

# Scalable Entity Resolution



**Amrit Kaur (3055863)**

Thesis presented in the partial fulfilment  
of the requirement for the degree of  
Master of Science in Computer Science  
at the Rheinische Friedrich-Wilhelms-Universität Bonn

**Supervisor:** Dr. Hajira Jabeen

**First Examiner:** Prof. Dr. Jens Lehmann

**Second Examiner:** Dr. Kuldeep Singh

18 October, 2019

## Declaration of Authorship

I, Amrit Kaur, declare that this thesis, titled “Scalable Entity Resolution”, and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work. I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

3055863	
Student number	Signature
Amrit Kaur	24 May, 2019
Initials and surname	Date

## Acknowledgements

With profound gratitude and immense pleasure, I would like to thank my supervisor Dr. Hajira Jabeen for her utmost support, guidance, understanding and erudite supervision through-out my thesis. I am wholeheartedly grateful to Prof. Dr. Jens Lehmann for letting me pursue my master thesis at the Smart Data Analytics(SDA) group in the Rheinische Friedrich-Wilhelms-Universität Bonn (at the Department of Computer Science). I am highly obliged to Mr. Gezim Sejdiu for sharing his knowledge and experience in Spark and Scala and helping me resolve technical blockers during the evaluation of entity resolution code on the Spark clusters and integration with the SANSA project. Lastly, I would like to acknowledge my special vote of thanks to my parents for their endless love and support.

## Table of contents

<b>Declaration of Authorship</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of figures</b>	<b>viii</b>
<b>List of tables</b>	<b>ix</b>
<b>Abstract</b>	<b>x</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Challenges . . . . .	2
1.3 Outline . . . . .	3
<b>2 TECHNICAL DETAILS</b>	<b>4</b>
2.1 Semantic web . . . . .	4
2.1.1 Linked Data . . . . .	6
2.1.2 RDF data . . . . .	8
2.2 Entity Resolution in Knowledge Graphs . . . . .	9
2.3 The world of Big Data . . . . .	11
2.3.1 Apache Spark . . . . .	14
2.3.2 Hadoop Distributed File System(HDFS) . . . . .	18
2.4 Scala Language . . . . .	19
<b>3 RELATED WORK</b>	<b>20</b>
3.1 Block Building Techniques in Entity Resolution . . . . .	20
3.1.1 Schema-based Blocking Techniques . . . . .	21
3.1.2 Schema-agnostic Blocking Techniques . . . . .	24
3.2 The Role of Meta-Blocking . . . . .	27
3.3 Block Processing Techniques in Entity Resolution . . . . .	27
3.3.1 Controlling Block Sizes for effective comparison . . . . .	27

3.3.2	Effective Block Scheduling and Information Propagation . . . . .	27
3.3.3	Introduction of Block Filtering and Load Balancing Techniques . . . . .	28
3.3.4	Block Refinement and Purging . . . . .	28
3.3.5	Learning Based Approaches for Block Processing . . . . .	29
<b>4</b>	<b>APPROACH</b>	<b>30</b>
4.1	Problem Description . . . . .	30
4.2	Local Sensitivity Hashing . . . . .	30
4.2.1	HashingTF vs CountVectorizer . . . . .	31
4.2.2	MinHash for Jaccard Distance . . . . .	32
4.2.3	Approx Similarity Join . . . . .	33
4.3	Approaches for Entity Resolution . . . . .	33
4.3.1	Approach 1 - MinHash LSH method (considering all attributes) . . . . .	34
4.3.2	Approach 2 - MinHash LSH method (1 or 2 attribute) . . . . .	35
4.3.3	Approach 3 - MinHash LSH subjects and Jaccard similarity attributes . . . . .	36
4.4	Our Design . . . . .	42
4.4.1	Storage Unit . . . . .	43
4.4.2	Input and Output for Approach-2 MinHash LSH method (1 or 2 attribute) . . . . .	43
4.4.3	Storage and computation for Approach-3 MinHash LSH subjects and Jaccard similarity attributes . . . . .	43
<b>5</b>	<b>EVALUATION</b>	<b>46</b>
5.1	Cluster Configurations . . . . .	46
5.2	Datasets . . . . .	46
5.3	Configuration Parameters . . . . .	48
5.4	Evaluation Parameters . . . . .	49
5.4.1	Precision . . . . .	50
5.4.2	Recall . . . . .	50
5.4.3	F-measure . . . . .	50
5.5	Evaluation Results . . . . .	51
5.5.1	Discussion of results for Approach-2 . . . . .	51

5.5.2	Execution Times for Approach-2 datasets with HashingTF and CountVector- izer models . . . . .	53
5.5.3	Discussion of results for Approach-3 . . . . .	54
5.5.4	Discussing Example cases from Evaluation Results . . . . .	55
5.5.5	Execution Times for Approach-3 datasets with HashingTF and CountVector- izer models . . . . .	59
<b>6</b>	<b>CONCLUSION</b>	<b>61</b>
	<b>REFERENCES</b>	<b>64</b>

## List of Figures

2.1	Semantic Web Stack <sup>1</sup> . . . . .	5
2.2	Interlinking and Connectedness in Data . . . . .	6
2.3	Linked Open data as of October 2019 <sup>4</sup> . . . . .	7
2.4	Example RDF triple . . . . .	8
2.5	Entities in Dataset1 . . . . .	10
2.6	Entities in Dataset2 . . . . .	10
2.7	The fast Growing data <sup>15</sup> . . . . .	12
2.8	The four V's of Big data <sup>16</sup> . . . . .	13
2.9	The Apache Spark Stack <sup>22</sup> . . . . .	14
2.10	Overview of Spark Components <sup>27</sup> . . . . .	17
2.11	The HDFS architecture <sup>31</sup> . . . . .	18
3.1	Usage of Block Building . . . . .	20
3.2	Example dataset for Attribute Clustering Blocking . . . . .	25
3.3	Clusters formed by Attribute Clustering blocking . . . . .	25
3.4	URI semantics blocking . . . . .	26
4.1	Vectorising text to features code snippet . . . . .	32
4.2	MinHash for Jaccard Distance . . . . .	33
4.3	Approx Similarity Join . . . . .	33
4.4	Spark MinHash LSH method with all attributes . . . . .	34
4.5	Spark MinHash LSH method with 1 or 2 attribute . . . . .	36
4.6	Spark MinHash LSH subject . . . . .	37
4.7	Filtering common predicates code snippet . . . . .	38
4.8	Compare the attribute names for matched entities . . . . .	38
4.9	Compare the attribute values for intersecting attribute names code snippet . . . . .	39
4.10	Compare the attribute values for intersecting attribute names . . . . .	39
4.11	Input Dataframes . . . . .	40
4.12	Similar Entities by MinHash LSH subject . . . . .	40
4.13	Attribute names comparison for matched entities . . . . .	41

4.14 Matched entities with intersecting attribute names filtered with Jaccard similarity on attribute names . . . . .	41
4.15 Attribute values comparison for intersecting predicates . . . . .	42
4.16 Similar Entities . . . . .	42
4.17 Entity Resolution execution pipeline . . . . .	44
5.1 Confusion Matrix <sup>2</sup> . . . . .	49
5.2 Execution Time (in seconds) with Approach-2 . . . . .	53
5.3 Execution Time (in minutes) with Approach-3 . . . . .	59
6.1 SANSA stack <sup>1</sup> . . . . .	61



## List of Tables

2.1	List of commonly used RDD methods . . . . .	16
3.1	Example Person dataset for Blocking . . . . .	22
3.2	Standard Blocking based on zipcode attribute . . . . .	22
3.3	Sorted Neighbourhood Blocking based on zipcode attribute . . . . .	22
3.4	Q-grams based Blocking based on zipcode attribute . . . . .	23
3.5	Blocks formed by Token blocking approach . . . . .	24
3.6	Blocks formed by Typi Match blocking approach . . . . .	26
5.1	Evaluation dataset description . . . . .	47
5.2	Evaluation DBpedia dataset description . . . . .	48
5.3	Spark configuration parameters in cluster mode . . . . .	51
5.4	Evaluation results for DBLP-ACM dataset . . . . .	52
5.5	Evaluation results for DBLP-Scholar dataset . . . . .	52
5.6	Evaluation results for Abt-Buy dataset . . . . .	53
5.7	Evaluation Results for DBpedia datasets . . . . .	55
5.8	Evaluation Results - Similar Entities matched by minHash LSH subject . . . . .	55
5.9	Evaluation Results - Predicate knowledge for entity matches by MinHash LSH subjects	56
5.10	Evaluation Results - Intersecting predicates for similar entities matched with Jaccard Similarity predicates threshold(Jp) . . . . .	57
5.11	Evaluation Results - Values of Intersecting predicates for entity matches by Jaccard Similarity threshold for predicates (Jp) . . . . .	58
5.12	Evaluation Results - Similar entities found . . . . .	59

## Abstract

Entity resolution is the crucial task of recognizing and linking entities that point to the same real-world object in various information spaces. It finds its application in numerous tasks, particularly for public sectors like de-duplicating entities in federal datasets related to medicine, finance, transportation, business and law enforcement, etc. With the advent of growth in (semi-)structured data on the web in terms of volume and velocity, the task of linking records in heterogeneous data collections become more complicated. It is difficult to find inferences and semantic relations between entities across different datasets containing noisy data that encompasses typos, inconsistent and missing values appearing with loose schema bindings. As a pairwise comparison of entities over large datasets implies cartesian operation and is computationally expensive exhibiting quadratic complexity, the need to identify a more generic approach becomes more significant. The existing approaches reduce this complexity by aggregating similar entities into blocks and then processing them through the traditional Brute force approach. In our master thesis, we implement a more generic method for entity resolution in RDF data using the Spark framework and Scala as a programming language. Our approach does not apply any prior blocking but still eliminates the quadratic comparison. We compare our approaches with the existing state of the art methodologies and perform a thorough evaluation on five real-world datasets, we show that our suggested approaches are effective, efficient and scalable over large heterogeneous information spaces.

