records. Classification or clustering techniques are applied to group similar records or to unique entities. Finally, the effectiveness of the entity resolution process is evaluated using metrics like recall, pair completeness, and overhead time. The following sections describe each of these steps in detail.

### 3.1 Phase 1: Data Pre-processing

The significance of clean and standardized data cannot be overstated when it comes to enhancing the quality and accuracy of any data integration endeavor. Given that entity resolution involves data from disparate sources, it is common for the data to exhibit variations in format or even inconsistencies and errors. Extensive research has underscored the criticality of data cleaning and standardization as essential steps for successful entity resolution [9] [10].

Data quality encompasses several dimensions, with accuracy, consistency, and completeness being particularly relevant to the entity resolution problem [11]. Accuracy reflects the correctness and precision of the data, while completeness assesses the extent to which data elements are adequately present and useful for the task at hand. Consistency ensures coherence and uniformity across records, fostering a cohesive and harmonized dataset.

To tackle these challenges, various standard data preprocessing techniques are commonly employed. These techniques involve a range of processes aimed at improving data quality, such as:

- **Standardization:** The goal of Standardization is to bring consistency and uniformity to the data. Techniques such as case normalization are used to convert names to a consistent case, such as converting "John Smith" and "JOHN SMITH" to "john smith". Punctuation and symbol removal techniques aim to eliminate punctuation marks, special symbols, non-alphanumeric characters and rectify misspellings to ensure consistency in the data.
- **Parsing:** This crucial step involves analyzing the structure and syntax of textual data to extract meaningful information and identify specific elements within the data. It is particularly important when dealing with complex textual data such as citation records or publication metadata. Two commonly employed techniques for parsing are rule-based parsing and statistical parsing. Rule-based parsing utilizes predefined grammatical rules to capture elements based on specific patterns, for example: author names based on patterns like "Last name, First name" or "First name Last name". Statistical parsing, on the other hand, leverages algorithms to analyze the syntax and structure of the data.
- **Transformation:** This key step involves performing various operations to ensure data quality and consistency. General transformation steps encompass different actions such as converting data types, renaming fields, decoding encoded values, checking data ranges, and validating dependencies [14]. In certain scenarios due to specific indexing or comparison techniques, data tokenization and segmentation are employed as additional steps [15].

### 3.2 Phase 2: Blocking Scheme

The blocking (indexing) phase in entity resolution is essential to reduce the computational complexity of the matching process. When dealing with a dataset of  $n$  records, the naive approach for

entity resolution would require  $n(n - 1)/2$  comparisons, resulting in a quadratic computational complexity of  $O(n^2)$ . As the dataset size grows, this approach becomes increasingly inefficient and can create performance bottlenecks [13].

To address this challenge, blocking partitions the input dataset into smaller blocks of similar records, utilizing a blocking criterion or key. Each block consists of records that share the same blocking key value (BKV) for a specific criterion. By grouping similar records together, the number of pairwise comparisons is significantly reduced, leading to improved efficiency and scalability in the entity resolution process.

While blocking reduces the complexity of the matching stage, it introduces the challenge of finding an optimal blocking strategy, this involves finding the right balance between minimizing the possibility of matching records assigned to different blocks and reducing the number of false matches that are incorrectly considered as true matches. There have been numerous blocking strategies developed, we will highlight a few notable approaches:

### 3.2.1 Standard Blocking

Standard blocking, initially proposed by Fellegi et al. [4], introduced an early approach to perform blocking in record linkage tasks. This method involves representing each entity with one or more BKVs. Each block is linked to a unique key value and encompasses all entities that share that same key value. Records in the same block are compared, which can reduce the number of record comparisons in the initial entity resolution process.

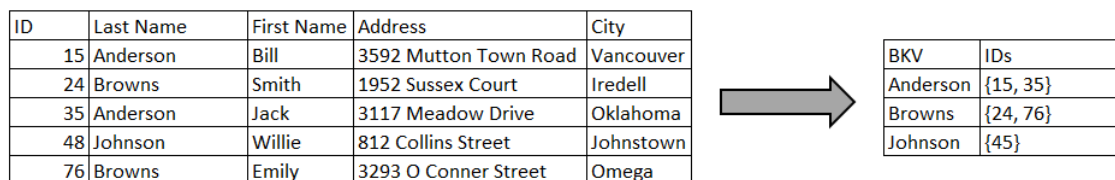


Figure 3.2: Result of standard blocking on Last Name

Although standard blocking does not have explicit parameters, the size of the blocks can vary based on the frequency distribution of the BKVs, which, in turn, is influenced by the definition of the blocking keys themselves. This variation poses challenges in predicting the number of record pairs to be compared during the process. Additionally, it is apparent that this method is highly susceptible to errors in the data, such as instances of noisy data or data containing typos and misspellings.

### 3.2.2 Sorted Neighborhood Blocking

Hernandez et al [8] proposed an alternative to standard blocking known as the sorted neighbourhood method (SNM). The main idea is to sort the records in alphabetical orders, based on

their BKVs. It then constructs blocks by passing a sliding window over sorted records. Records enclosed in one sliding window belong to the same block, Figure 3.3 shows the concept.

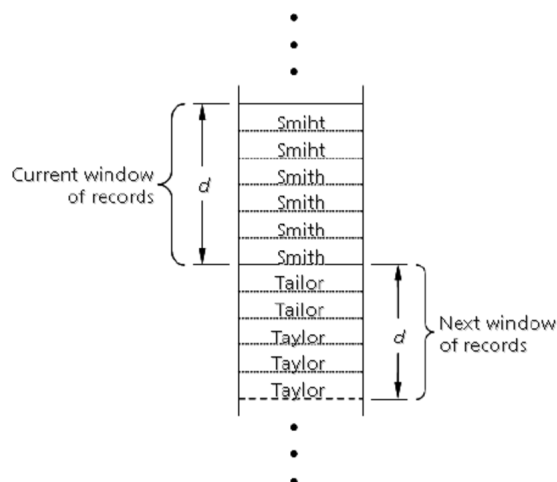


Figure 3.3: Concept of the Sorted-Neighborhood Method [18]

One key parameter to this approach is the sliding window size. If the size of the window is too small, and many records share the same BKVs, matching record pairs will be missed. On the other hand, large values of window size will introduce more false positives. Hence, there's a trade-off between recall and efficiency. There exist However an Adapative SNM that dynamically changes the size of the window, based on a comparison function and a threshold value.

### 3.2.3 Q-gram Based blocking

When employing the above mentioned blocking strategies, a significant number of true matches can be inadvertently missed, especially when the blocking keys for these true matches are not identical due to noisy data. To this effect, Gavarno et al [19] introduced the concept of positional q-grams, which involves considering sub-sequences of q characters with their respective positions within the characters. Assuming  $q=2$ , and an attribute value "Smith", the list of the positional q-grams would be ["Sm", "mi", "it", "th"].

The algorithm is based on the assumption that entities that share a large number of q-grams are more likely to be matches. While the approach showcases high resilience against noisy data, it comes at the cost of relatively lower efficiency due to the increased number of blocks generated. In order to improve the performance of Q-grams based blocking, Christen [20] uses combinations of q-grams, instead of individual q-grams. It generates these combinations as sublists from the initial list of q-grams, down to a certain minimum length  $l$  determined based on a user-defined threshold  $t$ . The minimum length  $l$  calculation is as follows:  $l = \max(1, [k \times t])$ , where  $k$  denotes the number of q-grams,  $t \in [0, 1)$ , and  $[ ]$  denoting the rounding to the next lower integer value. Each set of q-grams is then concatenated to provide the indexing key value. An example is shown in figure 3.4

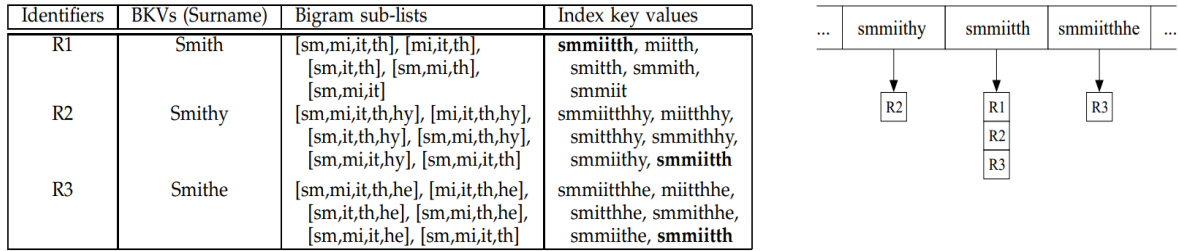
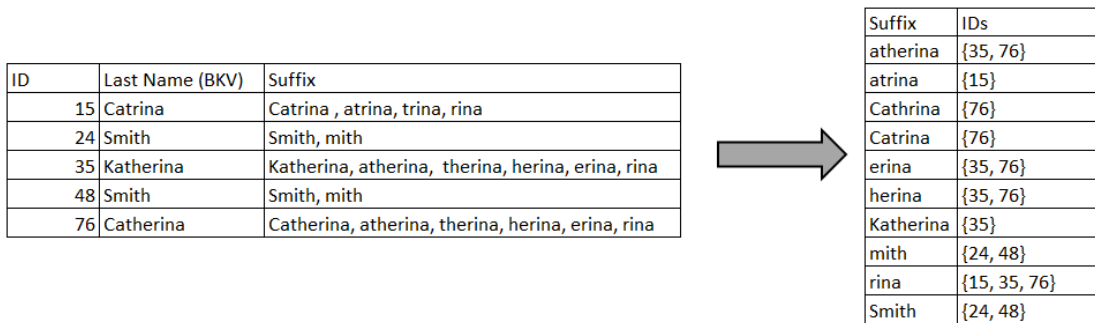


Figure 3.4: Example of Q-gram based blocking with Surnames as BKV [20]

### 3.2.4 Suffix Array Blocking

Similar to Q-gram based blocking, Aizawa et al. [21] propose a method that addresses errors and variations in BKVs by generating all of their suffixes, down to a minimum length  $l_{min}$ . A suffix of a string is a substring obtained by removing one or more characters from the beginning. For instance, the suffixes of the string 'Adams' would be 'dams', 'ams', 'ms', and 's'.

The basic idea involves inserting the BKV and its generated suffixes into an array-based inverted index called the suffix array. Subsequently, the algorithm sorts this index alphabetically to establish the blocking scheme. The process is visually illustrated in Figure 3.5.


 Figure 3.5: Example of suffix array based blocking with minimum length  $l_{min} = 4$ 

To control the size and number of blocks generated, the algorithm relies on two parameters:

- Minimum suffix length  $l_{min}$ : This parameter determines the minimum length of suffix strings that are generated. For instance, if  $l_{min}$  is set to 4, the string 'Johnson' would generate the following suffixes: 'ohnson', 'hnsion', and 'nson'. In cases where the BKVs are already shorter than the minimum length of  $l_{min}$  characters, their actual values are used directly as index keys.
- Maximum block size  $b_{max}$ : To limit the size of the generated blocks, this parameter sets maximum block size that is allowed. After the BKVs and the suffixes have been generated and inserted to the suffix array structure, all suffixes that group more than  $b_{max}$  record identifiers within their block will be removed. This deletion is performed as these suffixes typically represent common substrings present in certain names, such as "ane" or "ana" in Western names.

The algorithm demonstrates robustness against noise occurring at the beginning of blocking keys, as illustrated in the example of "Catherina" and "Katerina" above. However, it does not effectively handle noise located at the end. For instance, "Smith" and "Smithe" do not share any common suffix.

### 3.2.5 Block Filtering

The Block Filtering approach in entity resolution is a technique used to reduce the number of initial blocks generated by a blocking function in order to improve the efficiency and reduce the computational load in the subsequent phases of the process.

The approach was first proposed in [12] it leverages the idea that smaller blocks tend to be more precise because they contain records with higher similarities or matching attributes, while larger blocks indicate a suboptimal choice of blocking key and contain a mix of both matching and non-matching records, making them less informative.

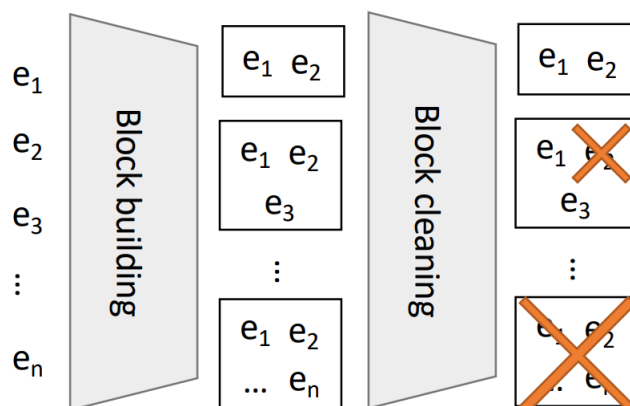


Figure 3.6: The Block Filtering Approach

While sorting and filtering blocks based on size is a straightforward approach, it's worth noting that the choice of threshold can significantly impact the effectiveness of the entity resolution process. Selecting an appropriate threshold requires a good understanding of the data and domain-specific knowledge. Experimentation and validation with real data may be necessary to fine-tune the threshold for optimal results

### 3.2.6 Evaluation

There are several metrics used to evaluate a blocking strategy. Here, we will talk about some of the most notable ones:

- **Reduction Ratio:** This measure assesses the efficiency of an indexing technique by comparing the number of candidate record pairs generated to the total number of possible

combinations of record pairs. The calculation for the reduction ratio is as follows:

$$RR = 1 - \frac{||B||}{||E||} \quad (3.1)$$

where:

$||B||$  = number of potential matching pairs generated by a blocking algorithm

$||E||$  = number of all pairs in the set E

- **Pairs Completeness:** Also referred to as "recall," quantifies the number of true matching record pairs generated by a blocking technique, divided by the total number of true matching pairs within the entire comparison space. The value of pairs completeness ranges between 0 and 1 and directly corresponds to the effectiveness of the algorithm. The formula for calculating pairs completeness is as follows:

$$PC = \frac{M_B}{M_E} \quad (3.2)$$

where:

$M_B$  = number of true matching pairs captured by a blocking algorithm

$M_E$  = number of true matching pairs in the set E

- **Pairs Quality:** A metric that corresponds to "precision", it represents the ratio of true matching pairs generated by a blocking algorithm to the total number of potential matching pairs generated. The calculation for pairs quality is as follows:

$$PQ = \frac{M_B}{||B||} \quad (3.3)$$

where:

$M_B$  = number of true matching pairs captured by a blocking algorithm

$||B||$  = number of potential matching pairs generated by a blocking

### 3.3 Phase 3: Pairwise Comparison

An essential step in the ER process, and one that makes it such a compute-intensive and time-consuming task. This step involves performing detailed comparisons once the initial comparison space has been significantly reduced by identifying candidate pairs.