**Keywords:** Semantic web, Entity Resolution(ER), Local Sensitivity Hashing(LSH), MinHash Algorithm

# CHAPTER 1

## INTRODUCTION

### 1.1. MOTIVATION

In the recent past, we have experienced a huge proliferation in data on the web owing to the success of Social and Semantic Web applications. This ever-growing data originates from a variety of sources including web crawled data, information extractors, sensor data aggregated and various user-generated content, involving unprecedented schema heterogeneity and high levels of data redundancy. As per the current stats recorded, the DBpedia<sup>1</sup> 2016-10 release consists of 13 billion (2016-04: 11.5 billion) pieces of information (RDF triples) out of which 1.7 billion (2016-04: 1.6 billion) were extracted from the English edition of Wikipedia, 6.6 billion (2016-04: 6 billion) were extracted from other language editions and 4.8 billion (2016-04: 4 billion) from Wikipedia Commons and Wikidata. Also, adding the large NLP Interchange Format(NIF) datasets for each language edition increased the number of triples further by over 9 billion, bringing the overall count up to 23 billion triples. Another semantic knowledge base YAGO<sup>2</sup>, derived from Wikipedia, WordNet, WikiData, GeoNames, and other data sources currently contain more than 10 million entities (like organizations, persons, cities, etc.) with more than 120 million facts about these entities.

The problem is further accentuated as the data generated from heterogeneous sources suffers huge quality issues and is loaded with erroneous information containing false or missing values. Thus, identifying different entity profiles, usually with contrasting schemata that map to the same real-world instance becomes a prerequisite task for effective data integration and reuse in the world of Semantic Web. Inherently, the task of Entity Resolution is quadratic  $O(N^2)$  in nature; given two large entity collections E1 and E2, the task of processing and comparing each entity profile in E1 with each entity in E2 naively to link records would take a long time for large datasets. Several approaches have been introduced till date that aims to reduce these extensive comparisons in large data collections by typically employing a blocking stage. This entails placing entities that are similar in the same block and performing comparisons only within the block and not outside it. But, we still face redundant and superfluous entity profile comparisons while processing these

---

<sup>1</sup><https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10>

<sup>2</sup><https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago>

blocks. The worst-case complexity is  $O(m^2 * |B|)$ , where  $m$  represents the block(s) of maximal size(i.e. containing most entities to be compared within a block) and  $|B|$  being the total number of blocks.

Many researches existing in the field of ER aim at generating blocks efficiently, reducing the time and space complexity. Some particularly focus on controlling the sizes of blocks, while others on efficiently processing these blocks by pruning or purging the information in these blocks. Our work focuses on introducing a generic approach that deals with the heterogeneity of data and maps duplicate profiles in two entity collections, without priory inducing time and effort in initially blocking the data. We eliminate the block building stage and focus on directly processing the entity collections through a MinHash LSH based approach to identify different entity profiles described with different schemata in a structured or semi-structured format, but are the same in the real world. Our approach increases the performance and scalability of ER tasks over large datasets. Performing computations over GBs of data on a standard machine with limited memory and processing speed is challenging. So, we utilize a distributed framework Spark that is fast and open-source with Scala as our chosen programming language, making it beneficial for our research work.

## 1.2. CHALLENGES

With the enormous dynamically growing real-world data, the task of Entity Resolution is of prime importance for effective reuse of (semi-)structured data and leveraging data integration. It, however, posses the various challenges due to inherent characteristics of the available data:-

1. *Dealing with the heterogeneity of data is difficult*- As the data stems from multiple sources, there are no specific standard schemata defined. With the wide variety of attributes, loose schema bindings, and data being in various formats ranging from a JSON object to pure tag-style annotations forming different profiles for the same real-world object, comparing such profiles become an extremely challenging task.
2. *Velocity and volume of data*- The data is growing dynamically and in a huge amount. The need has generated to link entities across cross-domain distributed data sources effectively on a large scale. Thus, neither the naive quadratic approach of pairwise comparisons nor the blocking strategy is efficient for larger datasets.

3. *Inconsistent and noisy values in data*- Several typos and missing values exist in the extracted data. Also, there could be extraction errors or alternative descriptions present for the same entity at different sources. Such deficiency or false information increases difficulty in our task.
4. *Need for labeled data*- When supervised training based block processing strategies are involved in the Entity Resolution task, additional time and effort are induced in labeling training data for the model to learn.

In these settings, it is difficult to perform entity resolution tasks efficiently and effectively on GBs of data. Thus, we propose a model that focuses on eliminating these discussed challenges and simultaneously introducing scalability in ER tasks while reducing the traditional quadratic complexity.

### 1.3. OUTLINE

This thesis comprises of six chapters. Chapter two represents a general overview of concepts like Semantic web, RDF data, Spark, etc. Chapter 3 provides knowledge of the previous work done in the area of Entity Resolution. Chapter 4 focuses on various approaches/methodologies suggested by us to efficiently resolve the problem of Entity Resolution in Big data. Chapter 5 highlights the validity of our approach by extensive evaluation results compared and explained against the existing state of the art methods. Chapter 6 concludes the research work.

## CHAPTER 2

### TECHNICAL DETAILS

In this chapter, we provide a brief overview of the topics that are closely connected with our thesis work. It includes the knowledge of Semantic web, Linked Data, Resource Description Format(RDF) and how Entity Resolution is a crucial task in the world of Big data. We also discuss the Spark framework and Scala language that we have used for our thesis implementation and evaluation.

#### 2.1. SEMANTIC WEB

The word Semantic indicates "meaning". Thus, Semantic Web is the web of data with its associated meaning. During recent years, there has been a huge expansion in data over the web. The web penetrates society in several forms such as Social contacts through blogging or social networking platforms, E-commerce (like buying, selling, advertising of commodities), Education, Recreation and Work-life balance, etc. With this tremendous growth, the web accompanies a transition from industrial to an information society, providing infrastructure for a new quality of information handling with acquisition and provisioning. The current web is immensely successful, containing huge amounts of information that is machine-processable and human-readable but since the web data is heterogeneous in terms of content, structure, and character-encoding, it needs automated reasoning techniques for intelligent information integration. Humans can derive information from a given piece of data but the web can only deal with syntax. It needs the ability to integrate and fuse data from diverse sources to satisfy the human quest of information. The lack of comprehensive background knowledge to interpret information found on the web can be dealt with when the syntax is combined with its semantics for processing information intelligently. Thus, Semantic Web deals with the understanding of the information available on the internet.

The term "Semantic Web" was forged by Tim Berners-Lee for a web of data where pages are structured and tagged in a way that can be directly read by computers. As per<sup>1</sup>, the Semantic Web extends the World Wide Web(WWW) through the standards approved by World Wide Web Consortium (W3C). These standards promote common data formats and exchange protocols on the Web, the Resource Description Framework (RDF) being the most elementary. According to the

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Semantic\\_Web](https://en.wikipedia.org/wiki/Semantic_Web)

W3C, "The Semantic Web provides a common framework to share and reuse data across different enterprises, applications, and community boundaries". The Semantic Web is therefore regarded as an integrator across varied systems, content, and information applications. Figure 2.1 illustrates the Semantic Web architecture, called The Semantic Web Stack. Also, known as "Semantic Web Layer cake" or "Semantic Web Cake", where each layer has a specific function to make the web data machine-understandable. RDF<sup>2</sup> is a plain language for expressing the data model. It encodes structured information and exchanges information on the web. The RDF Schema(RDFS)[1] extends the RDF data model and is a vocabulary describing schema( i.e. properties and classes) of RDF-based resources semantically. OWL is an acronym for Web Ontology Language. An Ontology provides a reference frame for the disambiguation and the global interconnection of knowledge. It has its own explicit formal semantics example OWL 2.0 with Manchester syntax. It provides more vocabulary and extensive knowledge for describing classes, properties and relationships between them (e.g. disjointness, cardinality and, equality, etc.), SPARQL<sup>3</sup> is the general query language used for querying RDF databases.

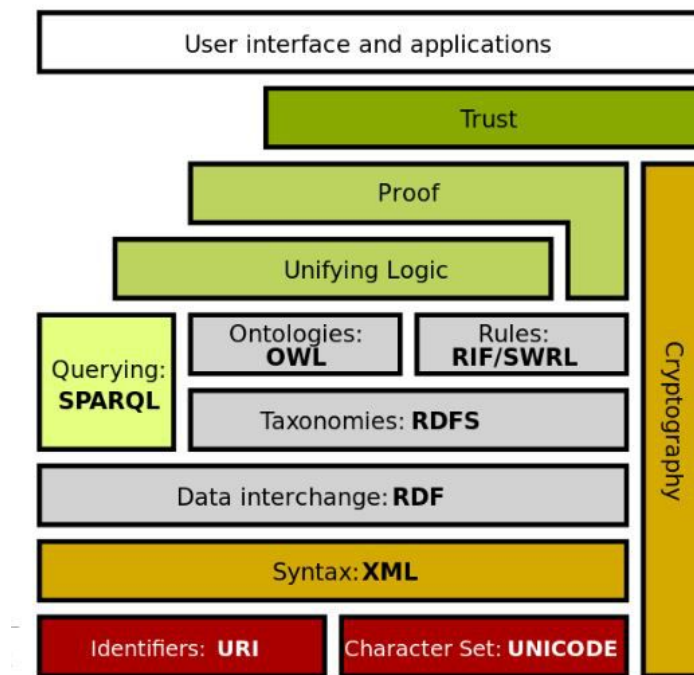


Figure 2.1: Semantic Web Stack<sup>1</sup>

<sup>2</sup><https://www.w3.org/TR/rdf-schema/>

<sup>3</sup><https://en.wikipedia.org/wiki/SPARQL>

### 2.1.1. Linked Data

Linked Data<sup>4</sup> refers to structured data that is bridged to other data to make it more useful. The Semantic web is all about linking data throughout the web and making it easily understandable to both humans and machines. Linked data is the best practice to achieve this goal. Approaching other datasets is viable through semantic queries. Since data is evolving to be more interlinked and connected. Hypertext<sup>5</sup> has links, blogs have pingback, tagging on social networking platforms groups all related data. Linked Data<sup>6</sup> introduces a set of design guidelines for sharing machine-readable interlinked data on the Web. Figure 2.2 shows the increasing connectedness in data. As the connectivity between data increases, the web becomes more powerful and fusion or integration of data becomes easier.