In contrast to the blocking step, which relies on one or a combination of a few attribute values, the comparison step utilizes multiple attribute values (e.g., last name, first name, address, zip code, etc.). The comparisons are performed by applying various similarity functions to the given attributes, resulting in a similarity vector that consists of one or more numerical values that



represent the overall similarity of two records.

For the sake of brevity, we will focus solely on a subset of similarity measures utilized for comparing strings. Other similarity measures for different data types are beyond the scope of this work.

### 3.3.1 Levenshtein Similarity

One of the early approaches to measuring string similarity based on the concept of edit distance (i.e. count of the smallest number of edit operations that are required to convert one string into another). The allowed operations for Levenshtein Distance are character insertions, character deletions, and character substitutions. It assigns a unit cost to each operation and is given by the following formula.

$$lev_{a,b}(i,j) = \begin{cases} \max(|i|, |j|), & \text{if } \min(|i|, |j|) = 0 \\ \min \begin{cases} lev_{a,b}(i-1, j) + 1 \\ lev_{a,b}(i, j-1) + 1 \\ lev_{a,b}(i-1, j-1) + k \end{cases} & \text{otherwise} \end{cases} \quad (3.4)$$

where:

- $a, b$  strings to compare
- $|a|, |b|$  length of strings
- $i, j$  indexes of strings  $a, b$  respectively

$k = 0$  if  $(a_i = b_j)$ ,  $1$  otherwise

Finally, to calculate the Levenshtein similarity:

$$sim_{lev}(a, b) = 1 - \frac{lev(a, b)}{\max(|a|, |b|)} \quad (3.5)$$

A commonly used variation of the Levenshtein distance is the Damerau-Levenshtein distance [22], which incorporates an additional edit operation known as transposition. This operation involves swapping two adjacent characters in the string.

### 3.3.2 Longest Common Substring Similarity

The longest common substring algorithm compares the characters of two strings to identify matching subsequences. It repetitively searches for the longest matching substring, removes it, and repeats the process until no substrings with a minimum number of characters, denoted as  $l_{min}$ , are found. In practice, the value of  $l_{min}$  is typically determined based on the specific attributes being compared. A similarity value is then calculated using either the Overlap, Jaccard or Dice coefficient, as follows:

$$sim_{lcs-overlap}(a, b) = \frac{l_c}{\min(|a|, |b|)} \quad (3.6)$$

$$sim_{lcs-jaccard}(a, b) = \frac{l_c}{|a| + |b| - |l_c|} \quad (3.7)$$

$$sim_{lcs-dice}(a, b) = \frac{2 \times l_c}{|a| + |b|} \quad (3.8)$$

where:

$a, b$  strings to compare

$l_c = \sum_{i=1}^n |s_{ci}|$  total summed length of all found common substrings  $s_{ci}$

Edit distance utilises the strings as a whole without splitting them into segments or token, which for longer strings results in an increased computational complexity, to this end several token-based similarity measure exist, the following overview some of them:

### 3.3.3 Q-gram Based Similarity

The q-gram similarity is a measure of how similar two strings are based on their shared substrings of length q, known as q-grams. Q-grams are obtained by using a sliding window of length q that moves through the strings, extracting substrings at each position. The q-gram similarity is calculated by comparing the sets of q-grams of the two strings and determining the number of common q-grams they share.

After identifying the common q-grams, various similarity coefficients can be used to calculate the similarity value. These coefficients include the Overlap coefficient, Jaccard coefficient, and Dice coefficient:

$$sim_{qgram-overlap}(a, b) = \frac{q_{common}}{\min(q_a, q_b)} \quad (3.9)$$

$$sim_{qgram-jaccard}(a, b) = \frac{q_{common}}{q_a + q_b - q_{common}} \quad (3.10)$$

$$sim_{qgram-dice}(a, b) = \frac{2 \times q_{common}}{q_a + q_b} \quad (3.11)$$

where:

$q_a, q_b$  number of q-grams generated from strings a, b respectively

$q_{common}$  number of common q-grams between a and b

The q-gram similarity is known for its computational efficiency and ease of implementation. It enables approximate matches by considering shared q-grams, even when the strings are not exact matches. However, it's important to recognize that the q-gram similarity has limitations. It focuses solely on the shared q-grams and does not consider the semantic or contextual meaning of the strings. As a result, it may not be suitable for applications that require a deeper understanding or interpretation of the string content.

### 3.3.4 Jaro-Winkler Similarity

The Jaro and Winkler comparison functions are A special group of string comparison methods primarily designed for measuring name similarity [31] [32].

The basic Jaro comparison method integrates edit distance and q-gram distance techniques. It calculates the number of shared characters between two strings within a defined character window, which is half the length of the longer string, and the number of transpositions (the swapping of adjacent characters). The Jaro similarity value can then be calculated as:

$$sim_{jaro}(a, b) = \frac{1}{3} \left( \frac{c}{|a|} + \frac{c}{|b|} + \frac{c-t}{c} \right) \quad (3.12)$$

Where:

$c$  number of common characters between strings a and b

$t$  number of transpositions between strings a and b

The Jaro-Winkler method increases the similarity value between two strings if they share the same number of starting characters (prefix). It is calculated as:

$$sim_{winkler}(a, b) = sim_{jaro}(a, b) + (1.0 - sim_{jaro}(a, b)) \frac{p}{10} \quad (3.13)$$

Where:

$p(0 \leq p \leq 4)$  the number of common characters of the prefixes of the strings

### 3.3.5 Monge-Elkan Similarity

A specialized approximate string comparison function designed for estimating the similarity of multi-word string values, such as addresses or institution names.

The fundamental concept behind this approach involves initially extracting tokens (i.e., words or elements separated by whitespace characters) from the two input strings. Subsequently, and then identify the best matching pairs of tokens in the sets of tokens using a secondary similarity function referred to as  $sim'$  (typically Jaro or Jaro-Winkler). In accordance with the notation initially introduced in [26], this recursive matching scheme divides the two strings, denoted as  $s1$  and  $s2$ , into two token sets,  $A$  and  $B$ . The similarity between  $s1$  and  $s2$  is then computed as:

$$sim_{monge\text{-}elkan}(s1, s2) = \frac{1}{|A|} \sum_{i=1}^{|A|} \max_{j=1}^{|B|} sim'(A_i, B_j) \quad (3.14)$$

### 3.4 Phase 4: Classification

As discussed earlier, the comparison phase generates a bunch of similarity vectors between each candidate record pairs, it's in the classification phase when a decision is made about these record pairs, as being either a match or non-match (and in some cases, potential matches ) based on the decision model used. There are various classification methods developed, we will list them based on the categories they fall into in the ER literature:

#### 3.4.1 Probabilistic Classification

In their widely recognized seminar paper, Fellegi and Sunter introduced the concept of "Probabilistic record linkage" [4]. Their approach examines the discriminatory power of individual identifiers (attributes) and assesses the likelihood of two records being a true match, considering the agreement or disagreement on different identifiers. Furthermore, they emphasized that the assigned weights for these identifiers should not solely rely on the general attributes' characteristics but also consider the specific attribute values themselves. For example, two records with the surnames, "Johnson" should receive less matching weight than two records that share the surname "Vanderheim", assuming that the number of people with the surname "Johnson" is more significant than the number of people with surname "Vanderheim".

Considering Datasets  $A$  and  $B$ , the sets  $M$  and  $U$ , such as:

$$\begin{aligned} M &= \{(a, b); a = b, a \in A, b \in B\} \\ U &= \{(a, b); a \neq b, a \in A, b \in B\} \end{aligned} \quad (3.15)$$

where  $M$  represents the set of true matches and  $U$  the set of true non-matches. Then for each attribute  $i$  compared, the individual weight  $w_i$  for that attribute is calculated as:

$$w_i = \begin{cases} \log_2\left(\frac{m_i}{u_i}\right) & \text{if } a_i = b_i \\ \log_2\left(\frac{1-m_i}{1-u_i}\right) & \text{if } a_i \neq b_i \end{cases} \quad (3.16)$$

$m_i$  and  $u_i$  are the probabilities that two records having the same value in attribute  $i$  are a true match or a true non-match. A limitation of the conventional Fellegi and Sunter approach is the challenge of estimating the conditional probabilities  $m$  and  $u$ . These probabilities typically require prior knowledge and often rely on domain experts' expertise. To address this, various extensions have been proposed that consider the frequency of attribute values to calculate the  $m$  and  $u$  probabilities [24] [10]. Furthermore, alternative approaches integrate Bayesian networks with the traditional probabilistic model [25].

#### 3.4.2 Rule Based Classification

Rule-based techniques employ a predefined set of rules to classify candidate record pairs as either matches or non-matches (or potential matches) [27]. The rules can be applied to the similarity values of the comparison vectors. They represent a set of tests on the similarity values, combining logical operators such as conjunctions (logical AND), disjunctions (logical OR), and

negations (logical NOT).

The formal notation of a rule is  $P \implies C$ , where  $P$  is a predicate that is applied on the similarity values, and  $C$  is the classification outcome. For example:

$$(s(\textit{LastName})[a, b] \geq 0.9) \wedge (s(\textit{BirthDate})[a, b] = 1) \implies [a, b] \longrightarrow \textit{Match}$$

The left-hand side represents the predicate that is applied. Once the predicate is triggered (i.e. the record pair has a similarity of more than 0.9 on `LastName` and an exact match on `BirthDate`), the record pair is classified as a match. The effectiveness of rule-based methods heavily relies on the expertise of domain specialists in defining a well-crafted set of rules. As a result, the generation and assessment of rules pose significant challenges in these approaches. Alternatively, some studies generate rules by directly learning them from the training data [28].

### 3.4.3 Learning-Based Classification

Learning-based approaches have emerged as a powerful paradigm for tackling entity resolution tasks. Instead of relying on predefined rules or heuristics, these approaches leverage machine learning algorithms to automatically learn patterns and relationships from the data. There have been various learning-based techniques applied to entity resolution. In the subsequent sections, we will delve into the explanation of some of these techniques.

#### Supervised Classification

Supervised classification approaches consider two distinct classes: matches and non-matches. These approaches train a classifier on a labeled training dataset. The trained model is subsequently applied to classify record pairs whose match status is unknown. Two of the most popular supervised learning classifiers applied in the ER process are Decision trees and support vector machines [29] [30]:

Supervised classification approaches have two significant drawbacks. Firstly, the nature of the entity resolution task leads to the creation of a highly imbalanced dataset, where the number of record pairs representing true non-matches significantly outweighs the number of record pairs representing true matches. This hinders the performance of most supervised classification methods, which results in poor performance.

The second challenge revolves around acquiring and generating training data. In many instances, these data necessitate manual labeling by human experts, which can be both costly and time-consuming. This further compounds the complexity of implementing the supervised classification approach effectively.

#### Unsupervised Approaches

The techniques mentioned above are all aimed toward performing entity resolution through pairwise comparisons of records. In contrast, various unsupervised approaches have been developed

to cluster or group records together, with each cluster representing a distinct entity.

Clustering approaches aim to create clusters characterized by high intra-cluster similarity and low inter-cluster similarity. This implies that elements within a cluster exhibit similarity to one another, while elements belonging to different clusters are dissimilar to each other.

In their unsupervised approach, Verykios et al. proposed utilizing similarity vectors generated from pairwise comparisons. The authors assumed that these vectors can be categorized into three groups: Match, Non-Match, and Potential Match. They aimed to cluster the vectors into these categories. Identifying the Match and Non-Match clusters was relatively straightforward. For the Match cluster, the centroid vector would closely resemble a perfect match vector  $[1.0, \dots, 1.0]$ , consisting of high similarity values in all components of the comparison vector. On the other hand, the centroid vector of the Non-Match cluster would closely resemble a perfect non-match vector  $[0.0, \dots, 0.0]$ , indicating low similarity in all components of the comparison vector. These two clusters are then used as training data for a decision tree classifier, which can then be used on the remaining records.

### 3.5 Phase 5: Clustering as a Post-processing Step

Clustering can be applied as a post-processing step to handle the match probabilities or scores obtained from a trained supervised model. This approach aims to address a common issue encountered by supervised methods, referred to as "intransitive closure" (see Appendix A).

Hassanzadeh et al. [16] proposed a clustering method called CENTER, which involves generating a graph of matching record pairs based on the output of a supervised model. In this graph, each node represents a record, and the edges between them represent the match probabilities provided by the supervised model. The algorithm performs clustering by partitioning the record graph, ensuring that each cluster has a center, and all records within a cluster are similar to that center.

To achieve this, the edges (pairs) are sorted in descending order of their match probabilities. When processing the edges, the algorithm identifies the first occurrence of an edge  $(u_i, u_j)$ , designating  $u_i$  as the center of the cluster. All subsequent nodes  $u_j$  that appear in an edge  $(u_i, u_j)$  are assigned to the cluster of  $u_i$  and are not considered again. If an edge is encountered where both nodes are already assigned to clusters, that edge is discarded. Figure 2.9 provides an illustrative example of the algorithm's result.

PairsID	LastName	FirstName	Address	Bdate	pred_prob
(15,17)	0.72	0.76	0.63	0.85	0.66
(17,18)	0.78	0.75	0.56	0.81	0.65
(18,22)	0.81	0.88	0.53	0.79	0.69
(18,23)	0.74	0.72	0.68	0.70	0.62
(22,23)	0.75	0.77	0.68	0.79	0.63

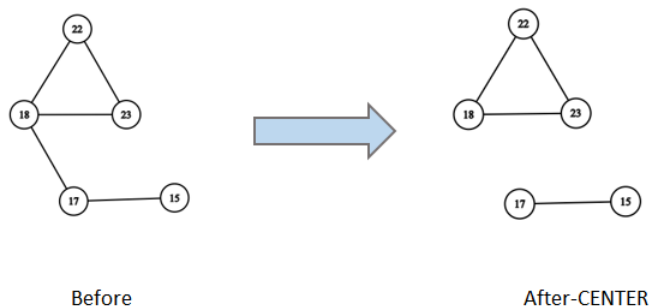


Figure 3.7: Example of applying CENTER to partition the subgraph of matches, to solve intransitivity

### 3.6 Phase 6: Evaluation

Evaluation of entity resolution processes serves various purposes, such as parameter tuning, benchmarking, and ensuring quality before deployment. While some metrics are borrowed from the classification and clustering domains, there are key differences that distinguish entity resolution from general classification or clustering tasks.