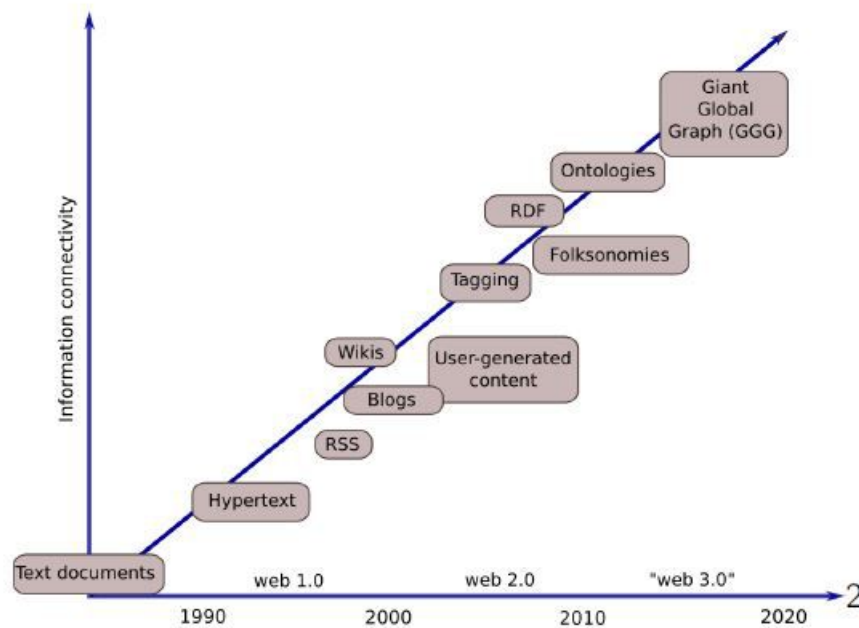


Figure 2.2: Interlinking and Connectedness in Data

Linked Open Data is a powerful amalgamation of Open Data and Linked Data, that is both linked and uses open sources, for example - DBpedia. Figure 2.3 shows the Linked Open Data as of

<sup>4</sup>[https://en.wikipedia.org/wiki/Linked\\_data](https://en.wikipedia.org/wiki/Linked_data)

<sup>5</sup><https://fr.slideshare.net/thobe/nosqleu-graph-databases-and-neo4j>

<sup>6</sup><https://ontotext.com/knowledgehub/fundamentals/linked-data-linked-open-data/>



October 2019. In early 2006, Tim Berners-Lee laid down four basic principles to achieve the vision of Semantic Web and link data from diverse sources. They are as follows:-

1. Use URIs as names for things.
2. Include links to other URIs so that they can discover more things.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Use HTTP URIs so that people can look up these names.

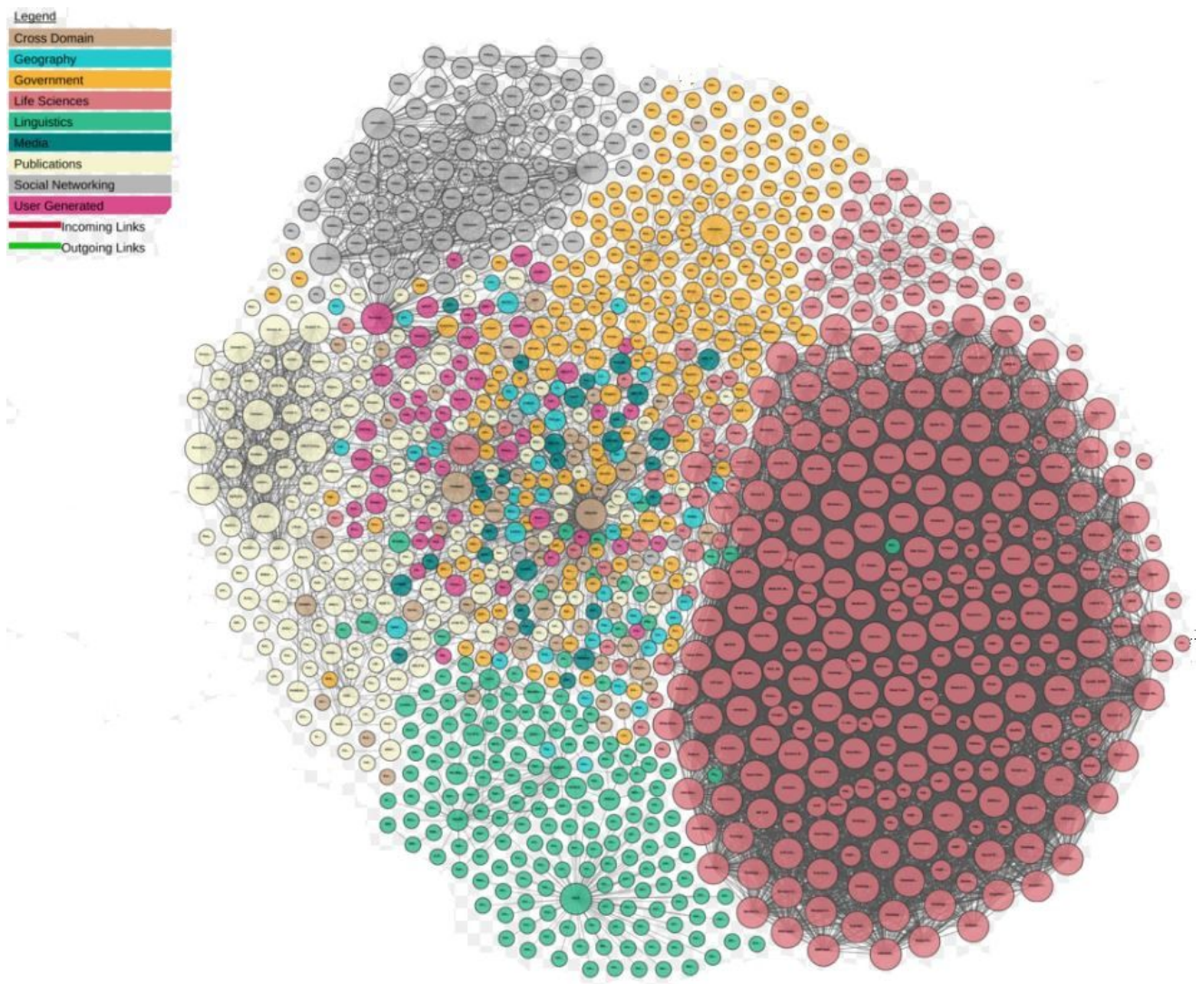


Figure 2.3: Linked Open data as of October 2019<sup>4</sup>

### 2.1.2. RDF data

RDF<sup>7</sup> stands for Resource Description Format. It originates from the family of World Wide Web Consortium(W3C) specifications. It is a common data model for modeling the conceptual description of the information available in web sources. Various RDF datasets available are open-source like DBpedia, YAGO, Wikidata, Freebase, etc. Originally, the design of the RDF model is inspired by the graphical representation of information, such that nodes or vertices denote the entities and several entities are connected by edges or lines, labeled with their relationships. Information is represented in RDF data in the form of triples. These triples show facts or statements in the form of a subject, predicate and object assigned as follows:-

1. A Subject is marked as a node containing URI or it is simply a blank node.
2. A Predicate or property should always be a URI.
3. An Object could be a URI, blank node or a literal.

Figure 2.4 represents an example triple(subject, predicate, and object).

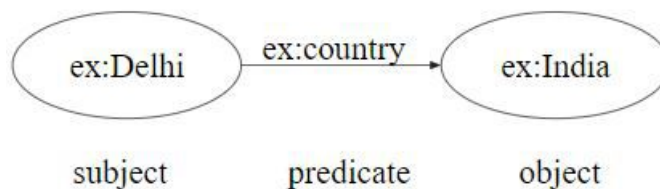


Figure 2.4: Example RDF triple

#### 2.1.2.1. Parts of an RDF graph

The RDF graph could be described in three parts:-

1. **URIs:-** URIs reference resources unambiguously i.e. give unique names globally. Example:- ISBN number for books
2. **Literals:-** Literals describe data values that do not have a clear identity. While describing a

---

<sup>7</sup>[https://en.wikipedia.org/wiki/Resource\\_Description\\_Framework](https://en.wikipedia.org/wiki/Resource_Description_Framework)

literal value, we can assign its datatype ex:- `"12-10-1993"^^xsd:date`, or even its representation language ex:- `"Tom Cruise"@en`. Literals are used only in Objects in an RDF triple.

3. **Blank Nodes:-** Blank nodes(or bnode)<sup>8</sup> represents an anonymous resource. This node in an RDF graph representing a resource for which a URI or literal is not given. It is only used as a subject or an object.

### 2.1.2.2. Serialization formats in RDF data

The data in the RDF data model is represented through a variety of syntax notations and serialization formats<sup>7</sup>. These are as described:-

1. **Turtle**<sup>9</sup>:- A text format with emphasis on human readability.
2. **N-Triples**<sup>10</sup>:- A text format with emphasis on simple parsing.
3. **RDF/XML**<sup>11</sup>:-The official XML-serialization of RDF.
4. **JSON-LD**<sup>12</sup>:-The W3C recommendation (originally in 2014) for expressing RDF data in JSON format.
5. **RDFa**<sup>13</sup>:- A mechanism for embedding RDFa in (X)HTML.

## 2.2. ENTITY RESOLUTION IN KNOWLEDGE GRAPHS

Entity Resolution<sup>14</sup> is the task of recognizing and de-duplicating entities across datasets i.e. Given two different datasets, the task of mapping entities that point to the same real-world data. The primary step in Entity Resolution is determining what an entity is. Thus, an entity is a unique object(a person, an organization, a location, a building or an animal) with a set of attributes(i.e. name, shape, size, habitat, etc.) describing its features. Each feature has a value associated with

---

<sup>8</sup>[https://en.wikipedia.org/wiki/Blank\\_node](https://en.wikipedia.org/wiki/Blank_node)

<sup>9</sup><https://www.w3.org/TR/turtle/>

<sup>10</sup><https://www.w3.org/TR/n-triples/>

<sup>11</sup><https://en.wikipedia.org/wiki/RDF/XML>

<sup>12</sup><https://en.wikipedia.org/wiki/JSON-LD>

<sup>13</sup><https://en.wikipedia.org/wiki/RDFa>

<sup>14</sup><https://www.districtdatalabs.com/basics-of-entity-resolution>

it. Since a single entity could have multiple entries across different datasets, we discuss entity resolution problem through such an interesting example.

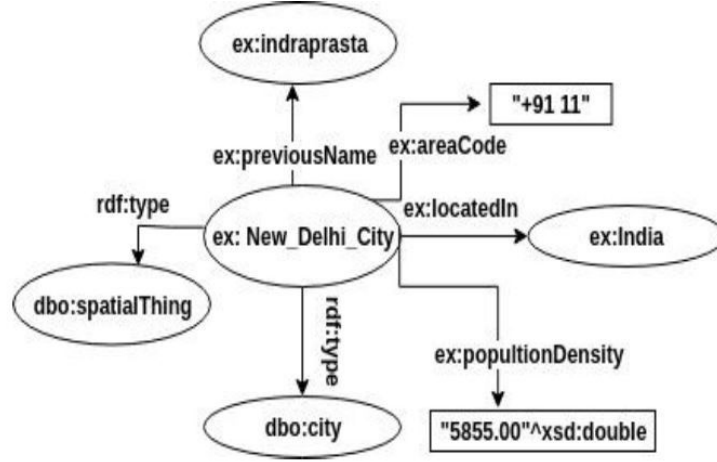


Figure 2.5: Entities in Dataset1

Figure 2.5 represents the city New Delhi in Dataset1 with its features (area code, previous name, country, type, and population density).

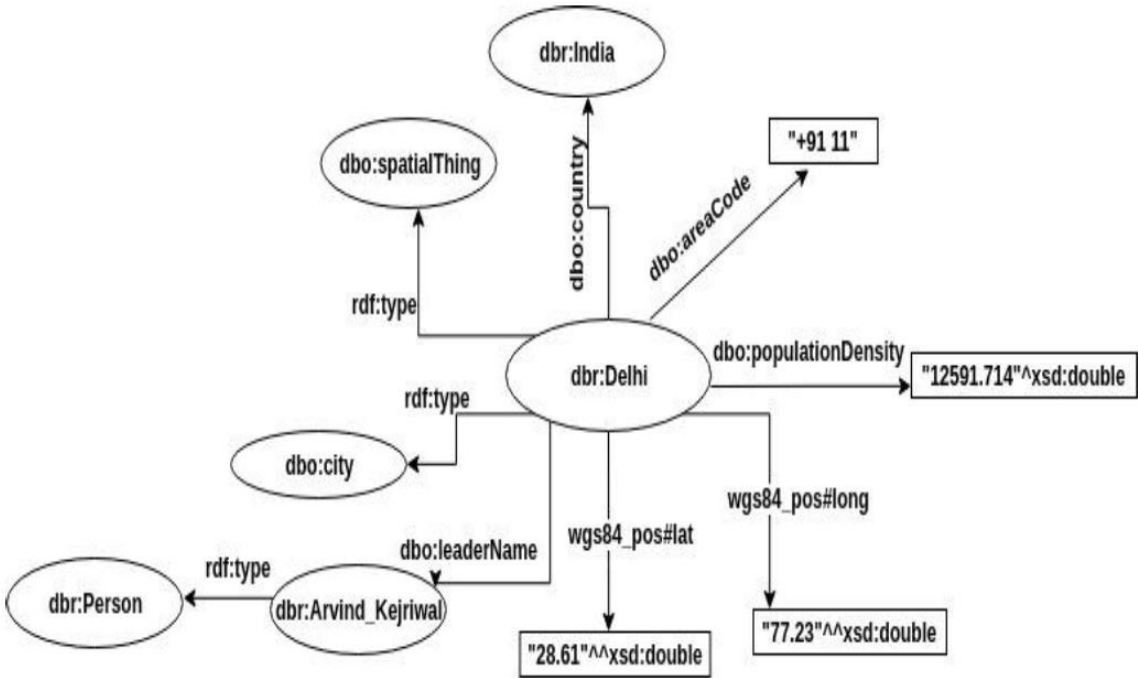


Figure 2.6: Entities in Dataset2

Figure 2.6 shows the exemplary representation of the city Delhi with its features (area code, latitude, longitude, country, type, population density, and its leaders) in DBpedia. These are two representations of the same city with varied features as they come from different sources. It can also happen that the same information is presented through different attribute names like `dbo:country` and `ex:locatedIn`. The problem is further accentuated by ambiguities in knowledge graphs. Several times the data contains typos or erroneous and missing values. Thus, the task of disambiguating entities among different datasets to achieve a single annotation with data firmly integrated becomes challenging.

The three primary tasks involved in entity resolution are as follows:-

1. **Deduplication:-** Removing redundant copies of data.
2. **Record linkage:-** Identifying records that refer to the same entity across different datasets.
3. **Canonicalization:-** Converting multiple representations of same data into a standard unambiguous form.

With the advent of interlinking entities on semantic web networks and massively growing data, the task entity resolution gets more complicated. The data originating from heterogeneous sources is noisy and has no standard representation defined. It suffers quality issues and has missing links. Our thesis work focuses on introducing a generic approach to deal with the task of disambiguating entities in RDF data with minimal time and cost. Our approach is designed in a way that is effective, efficient and scalable when it comes to big data.

## 2.3. THE WORLD OF BIG DATA

Since the onset of the digital age, the amount of information that we generate and transfer has increased dramatically in the past few years. Devices like laptops, mobile phones, PCs, cameras and automobiles are the prime source for flooding data. The idea of Smart Cities, Homes, and Kitchens, etc, simplify our life with various sensing devices incorporated in our environment, that are creating pools and oceans of information each day. Figure 2.7 illustrates the 50 fold increase in data within 10 years<sup>15</sup>. The data generated and aggregated from different sources has varied

---

<sup>15</sup><https://www.speicherguide.de/download/dokus/IDC-Digital-Universe-Studie-iView-11.12.pdf>

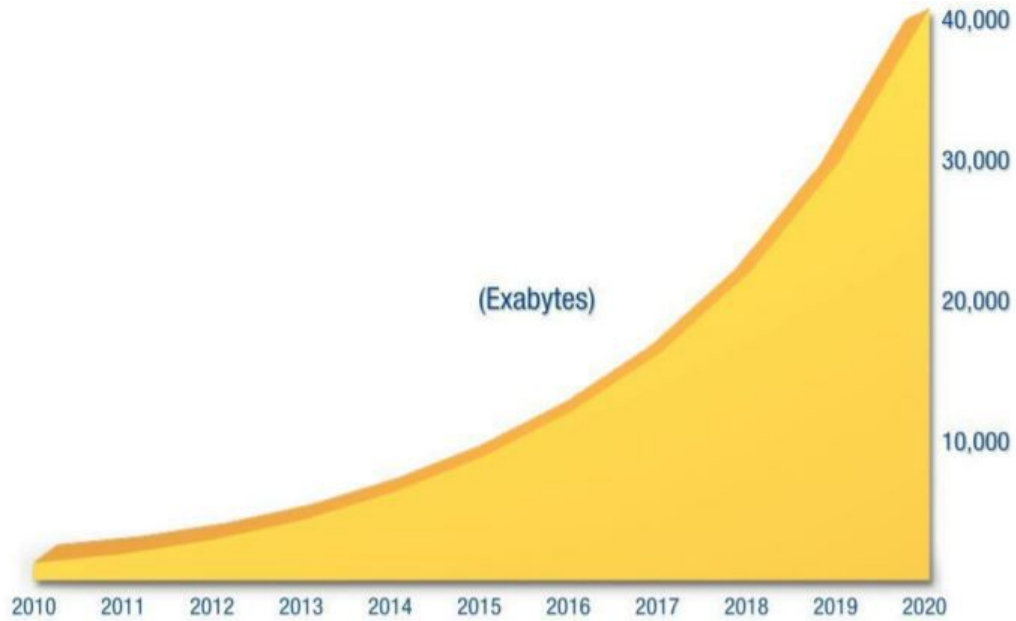


Figure 2.7: The fast Growing data<sup>15</sup>

attributes/fields and could be structured, unstructured or semi-structured. Merging this data effectively and extracting knowledge is useful but is a challenge for traditional machines with limited capacities to store, process and manage this voluminous data. So, with this increased connectivity between different devices and possible interlinking of information generated from heterogeneous sources, several agencies are developing new and powerful tools to analyze this data.

Big data refers to large and complex datasets that cannot be processed by traditional data-processing application software. Most organizations store and analyze this data to understand growing customer needs and predict market trends for their benefits. Working on these datasets effectively in acceptable time and cost is a challenge in itself. The various Big data technologies currently in the market are Hadoop, Cassandra, Pig, Hive, Spark, Storm, etc.

There are four main behaviors or concepts associated with the term "Big data" as shown in Figure 2.8<sup>16</sup>:-

1. **Volume:-** From software logs to fast-growing sensor event detection data recorded timely, the volume of data is increasing minute by minute. Currently, we have data in petabytes<sup>17</sup>.

<sup>16</sup><https://www.bluecoppertech.com/big-data-the-new-gold/>

<sup>17</sup><https://en.wikipedia.org/wiki/Petabyte>

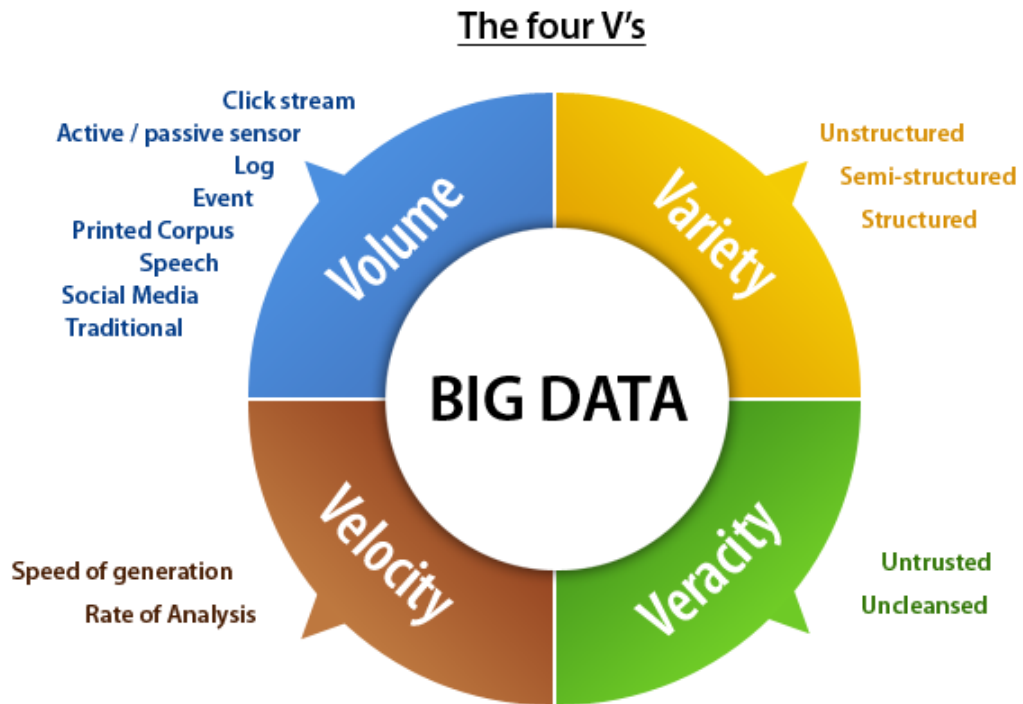


Figure 2.8: The four V's of Big data<sup>16</sup>

In future, it would be in Exabytes<sup>18</sup> or Yottabytes<sup>19</sup>. Figure 2.7 represents the growth in data by the year 2020.