Given an input set of entities and the predicted clustering  $P$ , and the ground truth clustering  $T$ , The evaluation metrics can be categorized into two main groups:

#### Pairwise performance measures

When considering ER as a binary classification task, metrics from supervised learning can be adapted to the ER problem. Each pair of entities is classified as a True Positive (True Match), True Negative (True Non-match), False Positive (False Match), or False Negative (False Non-match). The following metrics can be applied to measure performance in entity resolution:

$$Pair\_Precision = \frac{|Pairs(P) \cap Pairs(T)|}{|Pairs(P)|} \quad Pair\_Recall = \frac{|Pairs(P) \cap Pairs(T)|}{|Pairs(T)|}$$

The advantage of these pairwise metrics is their ease of interpretability. However, they may not always provide a comprehensive overview of performance, and that's where the next metrics help.

#### Clustering Performance Measures

While some metrics defined in the clustering domain can still be used for entity resolution evaluation, their results may be difficult to interpret due to the nature of the entity resolution task. In entity resolution, the number of generated clusters is often significantly higher compared

to the number of elements within a cluster. Therefore, in evaluating entity resolution, it is common to only consider matches (intra-cluster pairs) and exclude non-matches (inter-cluster pairs). The following cluster metrics have been proposed in the literature:

- Cluster precision, recall, and F1: These cluster-level metrics are defined in terms of exact cluster matches. They calculate the precision and recall of the predicted clusters compared to the ground truth clusters:

$$C\_Precision = \frac{|P \cap T|}{|P|} \quad C\_Recall = \frac{|P \cap T|}{|T|}$$

The F1 score is the harmonic mean of precision and recall. It is important to note that these metrics are more stringent because a single mismatched link would result in the exclusion of the entire cluster. Consequently, they are less frequently used compared to the next set of metrics.

- Closest Cluster Precision, Recall, and F1: An alternative measure to the ones described earlier is the closest cluster metrics. These metrics are less strict and use the Jaccard similarity coefficient to capture cluster similarity. They calculate the precision and recall of the predicted clusters based on the closest matching cluster in the ground truth:

$$CC\_Precision = \frac{\sum_{p \in P} \max_{t \in T} J(p, t)}{|p|} \quad CC\_Recall = \frac{\sum_{t \in T} \max_{p \in P} J(p, t)}{|t|}$$

where  $p$  and  $t$  are clusters in  $P$  and  $T$ , respectively. These metrics consider the similarity between clusters rather than exact matches, providing a more flexible evaluation approach for entity resolution.

- Average Cluster Purity :measures the quality of clustering results by evaluating the purity of each individual cluster. It assesses how well the instances within each cluster belong to the same class or category. It is defined as:

$$ACP = \frac{1}{N} \sum_{p \in P} \sum_{t \in T} \frac{|p \cap t|}{p}$$



## Chapter 4

# Proposed Solution

Having reviewed the current state of entity resolution techniques and established the essential knowledge required to comprehend the proposed solution, we will now proceed to present the solution for entity resolution in bibliographic datasets.

While bibliographic datasets encompass various entities (authors, journals, institutions, etc.), our solution will primarily address the challenge of author name disambiguation. This problem, in its own right, is widely recognized within the field of entity resolution. We will first introduce the mathematical formulation of the problem. Next, we will navigate through the framework described in Section 2.5, providing explanations of the various methods employed at each step. Additionally, whenever possible, we will evaluate the performance of these methods.

### 4.1 Problem Definition

The problem of author name disambiguation (AND) can be formulated as follows:

Given:

- A set of scholarly publications  $P = p_1, p_2, \dots, p_n$ , where each publication  $p_i$  is associated with a set of authors  $A(p_i) = a_1, a_2, \dots, a_m$ . The publications are indexed by  $i = 1, 2, \dots, n$ .
- A set of author names  $N = n_1, n_2, \dots, n_m$ , where each name  $n_j$  corresponds to one or more authors in the set of publications.

#### **Objective:**

The objective of AND models is to partition the set of author names  $N$  into clusters, such that author names within the same cluster are likely to represent the same author (entity). It should be noted that:

- Each author can be associated with multiple names that refer to them, such as variations in spellings or different name representations (e.g., Thompson, L. A. - Thompson, Larnes A., etc.).
- An author name can refer to multiple authors, as different individuals may share the same or similar names (e.g., Smith, J can refer to Smith, James or Smith, John, etc.)

In this section, we will delve deeper into the data that will be used for our analysis. We will explore the data generation process, examine data properties, and conduct a preliminary analysis.

## 4.2 Data Comprehension

Data comprehension is a crucial step in any data analysis process. It involves understanding the data collection process, examining data sources, analyzing the structure of the data, identifying patterns, and gaining insights into the underlying characteristics of the data. Furthermore, data comprehension helps in detecting and resolving any issues or problems related to the data.

### 4.2.1 Data Collection

The data utilized in the solution is sourced from Scopus, an online multidisciplinary citation database. Scopus stands as the largest abstract and citation database encompassing peer-reviewed literature, including scientific journals, books, and conference proceedings. This extensive repository comprises a vast collection of over 75 million research papers authored by millions of contributors.



Figure 4.1: For more information check: <https://dev.elsevier.com/>

To extract the data, a Python script is employed, which continuously runs and utilizes Scopus' API to access the database. Through the script, queries are executed to retrieve all publications within a specific domain, such as space resources in the context of this study. Unfortunately, the specific content and details of the script are proprietary information belonging to the company and cannot be shared in this context.

### 4.2.2 Data Analysis

The script is designed to execute specific queries that retrieve data from various time periods, encompassing everything prior to 2000 as well as every year from 2000 to the present. Subsequently, the data obtained from each time range is consolidated into a single file. Each consolidated file

encompasses comprehensive information pertaining to publications within a particular domain and time range. These files include the following relevant information:

- Title: The title of each publication.
- Author names: A list of all co-authors associated with each publication.
- Journal: The name of the publication journal where each article was published.
- Affiliations\*: The institutes that the authors associate to.
- Abstract\*: A concise summary of each research paper, typically comprising around 200-250 words.
- Author keywords\*: Terms selected by the authors to best reflect the content of the document

Upon closer examination of the datasets, we can analyze the presence of missing values in certain attributes and assess the coverage of these attributes. The figure presented below provides insights into the percentages of missing values across nine datasets:

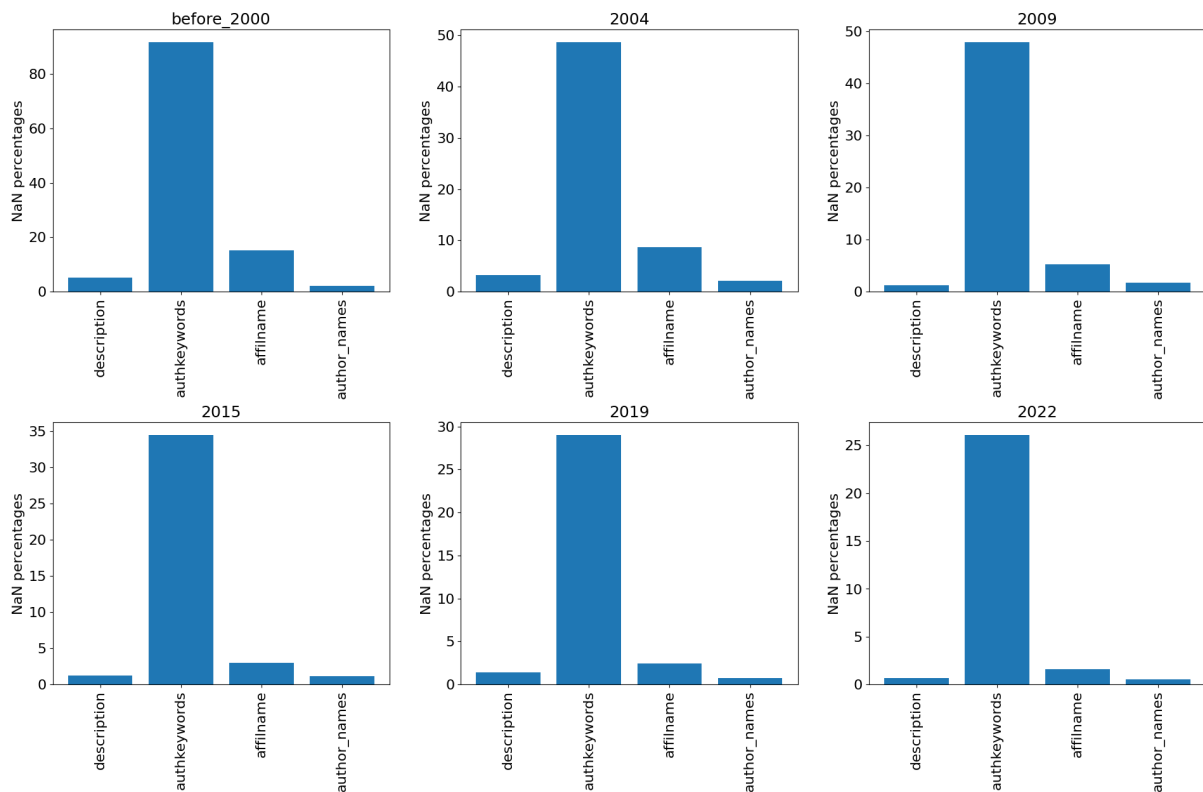


Figure 4.2: Percentage of missing values in some of the datasets

The figure (Figure 1) depicts the extent of missing values in the aforementioned datasets. As we delve into the data, we observe that several columns exhibit missing values, specifically in

---

\*Not always available

the fields of author names, affiliations, keywords, and descriptions (abstracts). An analysis of these missing values reveals certain patterns and observations:

- **Author Names:** The missing values in the author names column are not due to data errors but rather indicate conference proceedings where an actual author name is not applicable.
- **Abstracts:** The majority of missing values in the abstract column correspond to conference proceedings. Since these proceedings compile research papers, there is no abstract specifically associated with them. However, it is worth noting that a small number of actual research papers also have missing abstracts.
- **Affiliations:** Similar to author names and abstracts, missing values in the affiliations column primarily occur in conference proceedings. Nevertheless, the total number of missing affiliations surpasses that of author names or abstracts, indicating that there is a significant number of actual research papers with missing author affiliations.
- **Keywords:** The column containing author keywords exhibits the highest percentage of missing values across all datasets. In papers published before the year 2000, this percentage can exceed 80%. As a result, addressing the missing keywords becomes our primary concern, which we will address in the subsequent section.

These observations highlight the specific columns where missing values are prevalent and provide insights into their underlying causes. The focus now shifts towards addressing the substantial number of missing author keywords, considering their high prevalence in the datasets.

## 4.3 Data Preparation

Effective data preparation is essential for successful entity resolution and its subsequent phases. In this section, we will delve into the measures employed to ensure that the data is clean, well-formatted, and ready for further analysis. Additionally, we will discuss the creation of a golden dataset for evaluation purposes.

### 4.3.1 Data Cleaning

Cleaning the data involves applying various techniques to address inconsistencies, errors, and noise present in the dataset. This step aims to enhance the quality and reliability of the data. During data cleaning, the following actions were performed:

- **Standardizing Author Names:** Standardizing author names is a crucial aspect of data cleaning. This involves ensuring consistent formatting and removing extraneous characters. For example, the name "Wolff, J. Michael" is standardized as "WolffJMichael" by removing spaces and punctuation. This step ensures that author names are represented uniformly, enabling accurate matching.

- **Dealing with Parsing Issues:** Parsing issues can arise during data extraction, leading to erroneous values in certain columns. For example, an affiliation column might contain numerical values like "2021," indicating an error during data extraction. These cases are not frequent in the datasets, so for now we will just drop them. ??
- **Handling Missing Data:** Missing data represent a common challenge in data analysis projects. In our datasets, the author keywords column poses a significant concern as it contained a considerable number of missing values. To address this issue, we used Spark-NLP library, more specifically the YAKE keyword extractor model, to generate keywords from the abstract when the author keywords were missing.

## YAKE

YAKE (Yet Another Keyword Extractor) is an open source Python library designed for keyword extraction and text analysis. It enables users to effortlessly identify and extract the most important keywords or key phrases from text documents. YAKE boost multilingual capabilities, independence from annotated data, adaptability across different domains and languages, precision at the single-document level, and impressive computational efficiency, making it ideal processing extensive text data like research paper abstracts.

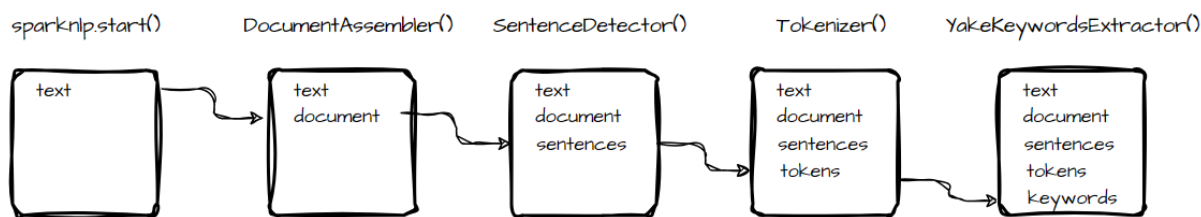


Figure 4.3: YAKE workflow pipeline

The code to apply all the aforementioned operations can be seen in the listing below:

```

1 from sparknlp.base import *
2 from sparknlp.annotator import *
3
4 def preprocess_data(df, normalize_name: bool = False):
5     """
6     Performs data processing like tokenizing, dformatting null values, normalize, etc.
7     @param df: The Pyspark dataframe to processes.
8     @param normalize_name: A Boolean to normalize author names or not.
9
10    @return: A new dataframe
11    """
12
13    # Normalize author names (remove spaces, and capitalization)
14    if normalize_name:

```

```

15     df = df.withColumn("author_name_clean", lower(trim(regex_replace(col("
author_name"), '[.]', ''))))
16
17     # Preprocess affiliations (parsing)
18     extract_names = F.udf(lambda affiliations, affils: [affil['affilname'] for affil
in affils if affil['afid'] in affiliations] or None, ArrayType(StringType()))
19     df = df.withColumn('author_affil_name', extract_names(F.col('author_affil'), F.
col('affils')))
20
21     # Format null values in affiliations
22     df= df.withColumn('author_affil', F.when(F.size(F.col('author_affil')) == 1, F.
when(F.col('author_affil')[0] == '', None).otherwise(F.col('author_affil'))).
otherwise(F.col('author_affil')))
23
24     # Add Index in each row (to identify later)
25     df = df.withColumn("row_index", monotonically_increasing_id())
26
27     # Define the keywords extractor pipeline
28     pipeline = Pipeline().setStages([
29         documentAssembler,
30         sentenceDetector,
31         tokenizer,
32         keyword_extractor
33     ])
34
35     # Apply pipeline
36     df = pipeline.fit(df).transform(df)
37
38     return df

```

Code Listing 4.1: Function to preprocess the dataframe

### 4.3.2 Data Transformations

Data transformations encompass a range of techniques aimed at converting the data into a suitable format for further analysis and entity resolution. These transformations involve tasks such as tokenization, segmentation, and reformatting of the dataset structure. Key aspects of data format transformations include:

- **Tokenization:** Tokenization is the process of breaking down textual data into individual tokens or words. In the context of entity resolution, tokenization is applied to attributes such as titles, abstracts, and keywords. Additionally, removing stopwords is often performed during tokenization. Stopwords are common words (e.g., "the," "is," "and") that do not carry much semantic meaning and can be safely excluded from analysis. For example, the keyword "machine learning in healthcare" is tokenized into ["machine", "learning", "healthcare"], enabling more precise comparisons and disambiguation.
- **Segmentation:** Segmentation involves dividing data into meaningful units or segments. In

the case of authors' affiliations, segmentation can be applied to associate each author with their respective affiliation, in case When an author has multiple affiliations, segmentation is even more important to ensure that the affiliations don't get mixed up.