2. **Variety:-** As the data comes from heterogeneous sources, it could range from texts, audio, video, tweets to pure-tag style annotations. Thus, it is difficult to determine its structure.
3. **Velocity:-** The high speed of information generation and constant transmission of generated data in real-time determines the velocity, as shown in Figure 2.7.
4. **Veracity:-** The data is noisy. It suffers quality issues like extraction errors, missing, erroneous values or other inconsistencies that could affect the accuracy of our results.

<sup>18</sup><https://en.wikipedia.org/wiki/Exabyte>

<sup>19</sup><https://en.wikipedia.org/wiki/Yottabyte>

### 2.3.1. Apache Spark

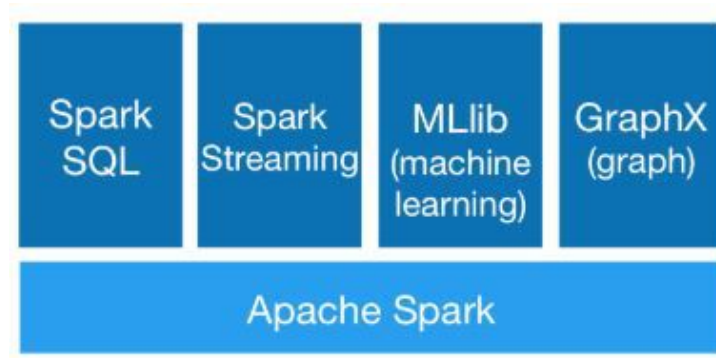


Figure 2.9: The Apache Spark Stack<sup>22</sup>

Apache Spark<sup>20</sup> is a fast, multi-paradigm, and general-purpose cluster computing system with its support available in multiple languages like Java, Scala, Python, and R. It<sup>21</sup> shines for big data as it is faster than other platforms like Hadoop in terms of computation and resource utilization with better memory capacities. Built on top of Scala, runs on a JVM, it could be used effectively for batch or real-time data processing. It comprises of a rich set of higher-level tools that are independent elements stacked under a common hood, making it a unified analytics engine for large data processing. Each element performs its specific task. Figure 2.9<sup>22</sup> shows the Apache Spark Stack. The description of which is as follows:-

1. **Spark Core:-** It forms the foundation of the overall project and provides support for basic I/O functionalities, distributed task dispatching, and scheduling exposed through an application programming interface based on the RDD abstraction.
2. **Spark SQL:-** It introduces the concept of data abstraction through Dataframes providing support for (semi-)structured data processing. It also provides support for SQL query language, with command-line interfaces and ODBC/JDBC server.
3. **MLlib:-** The distributed machine learning framework on top of Spark Core that makes statistical and predictive analysis easy for Spark users providing support for regression, clustering

<sup>20</sup><https://spark.apache.org/docs/latest/>

<sup>21</sup>[https://en.wikipedia.org/wiki/Apache\\_Spark](https://en.wikipedia.org/wiki/Apache_Spark)

<sup>22</sup><https://spark.apache.org/>



and dimensionality reduction techniques.

4. **GraphX and Graphframes:-** Distributed graph processing framework. The abstraction supported by GraphX is RDDs, while Graphframes supports Dataframes.
5. **Spark Streaming:-** It uses fast scheduling capabilities of Spark Core to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on them.

#### 2.3.1.1. Abstractions and Concepts of developing Apache Spark solutions

##### 1. Resilient Distributed Dataset(RDD):-

RDD<sup>23</sup> forms a collection of immutable elements that can be processed in parallel. They<sup>24</sup> are resilient(i.e. fault-tolerant) with the help of RDD lineage graph with the ability to recompute missing or damaged partitions caused by node failures. They<sup>25</sup> are the most fundamental data structure in Spark, where each dataset is divided into logical partitions, computed on different nodes of the cluster(i.e. distributed).

There are two ways to create RDDs:-

1. By parallelizing an existing collection in your driver program.
2. By referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, etc.

RDD supports two kinds of operations:-

1. Transformations:-When we load data into an RDD, we perform an activity on the loaded data like filter or map. This creates a new RDD. The transformations are lazily evaluated i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution.
2. Actions:- These are operations that trigger computation and returns value.

Table 2.1 shows a list of important RDD methods<sup>26</sup> with their explanations.

---

<sup>23</sup>[https://www.tutorialspoint.com/apache\\_spark/apache\\_spark\\_rdd.htm](https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm)

<sup>24</sup><https://spark.apache.org/docs/latest/rdd-programming-guide.html>

<sup>25</sup><https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd.html>

<sup>26</sup><http://snappydatainc.github.io/snappydata/apidocs/index.html#org.apache.spark.rdd.RDD>

Table 2.1: List of commonly used RDD methods

Method	Description
flatMap	Return a new RDD by first applying a function to all elements of the RDD, and then flattening the results.
map	Return a new RDD by applying a function to all elements of the RDD.
foreach	It runs a function on each element of the dataset.
filter	Return a new RDD containing only the elements, satisfying a given condition(s).
union	Return the union of two RDDs (combine RDDs).
intersection	Return the intersection of two RDDs (common RDDs).
persist	Persists the RDD with the default storage level (memory only).
count	Return the number of total elements.
unpersist	Mark the RDD as non-persistent, and removes it from memory and disk.
repartition	Shuffles the data across the network.
sparkContext	Used to create the RDD.
saveAsTextFile	Save the RDD as a text file.
reduceByKey	The reduceByKey method runs on an input data in (key, value) form and reduce it by key.

## 2. Directed Acyclic Graph(DAGs):-

When we run an application on Apache Spark, it constructs a graph for the sequence of computations, marking the nodes mapped to RDDs and providing the execution flow of algorithm and data in RDDs. Anything in SPARK is done through a process called lazy loading. When a DAG is created, Spark does not perform the computation and execution of underlying data but waits for an action to trigger it. Thus, only when output is required, the processing is done based on the flow determined by the DAG.

### 2.3.1.2. High-level overview of Spark components

The Figure 2.10 illustrates the major spark components<sup>27</sup>.

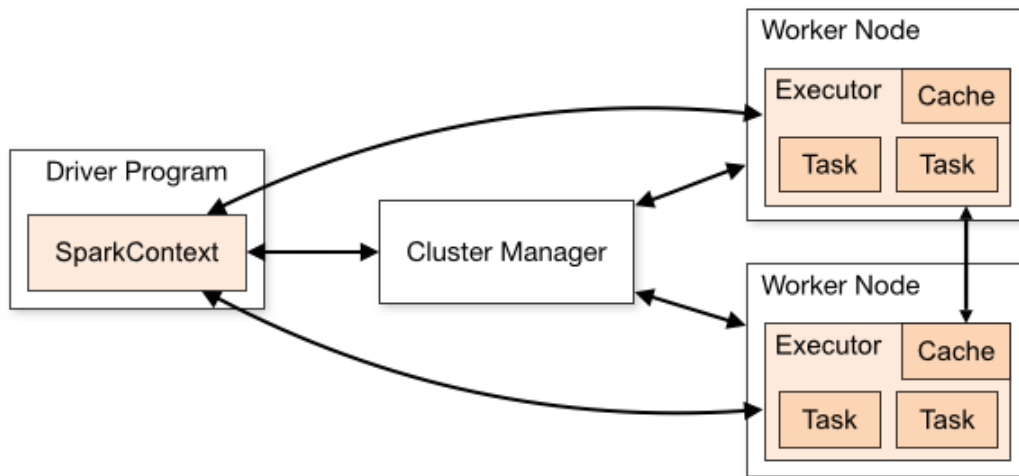


Figure 2.10: Overview of Spark Components<sup>27</sup>

1. **Driver Program:-** When we start Apache Spark in Spark-shell, the Spark-shell becomes the driver program or an entry point(ex:- main function in C), creating the spark context.
2. **Spark Context:-** The driver initiates the Spark Context. The Spark Context allows communication with other nodes in a cluster. These nodes are worker nodes.
3. **Worker nodes:-** When we run an application, it creates a different process within a worker node. So, every spark application runs in its own executor. A worker node will have many executors. Within an executor, we have DAGs (i.e. constructed graph broken down to stages and tasks). These individual tasks run within executor.
4. **Cluster manager:-** They are required for performing parallel operations on various nodes. Spark provides support for different cluster managers like Hadoop yarn, Apache Mesos, etc.

<sup>27</sup><https://spark.apache.org/docs/latest/cluster-overview.html>

### 2.3.2. Hadoop Distributed File System(HDFS)

The HDFS<sup>28</sup> is a distributed file system designed to run on commodity hardware, by employing a Namenode and a Datanode. It is used as the primary storage system by Hadoop<sup>29</sup> applications. It<sup>30</sup> is highly resilient and can be easily deployed on low-cost hardware. It is suitable for handling large data sets and provides high throughput access to application data. It is written in Java and is supported on all major platforms like Windows, Linux, and Unix-based operating systems. Figure 2.11 shows the HDFS architecture<sup>31</sup>.

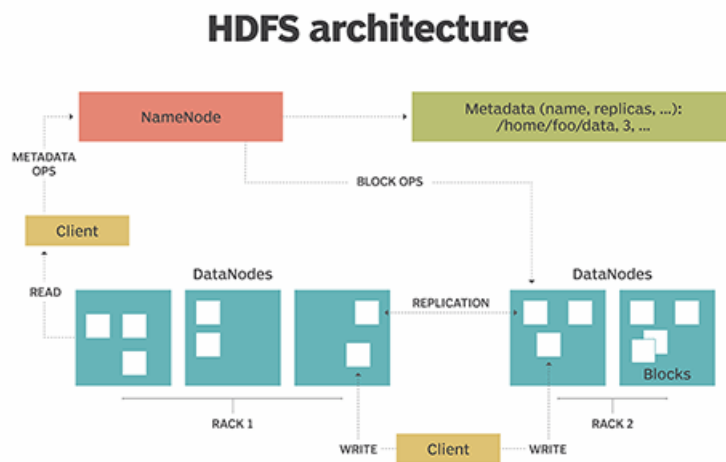


Figure 2.11: The HDFS architecture<sup>31</sup>

#### 2.3.2.1. HDFS Goals

**1. Hardware Failure:-** HDFS works in a distributed file system with a cluster of nodes where the failure of any node at any point of time is a prominent exception. The detection of such faults and fast automated recovery of data on the nodes is a crucial task of HDFS.