- Reformatting the Dataset structure: The shape and structure of the dataset are reformatted to align with the requirements of entity resolution algorithms. Specifically, author names and their corresponding authorship relationships are extracted, and the dataset is transformed to represent instances where an author's name appears in a specific paper. Each row in the reformatted dataset indicates the presence of an author, denoted as "*Author<sub>i</sub>* is listed as an author for *paper<sub>j</sub>*." This reformatted dataset captures the relationships between authors, papers, and their associated metadata.

The code to applying these transformations is as follows

```

1 import pyspark.sql.functions as F
2
3 def transform_data(df):
4     """
5     Performs a list of transformations to create the new author-paper dataframe
6     that is compatible for the entity resolution task
7     @param df: The Pyspark dataframe to processes
8
9     @return: A new dataframe
10    """
11
12    #Tokenizing string fields
13    df = df.withColumn("author_names", F.split(F.col("author_names"), ";"))
14    df = df.withColumn("author_ids", F.split(F.col("author_ids"), ";"))
15    df = df.withColumn("author_afids", F.split(F.col("author_afids"), ";"))
16    df = df.withColumn("afid", F.split(F.col("afid"), ";"))
17    df = df.withColumn("affilname", F.split(F.col("affilname"), ";"))
18
19    # Segmentation of compound fields (lists)
20    df = df.withColumn("authors", F.arrays_zip(df["author_names"], df["author_afids"],
21    ], df["author_ids"]))
22    df = df.withColumn("affils", F.arrays_zip(df["afid"], df["affilname"]))
23    df = df.withColumn("exploded_author", F.explode(df["authors"]))
24
25    # Expanding rows and restructuring the dataframe
26    df = df.withColumn("author_name", F.col("exploded_author.author_names"))
27    df = df.withColumn("author_id", F.col("exploded_author.author_ids"))
28    df = df.withColumn("author_affil", F.col("exploded_author.author_afids"))
29    df = df.withColumn("author_affil", F.split(F.col("author_affil"), "-"))
30
31    df = df.withColumn("other_authors", F.array_except(F.col("author_names"), F.
32    array(F.col("author_name"))))
33
34    cols = ["eid", "author_id", "author_name", "author_affil", "other_authors",
35    "title", "publicationName", "description", "authkeywords", 'affils',
36    "description_embeddings"]
37
38    # Selecting only the required columns (drop everything else)
39    df = df.select(cols)
40
41    return df

```

Code Listing 4.2: Function to apply data transformations

### 4.3.3 Reference Dataset

In order to evaluate and assess the effectiveness of author name disambiguation techniques, a high-quality reference dataset, referred to as the "golden dataset," was curated. Collaborators



and experts in the project contributed to the creation of this dataset, employing reliable author identifiers and ensuring the provision of accurate and dependable ground truth annotations for comparison purposes.

Each author in the database has an ID, however because of the duplication problem authors can have multiple IDs. Using Scopus API, you can query all the IDs that belong to certain Author (Scopus recently started their own attempt at disambiguating authors), and then normalize them into a single one. However that query is expensive since it has to be made for each other found, which would cause the API to reach the monthly query limit easily. So the reference dataset was created by using this approach for all authors in the 3 csv files for 2020, 2021, and 2022. Key characteristics of the golden dataset include:

- **Total rows:** the dataset comprises a total of 105,403 rows, representing the number of distinct instances or entries contained within.
- **Unique authors:** a collection of 58,674 unique authors is included in the dataset. Each author represents an individual entity that requires disambiguation.
- **Author names:** across the dataset, there are 63,840 unique author names associated with the authors, signifying the different variations and aliases used.
- **Research papers:** the dataset encompasses 17,342 research papers, with each paper serving as a distinct publication linked to one or more authors.

author_name	author_affiliation	coauthor_names	title	abstract	keywords
Ovberedjo, Martin	WHO Eritrea Country Office	['Mebrahtu, Goitom', 'Moleki, Mary M.'],	Antihypertensive medication adherence and associati	Background: Recent research suggests that poor adherence to ant	['Eritrea', 'Hypertension', 'Knowledge', 'Medication adherence']
Skute, Elizabeth C.	Planetary Science Institute	['Ye, Cheng', 'Glotch, Timothy D.'],	Orientation Averaged Visible/Near-Infrared and Mid-I	The hydrous Ca-sulfates, gypsum and bassanite, have been detect	['bioprinting', 'gene expression', 'gravity', 'neural stem cell', 're
Yermami, Messaouda	University of Carthage, Institut Pr	['Chkir, Najiba', 'Zouari, Kamel']	Stable Isotopes for Sustainable Management of Agric	Water stable isotopes have been used since the 61&#x2013	['Agricultural basin', 'Hydrogeochemistry', 'Nitrate', 'Water/rock
Perrodin, Delphine	Osservatorio Astronomico di Cagliari	['Melis, Andrea', 'Antonietti, Nicol&#x2013;B']	Involvement of the Sardinia Radio Telescope in the Br	The Breakthrough Listen (BL) Program is the most comprehensive	['Breakthrough Listen', 'Sardinia Radio Telescope', 'SETI']
Hutabarat, Johannes	Universitas Padjadjaran	['Prasetyo, Rasi', 'Daud, Yunus', 'Hendan	Fluid geochemistry and Isotope compositions to delin	Wayang Windu geothermal field, which is situated in quaternary v	['gas equilibria', 'geochemistry', 'geothermal', 'Isotopes']
Putrevu, Deepak	Indian Space Research Organization	['Bhiravarasu, Sriram S.', 'Chakraborty, T	Chandrayaan-2 Dual-frequency Synthetic Aperture Ra	The Dual-Frequency synthetic aperture radar (DFSAR) system man	['Lunar craters (949)', 'Lunar features (953)', 'Lunar science (972)
Putrevu, Deepak	Indian Space Research Organization	['Bhiravarasu, Sriram Saran', 'Das, Anup']	Scattering Mechanisms Associated with High Circular	The crater floors and walls of youngest impact craters on the Moo	['Chandrayaan-2', 'Craters', 'Moon', 'Polarimetry', 'Scattering']
Putrevu, Deepak	Indian Space Research Organization	['Bhattacharya, Anamiya', 'Pandey, Dhar	An Improvised Non-Invasive Method with Clutter Rem	This paper presents a free-space reflection measurement techniq	['Biofilms', 'Biosignatures', 'Element uptake', 'Hot springs', 'Sili
Pshirkov, Maxim	Lomonosov Moscow State University	['Tinyakov, Peter', 'Popov, Sergei']	Astroparticle physics with compact objects	Probing the existence of hypothetical particles beyond the Standar	['Axions', 'Dark matter', 'Neutron stars', 'White dwarfs']
Havig, Jeff R.	University of Minnesota Twin Cities	['Kuether, Joshua E.', 'Gangidine, Andrew	Hot Spring Microbial Community Elemental Composi	Hydrothermal systems host microbial communities that include so	['Biofilms', 'Biosignatures', 'Element uptake', 'Hot springs', 'Sili
Havig, Jeff R.	University of Minnesota Twin Cities	['Gangidine, Andrew', 'Walter, Malcolm']	Trace element concentrations associated with mid-pa	Identifying microbial fossils in the rock record is a difficult task be	['Biogeochemistry', 'Biosignatures', 'Hot springs', 'Microfossils',
Remesh, N.	Indian Space Research Organization	['Ramanan, R. V.', 'Lalithambika, V. R.'],	A Novel Indirect Scheme for Optimal Lunar Soft Landi	The problem of precise and soft lunar landing in a pre-specified ta	['Differential evolution', 'Differential transformation', 'Indirect app
Remesh, N.	Indian Space Research Organization	['Ramanan, R. V.', 'Lalithambika, V. R.'],	Fuel-optimal and Energy-optimal guidance schemes f	Two guidance schemes (i) fuel-optimal (ii) energy-optimal to reali	['Differential transformation etc.', 'Guidance', 'Indirect approach',
Zykin, Nikolay N.	Gazprom	['Tokarev, Igor V.', 'Vinograd, Natalia A.'],	Monitoring of stable isotopes ( $\delta^{13}C_{org}$ and $\delta^{15}N_{org}$ )	The isotopic composition of oxygen ( $\delta^{18}O$ ) and hydrogen ( $\delta^2H$ ) of	['Atmospheric precipitation', 'Climate', 'Isotopic composition of ox

Figure 4.4: Example of rows in the reference dataset

To ensure the integrity of the evaluation process, the reference dataset was further divided into separate train and test sets. This separation guarantees that the blocking and pairwise comparison steps are conducted independently on each set, eliminating any potential data leakage during the subsequent classification and clustering stages.

## 4.4 Blocking Scheme

After generating the golden dataset, blocking is the next step in the ER process, as discussed in section (2.5) blocking aims to group records together that are likely to be a match, while ensuring a balance between capturing all possible matching (recall) and minimizing the number of false matches, i.e. increasing the precision. we will look at the evaluation result of some common blocking schemes, and discuss how we can further optimize them for efficiency consideration.

### 4.4.1 Evaluation

Different indexing approaches are applied to the datasets. The results of the evaluation of these approaches on the train set are provided in the table below. <sup>†</sup> But first, some statistics regarding the train dataset and blocking phase are worth mentioning:

- **Total number of comparisons:** There are 5,55 b comparisons in the comparison space of the train set. This represents the size of the space in which potential matches are evaluated.
- **Number of matches:** Out of these comparisons, only 288,963 pairs are actual matches, indicating instances where authors with similar or identical names were correctly recognized as the same person.

Method	PC	PQ	# of comparisons
signature Blocking	95.51%	29.27%	942,657
prefix_array( $l_{min} = 5$ )	98.34%	5.1%	5,567,565
sorted_neighborhood( $t = 0.7$ )	94.40%	13.49%	2,021,713
imp_qgrams( $q = 2, t = 0.9$ )	62.51%	76.18%	237,117

Table 4.1: Blocking Phase results

### 4.4.2 Block Refinement

We can see from the blocking results that in some instances we get a higher PC (recall) values, but the PQ values is low. That means the blocking scheme generates a lot of false pairs, which would need to be compared in details later using the similarity function, this would obviously have negative impact on the efficiency (speed) of the process. To tackle this problem, we use a block filtering approach with a threshold value of 0.8.

---

<sup>†</sup>Prefix array: is analogous to suffix array blocking, except it stores prefixes(beginning) of minimum length in an inverse suffix array structure

Method	PC	PQ	# of comparisons
signature blocking	95.51%	29.27%	942,657
prefix_array( $l_{min} = 5$ )	95.71%	16.28%	1,698,107
sorted_neighborhood( $t = 0.7$ )	94.40%	13.49%	2,021,713
imp_qgrams( $q = 2, t = 0.9$ )	58.81%	87.04%	195,270

Table 4.2: Blocking Phase results (after block refinement)

An important point to note here is that signature blocking and sorted neighborhood methods do not benefit from block filtering. This is because each record exists in only one block only, defined by a single BKV, in contrast to other methods where a record can be linked to multiple blocks. Therefore, when selecting a blocking scheme, it’s essential to consider whether the blocks can be further enhanced for scalability<sup>‡</sup>. In this regard, we will utilize the **prefix\_array** approach in conjunction with block filtering

## 4.5 Pairwise Comparison

After the blocking phase, and generating the pairs. we can move ahead to the detailed comparison between records. This stage utilizes the remaining metadata associated with the publications and authors to make informed decisions about potential matches or mismatches.

We experimented with different similarity functions (see previous section), to find the best method to be applied for each attribute, the final feature set is described below:

- **Author Name Similarity:** High similarity between the names of two authors suggests that they refer to the same entity. The Jaro-Winkler method is used to calculate the similarity between author names.
- **Affiliation Similarity:** If two ambiguous authors share the same affiliation, it is likely that they refer to the same entity. However, affiliations are also susceptible to ambiguity. To address this, we employ the hybrid Monge-Elkan similarity function (with Jaro-winkler as an internal similarity function). This function performs well for names with multiple words, including abbreviations.
- **CoAuthors Similarity:** The presence of common co-authors between two ambiguous authors across different publications can serve as evidence that they are the same author. Again, the Monge-Elkan similarity function, similar to the affiliation similarity approach, can be utilized to compare co-author names and assess their similarity.
- **Abstract and title similarity:** The paper title and abstract are important metadata to help disambiguate author, since authors usually publish in the same scientific domain. to compare these attributes, we use BERT sentence embeddings to create a high dimensional vector representation of abstracts/titles. More specifically the SciBERT pretrained

---

<sup>‡</sup>Although signature blocking produced better result than prefix array, it was only evaluated on 3 files, and it will not scale well when using the more data

embeddings that were trained on large corpuses of scientific papers, we then apply cosine similarity between the embeddings.

- **Keywords Similarity:** When publications associated with ambiguous authors share several common keywords, it's highly likely that they belong to the same author. Authors generally focus on specific fields and use similar keywords in their work. For this metrics, because keywords are often represented in different ways, like "asteroids" vs "asteroid" or "asteroid impact", we need a semantic representation of these keywords, therefore we will use another pretrained BERT word embeddings for the keywords and calculate their cosine similarity.

```

1 from deduplicator.train import *
2
3 def create_embeddings(df:DataFrame, column_name:str, Type:str):
4     """
5     Calculate the Word/Sentence BERT Embeddings of abstracts/keywords
6     @param df: The Pyspark dataframe to process
7     @param column_name: The containing the abstracts/keywords
8     @param Type: the type of pipeline to use ["word" or "sentence"]
9
10    @return: A dataframe
11    """
12    if type=="word":
13        model = BERTWordPipeline.fit(df, InputCol=column_name)
14        df = model.transform(df)
15
16    if type=="sentence":
17        model = BERTSentencePipeline.fit(df, InputCol=column_name)
18        df = model.transform(df)
19
20    return df
21
22

```

Code Listing 4.3: Function to create embeddings

As explained above all blocking methods used above would have high number of false matches, which results in the creation of an unbalanced dataset (see Annexe). To address this issues we will we randomly sampled there are several methods we can use, for now we used the Random Undersampling approach to balance the number of instances per class.

	# d'instances
False Match	1 413 088
True Match	274 737

Table 4.3: Class distribution of pairs before sampling

	# d'instances
False Match	274 737
True Match	274 737

Table 4.4: Class distribution of pairs after sampling

## 4.6 Classification

After generating the feature vectors from the pairwise comparisons step, several classifiers were trained to identify the matching pairs, these classifiers were evaluated on the train set, which was also subsequently split into a 90-10% split. We also used 10-fold cross-validation, to see if there is any over-fitting happening. The result are below:

	F-1	
	Train	Train+CV
Gaussian NB	90.1 %	89.6%
Logistic Regression	91.4%	90.3%
Support Vector Machines	93.7%	93.1%
Decision Trees	98.49 %	93.45%
Random Forest	99.10 %	94.36%

Table 4.5: Classification results

Results analysis:

- It is apparent from the results that the Decision trees and random forrest models are overfitting, which is likely due to their complexity compared to the limited set of feautres used.
- GaussianNB, Logistic Regression and SVM do not seem to be overfiitng, However the first two have low result.
- We will retain the SVM model for optimazation due to it's consistent performance, and apply some regularization to the Random forest model

We also evaluate the results of the new models on both the train and test set as shown in table, as for tuning the hyperparameters we used a GridSearchCV approach:

	F-1		
	Train	Train + CV	Test
Best-SVM	94.4 %	94.0 %	93.2%
Best-Random-Forest	96.8 %	96.4 %	95.4%

Table 4.6: Classification results (after parameters tuning)

The results of this Random Forest model do not show signs of severve overfitting, same with the SVM model. For now we will retain both model for the clustering steps.

```

1 from deduplicator.train import *
2 from pyspark.ml.evaluation import Evaluator
3 from pyspark.ml import Model
4 from pyspark.ml.tuning import CrossValidator

```

```

5
6 def tune_model(train_data:DataFrame, model:Model, params:list[ParamMap], cv:int,
  metric:Evaluator):
7
8     """
9     Performs hyperparams tuning a the model, w/o cross-validation
10    @param train_data: The Pyspark dataframe to process
11    @param model: The Pyspark mlib model to tune
12    @param params: A parameter grid to use with gridSearch Algorithm
13    @param cv: number of cross validation folds
14    @param Evaluator: The metric to evaluate the models, and chose the best one
15
16    @return: A new tuned and trained model
17    """
18    crossval = CrossValidator(estimator=model,
19                              estimatorParamMaps=params,
20                              evaluator=metric,
21                              numFolds=cv)
22
23    # Run cross-validation, and choose the best set of parameters.
24    cvModel = crossval.setParallelism(4).fit(train_data)
25
26    # Get the best model from the CV
27    bestModel = cvModel.best_model
28
29    return bestModel, model_metrics

```

Code Listing 4.4: Function to tune model's hyperparameters

## 4.7 Clustering

The ultimate goal of author name disambiguation extends beyond predicting pairwise matches and involves the formation of clusters that group together matching records referring to the same entity. For instance, we want to identify all papers written by a specific author like "Jade, Williams" who may have been referred to by various aliases such as "Jade, Wil" or "Jade, W."

Clustering can serve as a post-processing step following pairwise classification. The basic idea is to transform the predicted probabilities from the classifier into distances. Since the probability of two records being a match is an equivalent measure of their similarity, we can therefore define the distance between any pair of records as follows:  $d_{ij} = f(p_{ij})$ , where  $p$  represents the probability of records  $i$  and  $j$  being a match, and  $f(x) = 1 - x$  is a monotonically decreasing function of the probabilities (similarities).

Once we have computed these distances, we can construct the  $D^*$  matrix, where  $D^*[i, j] = d_{ij}$ . Finally, we can apply clustering approaches to this distance matrix, enabling the grouping of similar records together.

Method	Classifier	ACP	Pair_F1
HAC	SVM	91.1%	94.9%
	RF	92.6%	96.4%
Lou- vain	SVM	90.3%	93.2%
	RF	90.9%	93.5%
CC	SVM	88.7%	87.9%
	RF	88.8 %	86.4%

Table 4.7: Clustering results

Results analysis:

- **Random Forest vs SVM:** Across all clustering methods, the Random Forest classifier consistently achieves higher Classifier Accuracy (ACP) and Pairwise F1 Score (Pair\_F1) compared to the SVM classifier. This suggests that Random Forest is more effective in predicting matches between author records.
- **HAC Leads to Highest Accuracy:** Among the clustering methods, hierarchical agglomerative clustering combined with Random Forest achieves the highest accuracy (92.6% ACP and 96.4% Pair\_F1). This indicates that this combination is particularly effective in grouping similar author records together.
- **Accuracy vs Efficiency:** It's important to consider the trade-off between accuracy and computational efficiency when choosing a clustering method. While hierarchical agglomerative clustering performs well, it is computationally more intensive -  $O(\log(n)n^2)$  - compared to the other two algorithms.

## 4.8 Model Deployment

In this section, we will talk about the methodology of model deployment, and how it will be used in the LIST project. First we will begin by examining the structure of the code repository, then we will talk a little bit about Apache Spark, its core components, and its deployment modes. Finally we will discuss the full python script that will be run in the Spark clusters, as well as the structure of the modules and dependencies needed to run the job.

### The Structure of the Code

The solution code is accessible through the company's private GitLab repository, encompassing various modules within separate .py files, each responsible for distinct facets of the entity resolution process. Additionally, the repository contains essential components such as:

- The 'data' folder housing .csv files.
- The 'deduplicator' folder, containing separate .py files.

- The 'config.py' file containing configurations pertinent to the Spark job, including session parameters and function parameters.
- The 'requirements.txt' file specifying the project's requisite dependencies.
- the Neo4j connector employed for facilitating data transfer to and from the Spark session.
- A comprehensive README file offering project documentation and guidance.
- The 'main.py' file serving as the central entry point for execution and coordination of the resolution process.

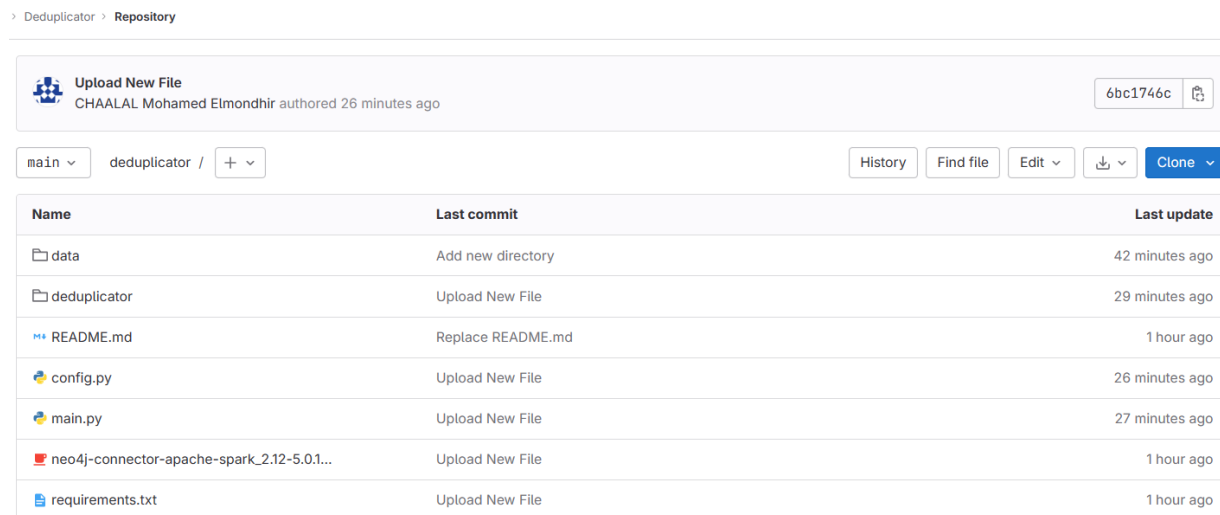


Figure 4.5: The Gitlab repository containing the code

For more details on the full code inside each module, see (Annex code).

## Apache Spark

Apache Spark is a powerful open-source, distributed computing framework designed for processing and analyzing large datasets in a distributed and parallel manner. It was developed to overcome the limitations of the Hadoop MapReduce model by providing faster and more flexible data processing capabilities.

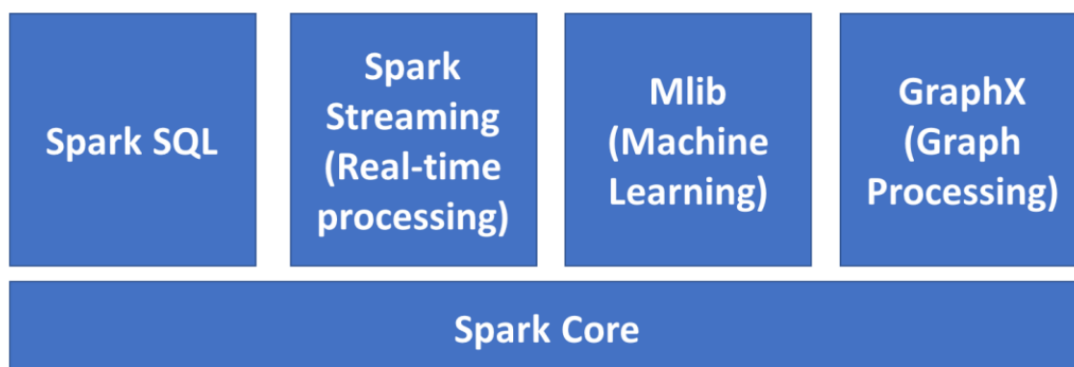


Figure 4.6: The components of Apache Spark



Apache Spark consists of several key components and provides support for different programming languages, including Python, R, Java and Scala. The main components are:

- **Spark Core:** The foundation of Apache Spark is the Spark Core. It provides essential functionality like task scheduling, memory management, fault recovery, and distributed data processing. At the core, Spark represents data as Resilient Distributed Datasets (RDDs), which are immutable distributed collections of data.
- **Spark SQL:** Spark SQL is a Spark module for structured data processing. It allows you to execute SQL queries on structured data and seamlessly integrate with other Spark components. Spark SQL also supports reading and writing data in various formats, including Parquet, JSON, and CSV.
- **Spark Streaming:** Spark Streaming is used for processing real-time data streams. It ingests data in small, discrete batches and performs batch processing on them, making it suitable for applications like log processing and monitoring.
- **Spark MLlib:** Spark MLlib is Spark's machine learning library, offering various machine learning algorithms and tools for building and evaluating models on large datasets. It provides support for classification, regression, clustering, and more.
- **Spark GraphX** Spark GraphX is a library for graph processing, enabling graph computations and graph-parallel algorithms. It is well-suited for analyzing and processing graph-structured data. Although as of this date, the library is only available to use through the Scala Driver.

### Spark deployment modes

Spark offers several running modes to accommodate various deployment scenarios, each with its advantages and use cases. These modes include:

- **Local mode:** the simplest way to run Spark is using local mode. this means the Spark job is ran on a single machine, typically for development, testing, and debugging purposes. In local mode, Spark runs in a single JVM (Java Virtual Machine) process on your local machine.
- **Standalone Mode:** Standalone mode is Spark's built-in cluster manager. It's a basic cluster manager that can be used to run Spark applications on clusters. It's suitable for small to medium-sized clusters.
- **Apache Hadoop YARN:** YARN (Yet Another Resource Negotiator) is the resource management framework in the Apache Hadoop ecosystem. Spark can be run on YARN clusters, which allows for efficient resource management and sharing resources with other Hadoop ecosystem components. It's well-suited for larger clusters and organizations already using Hadoop.

For our research and development purposes, we have chosen to deploy the code in Standalone mode. This selection enables us to simulate the conditions of running the code on clusters, allowing us to harness the full capabilities of Spark and parallelism. The job will run on my laptop that boasts an Intel Core i7-10750H CPU with a clock speed of 2.60GHz, along with 16.0 GB of RAM and 12 processing cores.

## 4.9 Step-by-Step Workflow

In this part we will explore the comprehensive step-by-step workflow to successfully run the Spark job in Standalone mode, while also ensuring the inclusion of essential dependencies.

### Step1: Create a Python Virtual Environment

Before initiating the Spark job, it's essential to establish a Python virtual environment to isolate project dependencies. The following commands should be executed in a terminal within the project directory to create the venv and activate it:

```
Microsoft Windows [Version 10.0.22621.2134]
Copyright (c) 2022 Microsoft Corporation. All rights reserved.

c:\User\Azus\deduplicator>python -m venv venv1

c:\User\Azus\deduplicator>venv1\Scripts\activate.bat

(venv1) C:\Users\Azus\project>
```

### Step 2: Install Dependencies from requirements.txt

After having created the virtual environment we need to install the project dependencies from the venv terminal using the pip command.

```
(venv1) C:\Users\Azus\project>pip install -r requirements.txt
```

### Step 3: Zip the Dependencies

In this step, we package all the dependencies into a zip file. This zipped package is vital for efficient distribution to all worker nodes in the Spark cluster, ensuring that each worker has access to the necessary codes and functions throughout the job execution.

```
(venv1) C:\Users\Azus\project> zip -r deduplicator.zip deduplicator/
```

```
(venv1) C:\Users\Azus\project>(cd venv1/Lib/site-packages/ && zip -r \
dependencies.zip .* && mv dependencies.zip ../../../../)
```

#### Step 4: Initializing Spark Master & Worker Nodes:

The first step in deploying Spark in Standalone mode on your laptop is to initialize the Spark Master. This can be achieved by running the following command:

```
Microsoft Windows [Version 10.0.22621.2134]
Copyright (c) 2022 Microsoft Corporation. All rights reserved.

c:\User\Azus>spark-class org.apache.spark.deploy.master.Master
```

After successfully initializing the Spark Master, the next step is to launch the Spark Workers, for this task we will use 3 worker nodes, while allocating 4 processing cores and 2gb of RAM for each worker. This is done with the following command:

```
Microsoft Windows [Version 10.0.22621.2134]
Copyright (c) 2022 Microsoft Corporation. All rights reserved.

c:\User\Azus>spark-class org.apache.spark.deploy.worker.Worker \
spark://127.0.0.1:7077 --cores 4 --memory 2g
```