**2. Streaming Data Access:-** HDFS is designed more for batch processing rather than interactive real-time data processing. More emphasis is laid on high throughput of data access rather than

<sup>28</sup><http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html>

<sup>29</sup>[https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)

<sup>30</sup><https://searchdatamanagement.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS>

<sup>31</sup><https://searchdatamanagement.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS>

low latency.

**3. Large Data Sets:-** HDFS is used for storing gigabytes to terabytes of data. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

## 2.4. SCALA LANGUAGE

Scala<sup>32</sup> is a general-purpose, a multi-paradigm programming language designed by Martin Odersky, that accounts for scalability. First introduced on 20 January 2004, its current stable release is 2.13.1 as of 11 June 2019. It is implemented in Scala and licensed under Apache License 2.0. The filenames are saved with .scala, .sc extensions. The main features of Scala are as follows:-

1. It is purely object-oriented, providing support for the functional programming approach in combination.
2. Highly influenced by Java, Scala programs can convert to bytecodes and can run on the Java Virtual Machine(JVM) in support.
3. It also provides language interoperability with Java.
4. Besides this, Scala helps a developer write concise codes.
5. It automatically detects the type-inference for a variable, we do not need to mention it explicitly. Just declare var or val and Scala is smart enough to understand data type on its own.

---

<sup>32</sup>[https://en.wikipedia.org/wiki/Scala\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Scala_(programming_language))

## CHAPTER 3

### RELATED WORK

This section provides a comprehensive overview of the previous work done in the field of Entity Resolution. Inherently the task of entity resolution is quadratic. To reduce the number of pairwise comparisons and retaining maximum true matches, the existing state of the art approaches imply a two-step process of grouping similar entities into the same block and non-similar entities in a different block, namely the Block building stage and latter comparing only the entities within a block i.e. block processing. Section 3.1 deals with the Block building techniques applied in ER and Section 3.2 provides an overview of techniques employed to process those blocks.

#### 3.1. BLOCK BUILDING TECHNIQUES IN ENTITY RESOLUTION

Consider datasets D1 and D2, each with 10000 entries, the absolute pairwise comparison, considering all entities would cause 100 million comparisons. If each comparison takes exactly 1 sec. The total comparisons would take 100 million seconds, which is equivalent to 1157.5 days i.e. 3.17 years. Typically to reduce this extensive computation, the existing methodologies suggest applying suitable blocking techniques as shown in Figure 3.1.

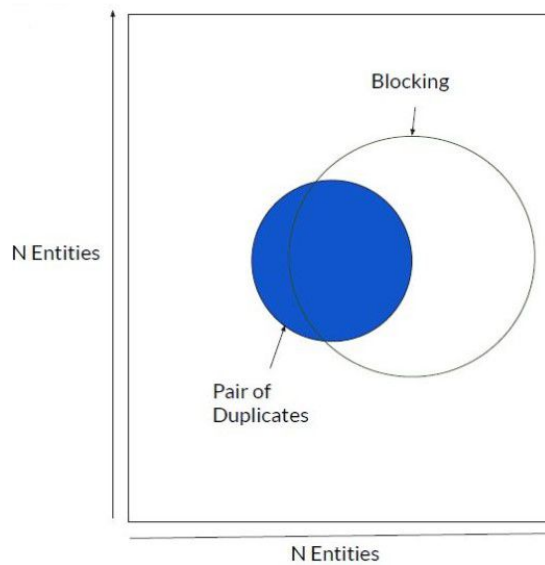


Figure 3.1: Usage of Block Building

The various techniques employed for block building in the ER, either use the available schemata knowledge or do not rely on it. Thus, they are categorized as schema-based or schema-agnostic block building techniques. During the blocking stage, the blocks formed could be disjoint such that an entity appearing in one block does not appear in any other block or non-disjoint depending on the datasets. Usually, with semantic data, the blocks formed contain overlapping entities. This leads to additional comparisons during block processing.

### 3.1.1. Schema-based Blocking Techniques

As the name suggests, available schemata knowledge is used to build the blocks. This is generally applied when we are trying to identify similar entities in homogeneous data sources like relational data (for ex:- Identifying similar persons in two databases based on their first\_name, surname, birth\_date, profession and address fields). Given two entity collections E1 and E2, internally each input entity into these two collections is transformed into a compact representation comprising one or more blocking keys. These blocking keys are nothing but selected attribute information. With the a-priori known schema, various techniques are employed to form blocks based on attribute name and value information of an entity. These are as described below:-

1. **Standard Blocking:-** The most prominently applied blocking technique based on schema knowledge is the Standard Blocking technique. Consider two relational entity collections E1 and E2 with n1 and n2 entity profiles respectively. Since the structure of these entity collections is homogeneous i.e. each entity profile has the same attribute names, we select the most appropriate attribute name based on noise levels and distinctiveness such that the blocking is done based on the similarity of attribute values for a particular attribute name. Example:- For a Person based dataset (with attributes id, first\_name, last\_name, address, and zipcode), all entity profiles from collection E1 and E2 having the same zipcode go into the same block. Table 3.1 represents the same example database. Table 3.2 represents the corresponding blocks formed based on zipcode attribute. Based on this technique the blocking is performed in the [2] publication. This technique is particularly effective for disjoint blocks, but the effectiveness of this approach diminishes when it encounters noisy or missing values in the data.

Table 3.1: Example Person dataset for Blocking

id	first_name	last_name	address	zipcode
r1	Elena P.	Grey	Los Angeles, California	54321
r2	Rhea	Joseph	Boston, USA	35421
r3	Elena	Grey	L.A., California, USA	54321
r4	Richah	Sveta	Greece	42867
r5	Rachel	Desouza	California, USA	35421

Table 3.2: Standard Blocking based on zipcode attribute

54321	35421	42867
r1	r2	r4
r3	r5	

**2. Sorted Neighbourhood:-** This technique is largely used because of its robust nature. An appropriate blocking key and window size(ws) are selected. The entities are sorted based on the selected blocking key in alphabetical order and a window of fixed size ws is slid over the sorted list. At each iteration, the entities co-occurring in the window are compared. Consider the Person database in Table 3.1 with zipcode as the selected attribute. Let window size(ws) be 2. The sorted list formed is as shown in Table 3.3. Thus, it would undergo four iterations comparing  $\{(r2, r5), (r5, r4), (r4, r1), (r1, r3)\}$ . This technique is typically employed in [3] publications.

Table 3.3: Sorted Neighbourhood Blocking based on zipcode attribute

Sorted List	Entity Profile
35421	r2
35421	r5
42867	r4
54321	r1
54321	r3



**3. Canopy Clustering:-** Another blocking scheme used in [3], when little knowledge of schema is available. This typically employs configuring parameters for string matching and threshold  $t_1$  and  $t_2$ . This methodology typically employs redundancy for robustness at the cost of low efficiency but deals with noisy data effectively. The method aims at creating overlapping clusters, called canopies. Select a random record  $r^*$  from the dataset  $M$  as the current cluster center and choose threshold  $t_1$  and  $t_2$  (with  $t_1$  greater than  $t_2$ ) such that the distance of all other records from  $r^*$  are determined. The records with distance from  $r^*$  less than  $t_2$  are members of the cluster but cannot be treated as centers. The records that are outside  $t_2$ , but inside  $t_1$  are also clustered members and can be considered for cluster centers also. The records outside  $t_1$  are not members of the cluster. Repeat the selection of another random record from  $M$ , if  $M$  is not empty. The technique is typically employed in [3], [4] publications. This method is mostly used when we aim to process blocks with supervised training of a model.

**4. Q-grams based blocking:-** In this technique, the appropriate blocking key is selected and sub-string q-grams of length  $q$  are created for the values of that blocking key (BKVs). For each of the created BKVs, the matching entity is placed into its block based on the similarity of its values. Consider the Person dataset in Table 3.1 Let Zipcode be the selected blocking key and let  $q=2$  for q-grams. The BKVs created are  $\{54, 43, 32, 21, 35, 42, 28, 86, 67\}$ . The blocks are formed are as shown in Table 3.4. It is robust to noisy attribute values, even for the BKVS but can cause a higher computational cost for large and overlapping blocks. The technique is considered in [4] publication.

Table 3.4: Q-grams based Blocking based on zipcode attribute

54	43	32	21	35	42	28	86	67
r1	r1	r1	r1	r2	r2	r4	r4	r4
r2	r3	r2	r1	r5	r4			
r3			r3		r5			
r5			r5					

### 3.1.2. Schema-agnostic Blocking Techniques

With tremendously exploding data on the internet from a wide variety of sources, the schema knowledge is not readily available or properly defined. The data aggregated varies from web crawled data to pure-tag style annotations. Thus to deal with this attribute heterogeneity, noisy or missing values and, loose schema bindings, there was a need to circumvent schema-agnostic blocking methods. Attribute-agnostic blocking methods completely ignore all attribute names, but utilizes all attribute values. Several approaches developed in this attribute-agnostic blocking methodology are discussed below:-

**1. Token Blocking:-** As the name suggests, token blocking is based on the idea of tokenizing attribute values. Given an entity profile, extract all the tokens from its attribute values. Create a set of tokens from all the entity profiles in the two entity collections E1 and E2. For each token, create a block placing all the entities containing that token into the block. Each block should contain at least two entities. Consider the Person dataset in Table 3.1. The resulting blocks obtained from token blocking are as shown in Table 3.5. The blocks show redundancy, containing overlapping profiles. The technique is typically employed in [4; 5; 6] publications.

Table 3.5: Blocks formed by Token blocking approach

Blocks	Entity Profile
Elena	r1, r3
Grey	r1, r3
Los Angeles	r1, r3
USA	r3, r2, r5
California	r1, r3, r5
54321	r1, r3
35421	r2, r5

**2. Attribute Clustering Blocking:-** As the name suggests, the idea is to group the similar attribute names into clusters and apply token blocking inside those clusters. The focus is to improve efficiency by reducing the size of blocks. For calculating similarity in attribute names to form the clusters, Jaccard similarity or TFDf measures could be applied. Consider the entity profiles in

Figure 3.2, on applying Attribute clustering blocking, the blocks formed are as discussed in Figure 3.3. The technique is typically employed in [4] publication.