#### Step 5: Running the Spark Job with spark-submit

In conclusion, we execute the Spark job in Standalone mode using the spark-submit command. We configure it with specific parameters, including additional settings for the number of cores and memory allocation for each driver. Furthermore, we ensure that the job has access to essential resources by including both the dependencies.zip file and deduplicator.zip as supplementary assets through the `-py-files` option

```
(venv1) C:\Users\Azus\project>spark-submit --master spark://127.0.0.1:7077
--driver-memory 8g --executor-memory 2 \
--py-files dependencies.zip ,\
deduplicator.zip \
main.py
```

#### Step 7: Monitor Job Progress Using SparkUI

After submitting the Spark job, we can closely monitor its progress and performance using Spark's built-in web-based interface, SparkUI. To access SparkUI, we open a web browser and navigate to the following address: <http://127.0.0.1:4040/>

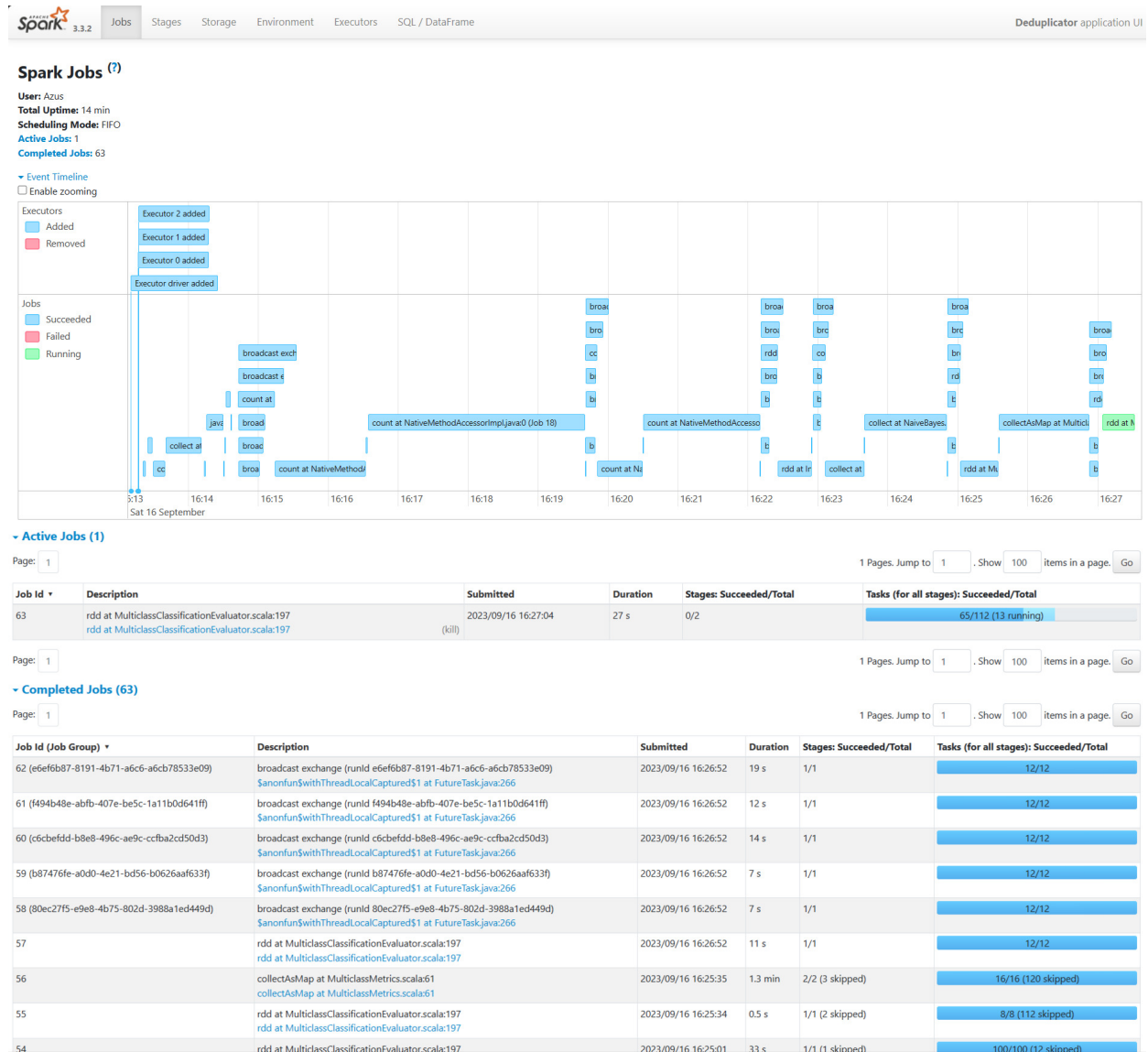


Figure 4.7: The timeline of the spark job execution

After the job is done, we can find the new duplicated .csv files in a new "output" folder, we can also visualize the created clusters in the Neo4j applications

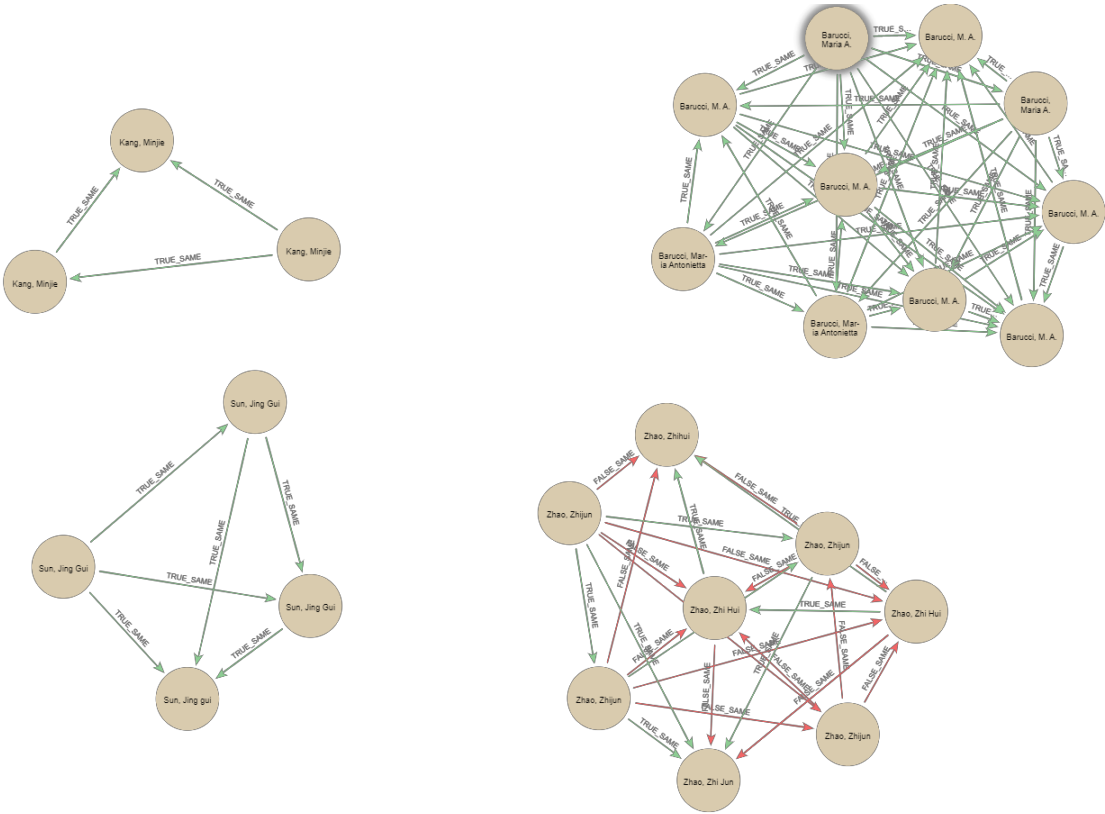


Figure 4.8: Resultat de regroupement graphic

# General Conclusion

In the culmination of this internship at LIST, it is imperative to provide a comprehensive conclusion that encapsulates the nuanced journey we undertook in addressing the complex challenge of Author Name Disambiguation within large databases. This chapter aims to reflect upon each phase or stage of the disambiguation process while also highlighting our involvement in the reviewing of multiple research papers, underscoring the enriched nature of our research experience.

The Data Pre-processing phase laid the groundwork for our disambiguation efforts. Here, we delved into the nitty-gritty of data quality enhancement, standardization, and transformation. We addressed missing data, inconsistencies, and outliers with important care

The Indexing/Blocking phase, focusing on computational optimization, has had implications beyond disambiguation. By judiciously partitioning the dataset into manageable blocks, this phase has significantly reduced computational complexity. Which was extremely important for resource-efficient data processing across the organization.

Our exploration of Pairwise Similarity Calculation enriched our analytical toolkit. We evaluated and compared various similarity functions, recognizing the importance of choosing the right one for the data type. Beyond the technicalities, this phase underscores the challenge of selecting the most suitable similarity function. It's about understanding the data and the problem at hand to make an informed choice.

In the Classification phase, we encountered the challenge of data biases and class imbalance. Understanding these biases and the presence of majority classes is crucial. It's not just about building classifiers; it's about recognizing that imbalanced data can skew results. Another common challenge in Machine Learning in general (and certainly in our case) is overfitting. To address potential overfitting, we tested different classifiers with different levels of complexities, coupled with cross-validation. This approach allowed us to assess how each classifier handles the large data load and evaluate the performance on the validation set.

Selecting the right model for the task and fine-tuning its parameters were critical aspects of the Classification and Clustering phases. The task revolved around understanding the nuances of different algorithms and making informed choices based on data characteristics. Parameter tuning ensured that our classifiers operated at peak performance, striking at the balance between complexity(overfitting) and generalization(underfitting).

Integral to our success was the transformative impact of leveraging Spark and Neo4j. Spark empowered us with distributed data processing capabilities, allowing us to scale our analyses

efficiently. Neo4j facilitated graph-based data modeling, enabling us to create and traverse authorship networks with ease. These technologies were instrumental in handling the extensive datasets and complex relationships inherent in our disambiguation task.

Notably, our disambiguation framework has extended its initial project scope, finding a new purpose as an internal tool at LIST. Within the organization, it serves as a part of an instrument for researcher performance evaluation, disambiguating researchers' names and ensuring that their contributions to the body of research are accurately accounted for. This internal application extends to the evaluation of quarterly and yearly Key Performance Indicators (KPIs), including metrics like the number of publications.

# Bibliography

- [1] Pruski, Cedric & Deladiennée, Louis & Scolan, Emmanuel & Silveira, Marcos. (2023). Une plateforme de management des connaissances pour le domaine des ressources spatiales.
- [2] IBM. "What is big data?". <https://www.ibm.com/analytics/hadoop/big-data-analytics>
- [3] Christen, P. "A survey of indexing techniques for scalable record linkage and deduplication." *IEEE Transactions on Knowledge and Data Engineering* 24.9 (2012): 1537-1555
- [4] Fellegi, I. and Sunter, A. (1969) A Theory for Record Linkage. *Journal of the American Statistical Association*, 1969, 64(328), 1183-1210.
- [5] A. Adelstein, "Record linkage techniques in studies of the aetiology of cancer," *Proceedings of the Royal Society of Medicine*, vol. 61, no. 7, p. 732, July 1968
- [6] H. B. Newcombe, J. M. Kennedy, S. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, Oct 1959
- [7] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *International Conference on Knowledge Discovery and Data Mining*, pages 269–278, 2002.
- [8] M. A. Hernandez and S. J. Stolfo, "The merge/purge problem for large databases," in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 127–138
- [9] Clark, D.E.: Practical introduction to record linkage for injury research. *Injury Prevention* 10, 186–191 (2004)
- [10] Herzog, T., Scheuren, F., Winkler, W.: *Data quality and record linkage techniques*. Springer Verlag (2007)
- [11] F. Naumann and M. Herschel, *An introduction to duplicate detection*. Morgan and Claypool Publishers, 2010.
- [12] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.
- [13] P. Christen and K. Goiser, "Quality and complexity measures for data linkage and deduplication," *Studies in Computational Intelligence*, vol. 43, 01 2007.



- [14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, “Duplicate record detection: a survey,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 1, p. 1–16, Jan. 2007
- [15] Tim Churches, Peter Christen, Kim Lim, and Justin Xi Zhu. Preparation of Name and Address Data for Record Linkage Using Hidden Markov Models. *BMC Medical Informatics and Decision Making*, 2(1):9, 2002.
- [16] Hassanzadeh, Oktie Miller, Renée. (2009). Creating probabilistic databases from duplicated data. *VLDB J.* 18. 1141-1166. 10.1007/s00778-009-0161-2.
- [17] Neil G. Marchant. *Statistical Approaches for Entity Resolution under Uncertainty*, 2021
- [18] Lehti, Patrick & Fankhauser, Peter. (2006). Unsupervised Duplicate Detection Using Sample Non-duplicates. *Lecture Notes in Computer Science*. 4244. 136-164. 10.1007/11890591\_5.
- [19] Gravano, Luis & Ipeirotis, Panos & Jagadish, H. & Koudas, Nick & Muthukrishnan, Senthilmurugan & Srivastava, Divesh. (2003). Approximate String Joins in a Database (Almost) for Free.
- [20] Christen P. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 2011, 24(9): 1537-1555.
- [21] Aizawa A, Oyama K. A fast linkage detection scheme for multi-source information integration. In *Proc. the 2005 International Workshop on Challenges in Web Information Retrieval and Integration*, Apr. 2005, pp.30-39
- [22] Damerau, F.J.: A technique for computer detection and correction of spelling errors. *Communications of the ACM* 7(3), 171–176 (1964)
- [23] Temple F Smith, Michael S Waterman, et al. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.
- [24] Winkler, W.E., Thibaudeau, Y.: An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. *Tech. Rep. RR1991/09*, US Bureau of the Census, Washington, DC (1991)
- [25] Winkler, W.E.: Methods for record linkage and Bayesian networks. *Tech. Rep. RR2002/05*, US Bureau of the Census, Washington, DC (2001)
- [26] Monge, A.E., Elkan, C.P.: The field-matching problem: Algorithm and applications. In: *ACM SIGKDD*, pp. 267–270. Portland (1996)
- [27] Hernandez, M.A., Stolfo, S.J.: Real -world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery* 2(1), 9–37 (1998)
- [28] Mudgal, Sidharth et al. “Deep Learning for Entity Matching: A Design Space Exploration.” *Proceedings of the 2018 International Conference on Management of Data (2018)*: n. pag.