Figure 3.2: Example dataset for Attribute Clustering Blocking

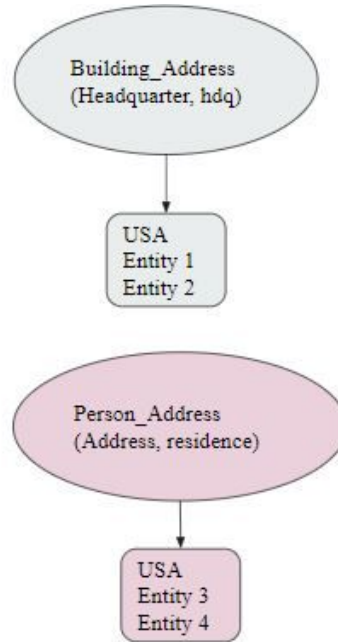


Figure 3.3: Clusters formed by Attribute Clustering blocking

**3. TYPi Match:-** In this technique, clustering is done on the basis of domain or type. So, we cluster entities into overlapping types and then apply token blocking on the attribute values. Consider the entity profiles in Figure 3.2, on applying Typi Match technique, the two clusters

formed are Organisation(Entity 1, Entity 2) and Person(Entity 3, Entity 4) respectively. The blocks formed by tokenising the attributes in the clusters are discussed in Table 3.6. The technique is also referred in [4] publication.

Table 3.6: Blocks formed by Typi Match blocking approach

Blocks	Entity Profile
WHO	Entity 1, Entity 2
USA_1	Entity 1, Entity 2
Elena	Entity 3, Entity 4
Grey	Entity 3, Entity 4
USA_2	Entity 3, Entity 4

**4. URI Semantics Blocking:-** As the name suggests, blocking is done based on the similarity of URI semantic. Used in [7] publication, the Example is shown in Figure 3.4 [7] with the dataset and the blocks formed. It can be done in the following three ways:-

1. Infix Blocking - all entities whose URI share a specific Infix are in a block.
2. Infix Profile Blocking - Every block corresponds to a specific Infix (of an attribute value) and contains all entities having it in their Infix Profile.
3. Literal Profile Blocking - Every block corresponds to a specific token and contains all entities having it in their Literal Profile.

<p>URL: <a href="http://dbpedia.org/resource/Barack_Obama">http://dbpedia.org/resource/Barack_Obama</a></p> <p>birthname: "Barack Hussein Obama II"</p> <p>dateOfBirth: "1961-08-04"</p> <p>birthPlace: "Hawaii" <a href="http://dbpedia.org/resource/Hawaii">http://dbpedia.org/resource/Hawaii</a></p> <p>shortDescription: "44th President of the United States of America"</p> <p>spouse: <a href="http://dbpedia.org/resource/Michelle_Obama">http://dbpedia.org/resource/Michelle_Obama</a></p> <p>Vicepresident: <a href="http://dbpedia.org/resource/Joe_Biden">http://dbpedia.org/resource/Joe_Biden</a></p>	<table> <tr> <td><b>Infix</b></td><td><b>Infix Profile</b></td></tr> <tr> <td>Barack_Obama</td><td>Michelle_Obama</td></tr> <tr> <td></td><td>Joe_Biden Hawaii</td></tr> <tr> <td><b>Literal Profile</b></td><td></td></tr> <tr> <td>Barack 08</td><td>America States</td></tr> <tr> <td>01 Obama 04 20 44th</td><td></td></tr> <tr> <td>2009 of Hussein Hawaii United</td><td></td></tr> <tr> <td>1961 the II President</td><td></td></tr> </table>	<b>Infix</b>	<b>Infix Profile</b>	Barack_Obama	Michelle_Obama		Joe_Biden Hawaii	<b>Literal Profile</b>		Barack 08	America States	01 Obama 04 20 44th		2009 of Hussein Hawaii United		1961 the II President	
<b>Infix</b>	<b>Infix Profile</b>																
Barack_Obama	Michelle_Obama																
	Joe_Biden Hawaii																
<b>Literal Profile</b>																	
Barack 08	America States																
01 Obama 04 20 44th																	
2009 of Hussein Hawaii United																	
1961 the II President																	

Figure 3.4: URI semantics blocking

### 3.2. THE ROLE OF META-BLOCKING

Some authors suggest applying a Meta-Blocking stage after block building to improve the effectiveness of the block processing stage. The Meta-Blocking technique aims at reducing the repeated comparisons in entities caused by overlapping blocks. Thus, meta-blocking reforms our block collection as a new collection which is equally effective but contains fewer comparisons i.e. diminishing the number of duplicated and superfluous comparisons in the block collection. It highlights the outline that the more blocks two entities share, the more similar they are. Thus, the high likelihood of them to be a matching record. It typically undergoes a process of building a graph from the block collection, weighing the edges of the graph by an appropriate similarity measure, then pruning the formed graph to reform our block collection. This technique is applied in [5; 8; 9] publication.

### 3.3. BLOCK PROCESSING TECHNIQUES IN ENTITY RESOLUTION

The blocks formed by the above techniques to reformed by the meta-blocking technique are now to be processed, such that the maximum true matches are retained in less number of comparisons. Various techniques are employed in previous work to accomplish this task efficiently.

#### 3.3.1. Controlling Block Sizes for effective comparison

The [10] focuses on controlling the sizes of block collections using clustering approaches. It focuses on the fact that effective block construction done based on the attribute name similarity or by limiting the size of blocks would lead to a decreased complexity while processing the blocks.

#### 3.3.2. Effective Block Scheduling and Information Propagation

The [11; 6] suggests a robust way to deal with the high redundancy challenges persisting in overlapping blocks. Since the same entity profile appears in multiple blocks, it inevitably leads to an increased number of comparisons required for de-duplication. So, block to be processed can be scheduled blocks through block utility based on the probabilistic analysis. Once the blocks are scheduled, purging is performed that removes the entity blocks that were built by a too common

attribute value i.e token. This cleans the oversized blocks and improves efficiency. The set of blocks is then compared and a hash table is maintained to keep track of the entity matches and this information is propagated further since identified matches are not reconsidered in the subsequently processed blocks. Lastly, block purging is involved to remove the blocks with few utility values and preempt the resolution if the cost of further comparison becomes too expensive. The comparison of exact entity matches is performed using the generic R-Swoosh algorithm suggested in [12].

### 3.3.3. Introduction of Block Filtering and Load Balancing Techniques

The [5; 8] employ various block filtering techniques during the initial phase of block building and meta-blocking as preprocessing strategies to achieve effectiveness. This includes various techniques to prune graphs constructed in a meta-blocking phase like *Weighted Edge Pruning (WEP)*, where the threshold is determined to keep average weight across all the edges, *Cardinality Edge Pruning (CEP)*, which sets the determined threshold by cardinality instead of weight over the edges. Other pruning methodologies are applied to nodes instead of edges. This includes *Weighted Node Pruning (WNP)*, where the threshold is set for each node as the average weight of the adjacent edges and *Cardinality Node Pruning (CNP)* that sets the determined threshold based on the cardinality of the nodes instead of weight. On applying these techniques, effective load balancing is done to assure maximum potential matches enhancing efficiency and scalability.

### 3.3.4. Block Refinement and Purging

The maximum effective true matches are retrieved while eliminating repeated comparisons, discarding superfluous comparisons and avoiding non-matching comparisons simultaneously in collected blocks. To achieve this, the [7] suggests composite blocking techniques, while [4] introduces a four-step approach, enhancing the block building stage with partitioning over categories and redundancy bearing methods. Once the blocks are built, both the publications suggest the initiation of block purging i.e. removing the blocks with fewer utility values and propagating the information about detected matches to further blocks that are yet to be processed. The [4; 6] suggests scheduling of blocks by the merge-purge approach and latter pruning the blocks for avoiding non-match comparisons. The approaches discussed till now are evaluated in terms of *Pair completeness (PC)* and

*Reduction Ratio (RR)*, which is equivalent to *Recall* and *Precision* respectively.

### 3.3.5. Learning Based Approaches for Block Processing

The effectiveness of Machine-Learning approaches depends on the provision of suitable, sufficient, and balanced training data. The [3] suggests combining different learning-based approaches (i.e. Decision trees, Logistic Regression, SVM and a multiple learning approach combining all three) and finding suitable match configurations like similarity thresholds. It highlights the need for suitable learning data (training data) and the effort of labeling it for supervised learning. It primarily has three main operator types (blocking, matching and training selection). It needs mappings as input (mappings contain entity a, entity b and their similarity values), to be compared against a similarity threshold(ts). If their similarity values exceed the specified ts, the entity pair matches, otherwise not. The training data is selected either randomly or by a ratio of matching or non-matching entities (suitable ratio – 0.2 to 0.5) and (suitable similarity threshold between 0.3 and 0.6). The ratio approach is generally considered better than the random approach. In this way, it follows the iterative two-way phase of the first generating a model with suitable labeled training data and latter applying the model. FEVER framework is used to evaluate these approaches.

## CHAPTER 4

### APPROACH

#### 4.1. PROBLEM DESCRIPTION

The largely booming heterogeneous data on the web has duplicate entities that need to be identified for data integration and de-duplication. Given two different knowledge bases, the task of recognizing entities that point to the same real-world data with the typical brute-force approach comparing all entities implies  $O(N^2)$  complexity. The existing above discussed strategies are also time-consuming because they undergo the two-step process of block building and then processing those blocks through learning or non-learning based approach. These approaches become volume intensive and hardware resilient when it comes to processing very large databases. With the advent of our next approaches, we focus on removing the block building stage and the effort incurred in labeling training data for learning-based approaches in the task of entity resolution. In this section, we suggest Local Sensitivity Hashing based approaches for entity resolution, that will not need multiple iterations of training phase (as required for the learning-based approaches) and can handle large heterogeneous knowledge bases.

#### 4.2. LOCAL SENSITIVITY HASHING

Local Sensitivity Hashing (LSH)<sup>1</sup> [13] is a modern technique that finds its applications in Near-duplicate detection, Genome-wide association study and Audio/video fingerprinting for large databases. It refers to a family of functions (known as LSH families) to hash data points into buckets. It works on the intuition that data points close to each other are hashed into the same bucket with high probability, while the data points located far apart are placed into different buckets. This helps in disambiguating manifestations of the same real-world entity in different records or mentions. An LSH [14] family is formally defined as follows:-

In a metric space  $(M, d)$ , where  $M$  is a set and  $d$  is a distance function on  $M$ , an LSH family is a

---

<sup>1</sup><https://spark.apache.org/docs/2.2.0/ml-features.html#locality-sensitive-hashing>



family of functions  $h$  that satisfy the following properties:-

$$\forall p, q \in M, \quad (4.1)$$

$$d(p, q) \leq r1 \Rightarrow Pr(h(p) = h(q)) \geq p1 \quad (4.2)$$

$$d(p, q) \geq r2 \Rightarrow Pr(h(p) = h(q)) \leq p2 \quad (4.3)$$

This LSH family is called  $(r1, r2, p1, p2)$ -sensitive.