- [29] Christen, Peter. “Automatic record linkage using seeded nearest neighbour and support vector machine classification.” *Knowledge Discovery and Data Mining* (2008).
- [30] V.S. Verykios, A.K. Elmagarmid, and E.N. Houstis. Automating the Approximate Record Matching Process. *Journal of Information Sciences*, 126(1-4), pages 83-98, July 2000.
- [31] Jaro, M.A.: Advances in record-linkage methodology a applied to matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association* 84, 414–420 (1989)
- [32] Winkler, W.: String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In: *Proceedings of the Section on Survey Research Methods*, pp. 354–359. American Statistical Association (1990)
- [33] Verykios, Vassilios & Elmagarmid, Ahmed & Houstis, Elias. (2002). Automating the approximate record-matching process. *Information Sciences*. 126. 83-98. 10.1016/S0020-0255(00)00013-X.
- [34] Hassanzadeh, O., Miller, R.J. Creating probabilistic databases from duplicated data. *The VLDB Journal* 18, 1141–1166 (2009). <https://doi.org/10.1007/s00778-009-0161-2>

# Appendix A

## Intransitive closure

In entity resolution, "intransitive closure" occurs when there are pairwise matches between records, such as A-B and B-C, but the classifier or matching algorithm doesn't consider A-C as a match. In other words, based on individual pairwise comparisons, A-C appears dissimilar or doesn't meet the threshold for a match, even though there is a transitive relationship through B that suggests A-C should also be considered a match. This can be a significant challenge in entity resolution because it can lead to incomplete or inaccurate results. When the intransitive closure problem is not addressed, related records may remain unlinked, and the true consolidation of entities is not achieved

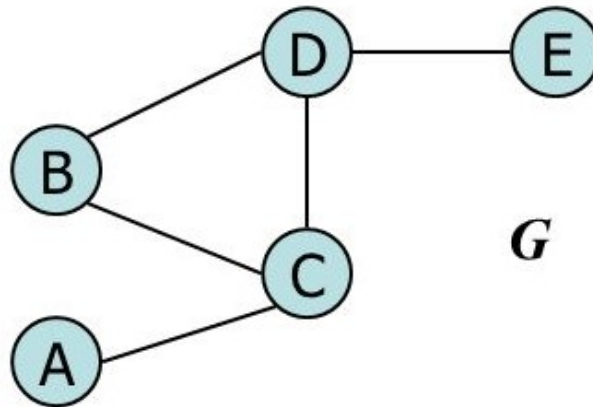


Figure 4.9: Example of Intransitive closer in a graph (nodes a, and e)

## Sous/sur-apprentissage

Underfitting and overfitting are common challenges in machine learning that arise when a model fails to generalize well to unseen data. They can occur when the model's complexity is not appropriately matched to the complexity of the underlying data.

Underfitting refers to a situation where the model is too simple to capture the underlying patterns in the data. It typically occurs when the model has high bias and fails to learn the

complexities of the training data. As a result, the model tends to have high error rates both on the training data and on new, unseen data. Underfitting can lead to poor performance and lack of generalization.

Overfitting, on the other hand, occurs when the model becomes too complex and learns the noise or random variations in the training data. The model starts to memorize the training examples instead of learning the underlying patterns. This results in a low error rate on the training data but a high error rate on new, unseen data. Overfitting is often associated with models that have high variance, where the model is too flexible and tries to fit the training data too closely.

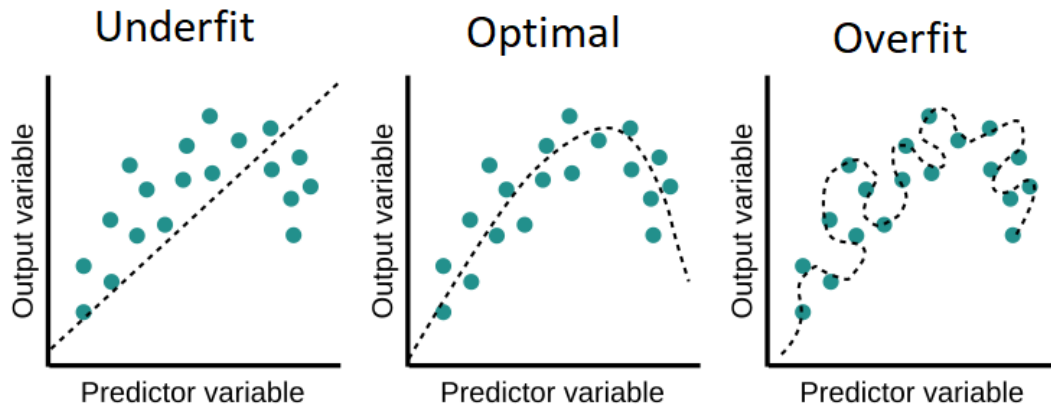


Figure 4.10: The difference between under and over fitting

## Cross Validation

To address overfitting and evaluate the performance of a machine learning model, cross-validation is often used. Cross-validation is a technique that involves splitting the available data into multiple subsets or folds. The model is trained on a subset of the data called the training set and evaluated on the remaining subset called the validation set or test set. This process is repeated multiple times, with different subsets serving as the validation set each time.

One common approach to cross-validation is k-fold cross-validation. In k-fold cross-validation, the data is divided into k equal-sized folds. The model is trained on k-1 folds and evaluated on the remaining fold. This process is repeated k times, with each fold serving as the validation set exactly once. The performance of the model is then averaged over the k iterations to obtain an overall evaluation metric.



Figure 4.11: k-folds cross validation

## Unbalanced Datasets

In machine learning, an unbalanced dataset refers to a dataset where the distribution of class labels is heavily skewed, with one or more classes having significantly fewer instances compared to other classes. Dealing with unbalanced datasets can pose challenges for learning algorithms, as they tend to be biased towards the majority class and may struggle to effectively learn patterns from the minority class(es).

Several methods exist for handling class imbalance, such as Random Under/Over sampling, Synthetic Minority Over-sampling Technique (SMOTE), and ensemble-based approaches. The choice of method depends on the specific characteristics of the dataset and the learning algorithm being used.

## Random Undersampling

A technique that aims to balance the dataset by reducing the number of instances from the majority class to match the number of instances in the minority class(es). Let's denote the original dataset as  $D$  with  $N$  total instances. Let  $D_0$  represent the instances belonging to the majority class (class 0) and  $D_1$  represent the instances belonging to the minority class (class 1). The number of instances in  $D_0$  is denoted as  $N_0$ , and the number of instances in  $D_1$  is denoted as  $N_1$ .

The desired sample size for each class is denoted as  $n$ , which can be equal to  $N_1$  or a user-defined value. Random Under sampling involves randomly selecting  $n$  instances from  $D_0$  without replacement.

The steps involved in Random Under sampling are as follows:

1. Random selection of instances: Randomly select  $n$  instances from  $D_0$  without replacement.

This can be represented as  $D'_0 = \{x_i | x_i \in D_0\}, i = 1, 2, \dots, n$

2. Create the balanced dataset: Combine the randomly selected instances from the majority class  $D'_0$  with all instances from the minority class  $D_1$  to form the balanced dataset  $D'$ . The balanced dataset  $D'$  consists of  $n$  instances from class 0 and  $N1$  instances from class 1.

## Vector embeddings

Vector embeddings, also known as word embeddings or word vectors, are a method used to represent words or entities as vectors in a high-dimensional continuous space. The main idea behind vector embeddings is to capture the semantic and syntactic relationships between words based on their contextual usage in a given corpus.

In this approach, each word is assigned a dense vector representation, where words with similar meanings or usage patterns are represented by vectors that are close to each other in the vector space. This enables machines to perform computations and make inferences based on the geometric relationships between these vectors.

The resulting word vectors can capture various linguistic properties, such as word analogies (e.g., "king" - "man" + "woman" = "queen") and semantic similarities (e.g., "cat" and "dog" are closer in the vector space compared to "cat" and "car"). These embeddings provide a rich representation of words that can be utilized in various natural language processing (NLP) tasks.

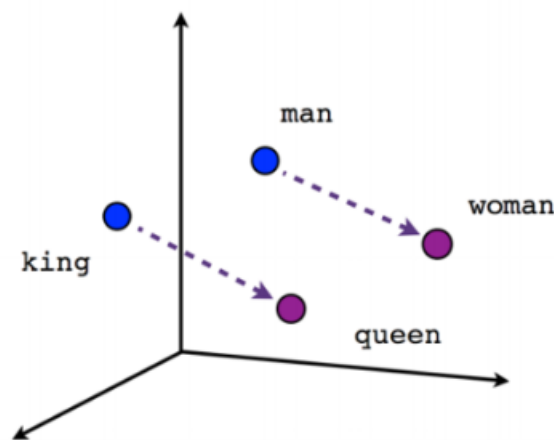


Figure 4.12: Example of the semantic distance between two analogies

## ”Word2Vec”

Word2Vec is a neural network-based algorithm that learns word embeddings by maximizing the likelihood of predicting context words given a target word or vice versa. It represents words as vectors in a continuous vector space. There exist two main architectures: Skip-gram and CBOW.

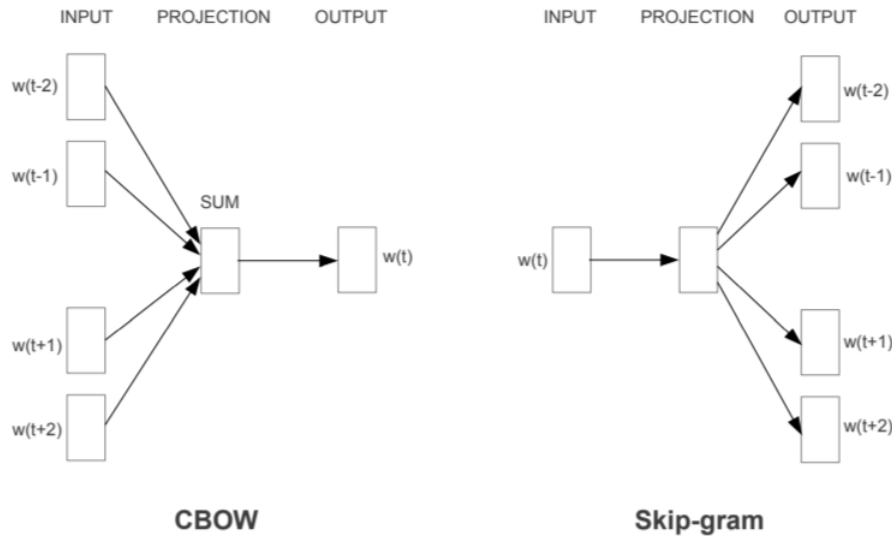


Figure 4.13: Architecture of skip vs CBOW

The Skip-gram model is a type of word embedding model that aims to predict the context words given a target word. In Skip-gram, we represent each word in a high-dimensional vector space, where the position of each word vector captures its semantic meaning.

The CBOW model, on the other hand, predicts the target word based on its surrounding context words. It aims to learn the word vectors that can best represent the context information.

Both Skip-gram and CBOW models utilize a neural network architecture, typically a shallow neural network with a single hidden layer. The word vectors are learned through the training process, where the model adjusts the word vectors to improve the prediction performance.

## Gradient Descent

When it comes to solving machine learning problems, optimization plays a crucial role in finding the best set of model parameters. Optimizers are algorithms or methods that guide the learning process by iteratively updating the model parameters to minimize a defined objective function or maximize a desired performance metric. One commonly used optimization algorithm is Gradient Descent.

The goal of Gradient Descent is to minimize the cost or loss function of the classification model by iteratively updating the model parameters. It achieves this by computing the gradient of the cost function with respect to each parameter and taking steps in the direction of the steepest descent. Given a cost function  $J(\theta)$ , where  $\theta$  is the vector of weights (parameters). The updated values at each step can be calculate using the following rule:

$$\theta = \theta - \alpha \cdot \frac{\partial J}{\partial \theta}$$

where  $\alpha$  is the learning rate, controlling the size of the parameter updates, and  $\frac{\partial J}{\partial \theta}$  denotes the gradient of the cost function with respect to  $\theta$ .

To compute the gradient, we can use the chain rule of calculus to obtain:

$$\frac{\partial J}{\partial \theta} = \frac{1}{m} \cdot \sum (h(x_i, \theta) - y_i) \cdot x_i$$

The weights are updated iteratively for a specified number of iterations or until convergence.

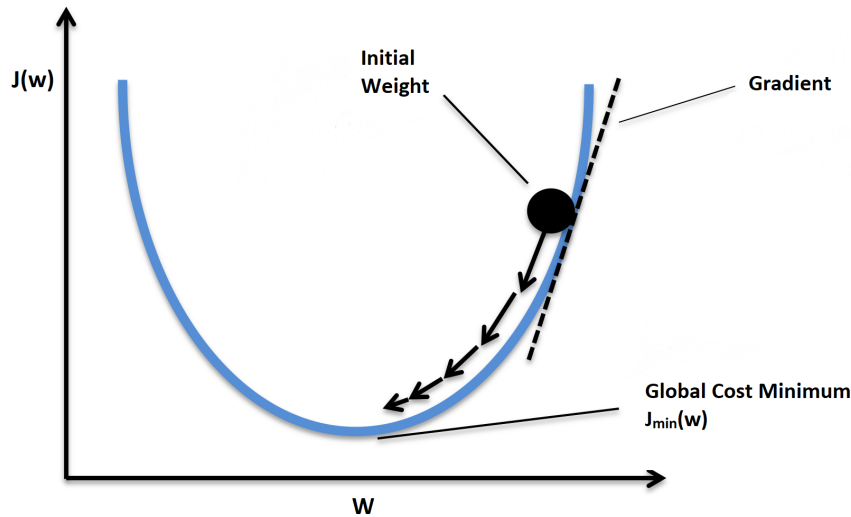


Figure 4.14: Gradient Descent illustrated in a two dimensional example

## GridSearch

Grid search is a hyperparameter tuning technique used in machine learning to find the best combination of hyperparameter values for a given model.

The grid search algorithm works by exhaustively searching through a predefined grid of hyperparameter values. For each combination of hyperparameters, the model is trained and evaluated using a chosen evaluation metric, such as accuracy or mean squared error. The performance of the model is then recorded.

Once the grid search is complete, the best combination of hyperparameters can be determined based on the evaluation metric. This set of hyperparameters can then be used to train the final model on the full training dataset and evaluate its performance on a separate test set to obtain an unbiased estimate of its generalization performance.



```
Hyperparameter_One = [a, b, c]  
Hyperparameter_Two = [x, y, z]
```

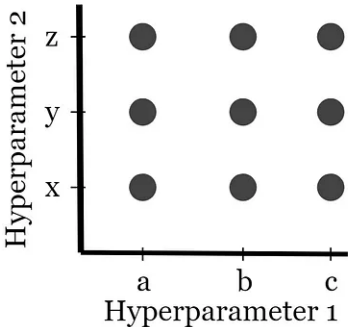
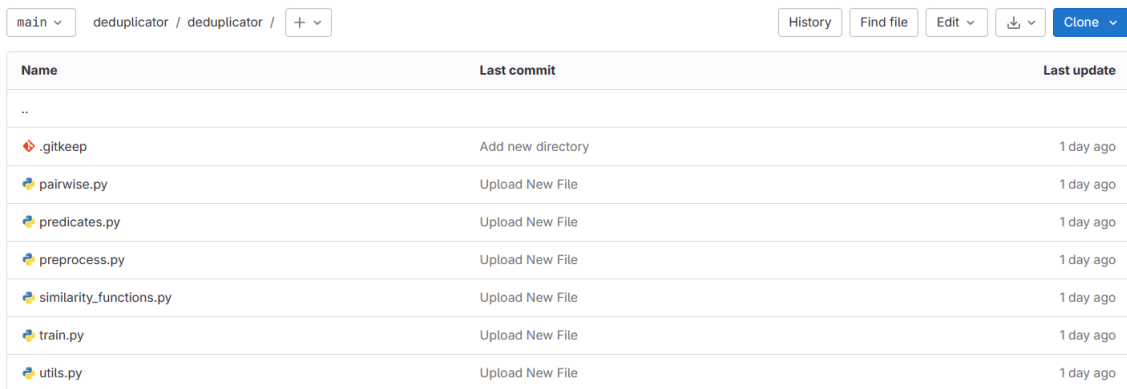


Figure 4.15: GridSearch parameter tuning

# Appendix B

In this appendix, we will explore the structure of the (.py) modules within the deduplicator repository. These modules are categorized based on their respective functions in the workflow process, allowing us to gain insights into how each module plays a distinct role in the overall functionality of the deduplication system.



The screenshot shows a GitHub repository interface for 'deduplicator'. At the top, there is a breadcrumb navigation: 'main > deduplicator / deduplicator / +'. To the right are buttons for 'History', 'Find file', 'Edit', a download icon, and 'Clone'. Below this is a table with three columns: 'Name', 'Last commit', and 'Last update'. The table lists several Python files, each with a commit message and a timestamp of '1 day ago'.

Name	Last commit	Last update
..		
.gitkeep	Add new directory	1 day ago
pairwise.py	Upload New File	1 day ago
predicates.py	Upload New File	1 day ago
preprocess.py	Upload New File	1 day ago
similarity_functions.py	Upload New File	1 day ago
train.py	Upload New File	1 day ago
utils.py	Upload New File	1 day ago

Figure 4.16: Deduplicator’s python modules

## 4.10 Preprocessing Dataframe (preprocess.py)

Within this file, we find a collection of functions dedicated to the various aspects of handling the Spark Dataframe. Many of these functions were already encountered the previous section. These functions are designed to facilitate the loading, processing, transformation, and reshaping of the Dataframe

These functions perform the operations mentioned in the Data preparation phase.

### 4.10.1 Reading the files

```
1 from pyspark.sql import SparkSession
2
3 def read_data(files: list, Session: SparkSession):
4     """
5     Reads the CSV files and combine them into a Pyspark Dataframe
6     @param files: A list of csv file paths to read
7     @param Session: a Pyspark Session object that has already been created.
```

```

8
9
10 @return: A new dataframe
11 """
12 df = Session.read \
13     .option("header", True) \
14     .option("ignoreLeadingWhiteSpace", True)\
15     .option("delimiter", ";") \
16     .option("quote", "'') \
17     .option("escape", "'') \
18     .csv(files)
19
20 # columns to keep
21 cols=['eid', 'title', 'creator', 'afid', 'affilname', 'affiliation_city', '
22 affiliation_country', 'author_count', 'author_names', 'author_afids', 'author_ids',
23 'coverDate', 'publicationName', 'description', 'authkeywords']
24
25 df= df.select(*cols)
26
27 return df

```

Code Listing 4.5: Function to read the .csv files

## 4.11 Blocking functions (predicate.py)

### 4.11.1 Perform prefix array blocking

```

1 from pyspark.sql import DataFrame
2 import pyspark.sql.functions as F
3
4 def prefix_array(df:DataFrame , column_name: str , l_min: int):
5     """
6     Performs prefix array blocking on a dataframe column (author_name)
7     @param df: The Pyspark dataframe to processes
8     @param column_name: The column to block based on (usually author_name)
9     @param l_min: the minimum length of prefixes to extract (eg: for l_min=
10     5, "Johnson" becomes ["Johns", "Johnso", "Johnson"])
11
12     @return: A dataframe
13     """
14     def generate_substrings(row, l_min = l_min, column_name=column_name):
15         author_name = row[column_name]
16
17         substrings = []
18
19         for j in range(l_min, len(author_name) + 1):
20             substrings.append(author_name[0:j])
21         return [row['row_index'], author_name, substrings]
22
23     df = df.select("row_index", column_name)

```

```

24
25 # Apply the map function to generate substrings
26 df = df.rdd.map(generate_substrings)
27 df = df.toDF(["row_index", column_name, "BKV"])
28
29 # Explode the BKV column
30 df = df.select("row_index", column_name, F.explode("BKV").alias("BKV"))
31
32 # Group by BKV and collect row_index values
33 df.createOrReplaceTempView("temp_df")
34 df = spark.sql("""
35     SELECT BKV, collect_list(row_index) AS indices
36     FROM temp_df
37     GROUP BY BKV
38 """)
39
40 #Create the size column
41 df = df.withColumn("size", F.size(F.col("indices")))
42
43 # Filter rows where "size" is greater than 1 (can't have a pair with only 1
44 value)
45 df = df.filter(df["size"] > 1)
46
47 # Sort the rows block filtering later
48 df = df.sort(df.size.asc(), df.BKV.asc())
49
50 return df

```

Code Listing 4.6: Function to perform prefix array blocking

#### 4.11.2 Perform sorted neighborhood blocking

```

1 from pyspark.sql import DataFrame
2 import pyspark.sql.functions as F
3 import jellyfish
4
5 def sorted_neighborhood(df:DataFrame, column_name: str, threshold: int):
6     """
7     Performs adaptive sorted neighbourhood blocking on a dataframe column
8     @param df: The Pyspark dataframe to process
9     @param column_name: The column to block based on (ex: author_name)
10    @param threshold: the threshold value when comparing records to block together
11
12    @return: A dataframe
13    """
14    df = df.select(F.col(column_name), F.col("row_index"))
15    df = df.repartition(1)
16    df = df.sortWithinPartitions(column_name, ascending=True)
17
18    def reformat(partitionData):

```

```

19     all_blocks = []
20     current_block = []
21     block_index = []
22     for row in partitionData:
23         if not current_block:
24             # Initialize the first row as the current block
25             current_block.append(row)
26             block_index.append(row["row_index"])
27         else:
28             # Calculate Jaro-Winkler similarity
29             similarity = jellyfish.jaro_similarity(current_block[0] \
30             [column_name], row[column_name])
31             if similarity >= threshold:
32                 current_block.append(row)
33                 block_index.append(row["row_index"])
34             else:
35                 all_blocks.append(block_index)
36                 current_block = []
37                 block_index = []
38                 current_block.append(row)
39                 block_index.append(row["row_index"])
40
41     for row in all_blocks:
42         yield [row]
43
44     df=df.rdd.mapPartitions(reformat).toDF(["indices"])
45
46     #Create the size column
47     df = df.withColumn("size", F.size(F.col("indices")))
48
49     # Filter rows where "size" is greater than 1 (can't have a pair with only 1
50     value)
51     df = df.filter(df["size"] > 1)
52
53     # Sort the rows block filtering later
54     df = df.sort(df.size.asc(), df.BKV.asc())
55     return df

```

Code Listing 4.7: Function to perform sorted neighborhood blocking

### 4.11.3 Perform block filtering

```

1 from pyspark.sql import SparkSession
2 from pyspark.accumulators import AccumulatorParam
3
4 #Define the counter class
5 class IndexCounterAccumulator(AccumulatorParam):
6     def zero(self, value):
7         return {}
8

```

```
9     def addInPlace(self, v1, v2):
10         for key, value in v2.items():
11             v1[key] = v1.get(key, 0) + value
12         return v1
13
14 def block_filtering(Session: SparkSession, pairs_df: DataFrame, threshold: int):
15     """
16     Performs the block filtering algorithm on the dataframe column
17     @param Session: a Pyspark Session object that has already been created.
18     @param pairs_df: The dataframe containing the pairs row indexes generated in
19     blocking phase.
20     @param threshold: the filtering threshold value to use for the algorithm
21
22     @return: A dataframe
23     """
24     def count_appearances(row):
25         global accum
26         indices = row.indices
27         for index in indices:
28             accum += {index: 1}
29         return (row.BKV, indices)
30
31     accum = Session.sparkContext.accumulator({}, IndexCounterAccumulator())
32
33     # Use map to apply the function to the DataFrame and transform it into an RDD
34     counted_df = pairs_df.rdd.map(count_appearances)
35
36     # Trigger an action that forces Spark to execute the transformations and
37     # accumulate the counts
38     counted_data = counted_df.count()
39
40     # Access the accumulator's value to see the counts
41     external_dict = accumulator.value
42     # print("External Dictionary:", external_dict)
43
44     for key, value in external_dict.items():
45         external_dict[key] = round(thresh*(value))
46
47     # Initialize a Python dictionary to store counts
48     count_dict = {}
49
50     def apply_filtering(data):
51         # Extract the values from the collected data
52         BKV, indices, size = data
53
54         global count_dict
55         updated_indices = []
56
57         for index in indices:
58             if index not in count_dict:
```

```

58         count_dict[index] = 1
59     else:
60         count_dict[index] += 1
61
62     # Check if the count for the index exceeds the threshold
63     if count_dict[index] <= external_dict.get(index, float('inf')):
64         updated_indices.append(index)
65
66     return {"BKV":BKV, "updated_indices": updated_indices}
67
68     # Collect the data to the driver
69     collected_data = result_df.collect()
70
71     # Apply the filtering function to each element in the collected data
72     result_list = [apply_filtering(data) for data in collected_data]
73
74     result = Session.createDataFrame(result_list)
75
76     # Filter rows where "size" is greater than or equal to 2
77     result = result.withColumn("new_size", F.size(F.col("updated_indices")))
78
79     # Filter rows where "size" is greater than or equal to 2
80     result = result.filter(result["new_size"]>= 2)
81
82     return result

```

Code Listing 4.8: Function to perform Block Filtering

## 4.12 Comparison Functions (similarity\_functions.py)

```

1 import numpy as np
2 from jellyfish import jaro_winkler_similarity
3
4
5 @udf(returnType=DoubleType())
6 def jaro_winkler_similarity_udf(str1, str2):
7     """
8     Defines a UDF for the jaro_winkler_similarity function
9     """
10    return distance.get_jaro_distance(str1, str2)
11
12
13 def jaccard_similarity(list1, list2):
14    """
15    Computes the jaccard_similarity between two lists
16    @param list1, list2: The lists to be compared
17
18    @return: A measure of similarity between 0-1
19    """
20    try:

```

```

21     set1 = set(list1)
22     set2 = set(list2)
23     intersection = set1.intersection(set2)
24     union = set1.union(set2)
25     similarity = len(intersection) / len(union) if len(union) > 0 else math.nan
26     return similarity
27 except ZeroDivisionError:
28     return np.nan
29
30 @udf(returnType=DoubleType())
31 def jaccard_similarity_udf(str1, str2):
32     """
33     Defines a UDF for the jaccard_similarity function
34     """
35     return jaccard_similarity(str1, str2)

```

Code Listing 4.9: Function to apply similarity comparison

### 4.13 Calculating pairwise similarities (pairwise.py)

```

1 from pyspark.sql import DataFrame
2 import pyspark.sql.functions as F
3
4 def pairwise_similarity(df: DataFrame, pairs_df: DataFrame, similarity_dict: dict):
5     """
6     Calculate the pairwise similarity between pairs generated in blocking phase
7     @param df: The Pyspark dataframe to process
8     @param pairs_df: The dataframe containing the pairs row indexes generated in
9     blocking phase.
10    @param similarity_dict: a dictionary containing key:value pairs or columns and
11    the similarity function to apply (ex: {"author_name": jaro_similarity, "co-
12    authors_names": jaccard_similarity})
13
14    @return: A dataframe
15    """
16    for column_name, similarity_func in similarity_dict.items():
17        # Join the pairs DataFrame with the second DataFrame based on smallest_value
18        # and largest_value
19        pairs_df = pairs_df.join(df.select("row_index", column_name).alias(
20            f'df1_{column_name}'), F.col('smallest_value') == F.col(f'df1_{
21            column_name}.row_index'), 'inner') \
22            .join(df.select("row_index", column_name).alias(f'df2_{column_name}
23            '), F.col('largest_value') == F.col(f'df2_{column_name}.row_index')
24            , 'inner')
25
26        # Calculate the similarity using the provided similarity function for the
27        # specified column
28
29        similarity_column = f'{column_name}_similarity'
30        pairs_df = pairs_df.withColumn(similarity_column, similarity_func(F

```



```

26     .col(f'df1_{column_name}.{column_name}'), F.col(f'df2_{column_name}
27     }.{column_name}'))))
28
29     output_columns = ['smallest_value', 'largest_value'] + [f'{col}
30     _similarity' for col in similarity_dict.keys()]
31     # Select the desired columns in the final output
32     output_df = pairs_df.select(*output_columns)
33
34     return output_df

```

Code Listing 4.10: Function to calculate pairwise similarities between candidate records

## 4.14 Utility functions (utils.py)

Utility functions do not necessarily have an impact on the performance of the models, but they are a convenient way to restructure code in a neat way, that allows for ease of read and reproducibility

```

1 import pyspark.sql.function as F
2
3 def generate_pairs(predicates, column: str):
4     """
5     Generate a dataframe of row indexes of pairs to be compared later.
6     @param predicates: The dataframe containing the predicates results (prefix array,
7     or q-grams, etc.)
8     @param column: column name containing the result of applying the predicate (
9     default: 'indices').
10
11     @return: A new dataframe
12     """
13     # Generate pairs of indices within each prefix
14     def generate_pairs_list(indices):
15         pairs = list(combinations(indices, 2))
16         # added this condition
17         pairs = [(smallest, largest) if smallest < largest else (largest, smallest)
18         for smallest, largest in pairs]
19
20         return pairs
21
22     generate_pairs_udf = F.udf(generate_pairs_list, ArrayType(ArrayType(LongType()))
23     )
24
25     predicates = predicates.withColumn("pairs", generate_pairs_udf(F.col(column))).
26     select(column, F.explode("pairs").alias("pair"))
27
28     # Extract the smallest and largest values from each pair
29     predicates = predicates.select(F.col("pair").getItem(0).alias("smallest_value"),
30     F.col("pair").getItem(1).alias("largest_value"))
31
32

```

```
27     return predicates
28
29
```

Code Listing 4.11: Function to generate the pairs after blocking