In Spark, different LSH families are implemented in separate classes (e.g., MinHash), and APIs for feature transformation, approximate similarity join, and approximate nearest neighbor are provided in each class. In LSH, we define a false positive as a pair of distant input features (with  $d(p,q) \geq r2$ ) which are hashed into the same bucket, and we define a false negative as a pair of nearby features (with  $d(p,q) \leq r1$ ) which are hashed into different buckets. The spark LSH algorithms include Bucketed Random projection with Euclidean distance and MinHash for Jaccard Distance. In our next discussed approaches we are working with **MinHash LSH for Jaccard Distance** algorithm.

#### 4.2.1. HashingTF vs CountVectorizer

Both HashingTF and CountVectorizer<sup>2</sup> are methods to vectorise the tokenised text (i.e. generate term frequency vectors). HashingTF takes sets of terms or tokenized text as input and converts them into fixed-length feature vectors. It utilizes the hashing trick, using MurmurHash 3 typically as its hashing function and transforms a raw feature by mapping it into an index (term). Then term frequencies are calculated based on the mapped indices. Hashing usually suffers from potential collisions when different terms are hashed into the same buckets. This can be reduced by increasing the number of buckets in our hash tables, which in our case is the target feature dimension.

Count Vectorizer is a similar transformer, but unlike HashingTF that parses data once and then forms feature vectors directly, CountVectorizer undergoes scanning of data twice, initially for building the CountVectorizer model and then transforming tokenized text to feature vectors. This incurs the cost of additional storage space for unique features and also more time consuming, but results in a reversible process. Since, we can convert the features vectors to the text again with CountVec-

---

<sup>2</sup><https://spark.apache.org/docs/2.2.0/ml-features.html#feature-extractors>

torizer model, which is not possible with the typical HashingTf model. Figure 4.1 demonstrates HashingTf and Count Vectorizer via a code snippet.

```
def applyHashingTf_sub(inp_col: String, out_col: String, numFeatures: Int, data1: DataFrame, data2: DataFrame): (DataFrame, DataFrame) = {
  val vectorizer = new HashingTF().setInputCol(inp_col).setOutputCol(out_col).setNumFeatures(numFeatures)
  /*
   * For CountVectorizer model
   * val vectorizer = new CountVectorizer().setInputCol(inp_col).setOutputCol(out_col).setVocabSize(numFeatures).setMinDF(1).fit(data1)
   */
  val hashfeatured_entities_Df1 = vectorizer.transform(data1)
  val hashfeatured_entities_Df2 = vectorizer.transform(data2)
  return (hashfeatured_entities_Df1, hashfeatured_entities_Df2)
}
```

Figure 4.1: Vectorising text to features code snippet

#### 4.2.2. MinHash for Jaccard Distance

MinHash is an LSH family for Jaccard distance where input features  $\in N$  (i.e. sets of natural numbers). Jaccard distance of two sets is defined by the cardinality of their intersection and union. Given two sets A and B respectively, their Jaccard distance is identified as:-

$$d(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \quad (4.4)$$

MinHash<sup>3</sup> applies a random hash function  $g$  to each element in the set and takes the minimum of all computed hashed values:

$$h(A) = \min_{a \in A} (g(a)) \quad (4.5)$$

The input sets for MinHash are represented as binary vectors, where the vector indices represent the elements themselves and the non-zero values in the vector represent the presence of that element in the set. While both dense and sparse vectors are supported, typically sparse vectors are recommended for efficiency. Since empty sets cannot be transformed by MinHash implies any input vector must have at least 1 non-zero entry. The spark MinHashLSH function takes `setNumHashTables` as a parameter determining the number of hash tables to be used on input features for computing the hash values. This parameter value is adjusted by the user as per the requirement. A higher value leads to better accuracy when performing approx similarity join among the two sets and a lower value leads to time-saving solutions. This is a race of efficiency vs effectiveness of the solution. We have optimally set this value to 3 in our solutions. Figure 4.2 demonstrates the code snippet for

<sup>3</sup><https://spark.apache.org/docs/2.2.0/ml-features.html#minhash-for-jaccard-distance>

MinHashLSH function of Spark.

```
def minHashLSH(featured_entities_Df1: DataFrame, featured_entities_Df2: DataFrame): (MinHashLSHModel, DataFrame, DataFrame) = {
  val mh = new MinHashLSH().setNumHashTables(3).setInputCol("features").setOutputCol("hashed_values")
  val model = mh.fit(featured_entities_Df1)
  val transformed_entities_Df1 = model.transform(featured_entities_Df1)
  val transformed_entities_Df2 = model.transform(featured_entities_Df2)
  return (model, transformed_entities_Df1, transformed_entities_Df2)
}
```

Figure 4.2: MinHash for Jaccard Distance

#### 4.2.3. Approx Similarity Join

Approx Similarity Join<sup>4</sup> takes two datasets as input, computes distances among the entities in two datasets and returns pairs of rows in datasets whose distance is smaller than the user-defined threshold( $t$ ). It supports self-join as well as the join between two different datasets. The returned output also contains a distance column indicating the distance between each of the paired entities returned as a row. Figure 4.3 demonstrates this API via code snippet. Another feature approx nearest neighbor is available as an LSH operation in spark, but we are not using it currently in our approach.

```
def approxsimilarityjoin(model: MinHashLSHModel, df1: DataFrame, df2: DataFrame, threshold: Double): DataFrame = {
  val dataset = model.approxSimilarityJoin(df1, df2, threshold)
  val refined_ds = dataset.select(col("datasetA.entities").alias("entity1"), col("datasetB.entities").alias("entity2"))
  return refined_ds
}
```

Figure 4.3: Approx Similarity Join

### 4.3. APPROACHES FOR ENTITY RESOLUTION

As discussed above, our approaches focus on reducing the efforts and time consumption induced in block building stage (by the above-discussed approaches) for entity comparison i.e. we compare the entities without categorizing them into blocks and without multiple iterations of training our model as involved in various learning-based approaches. Also, our approach eliminates the need for labeled data to train our model.

---

<sup>4</sup><https://spark.apache.org/docs/2.2.0/ml-features.html#approximate-similarity-join>

#### 4.3.1. Approach 1 - MinHash LSH method (considering all attributes)

In this approach, we utilize all the information available with us about the entity mentions in the two databases, without any filtrations.

*Definition:- Let the entity collections in two databases be  $E1$  and  $E2$ , then an entity mention  $e1$  in a certain collection be recorded as a tuple  $\langle s, Ap \rangle$ , where  $Ap$  is a set of attributes, namely containing subject, attribute names and attribute values for the specific entity mention  $e1$  and  $s$  be the subject for the profile. The Entity collections  $E1$  and  $E2$  contains several entity mentions such that there are no duplicates within an entity collection i.e no two entity mentions within  $E1$  would be pointing to the same real-world object.*

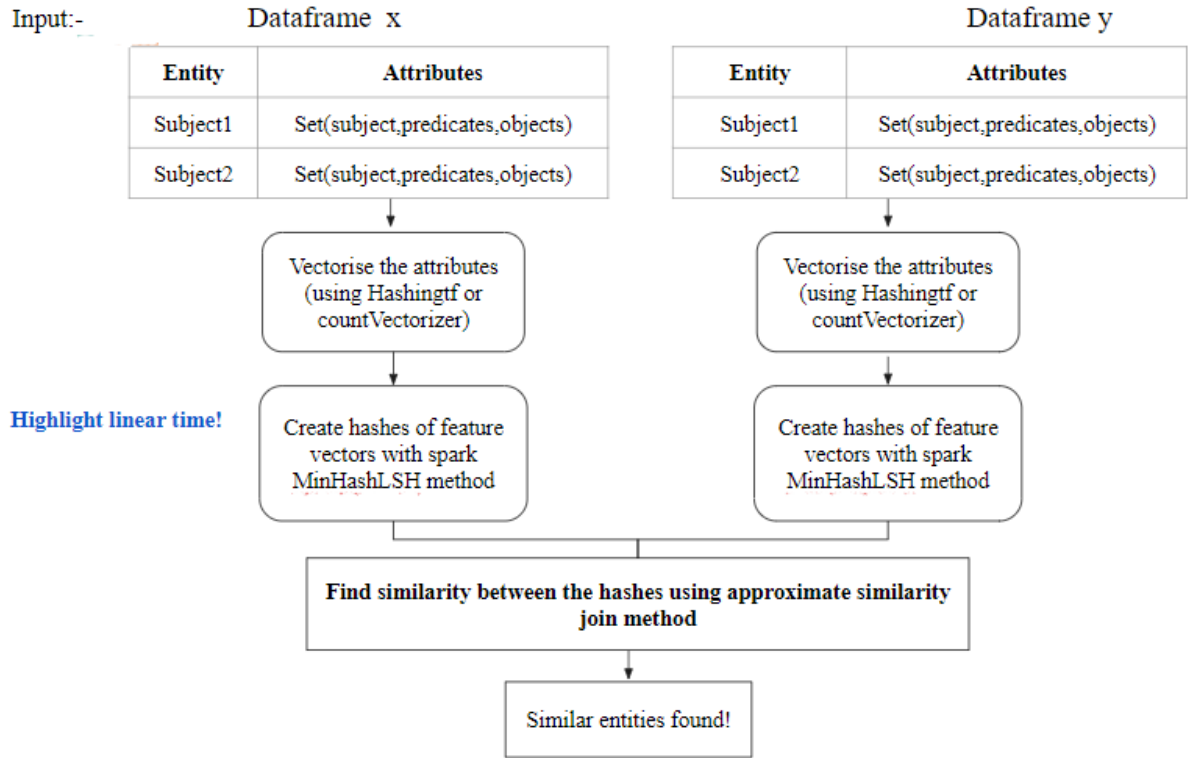


Figure 4.4: Spark MinHash LSH method with all attributes

For each of the entity collections, for every single entity mention, we vectorize its attributes using an HashingTF or a CountVectorizer technique and then create hashes of the respective feature vectors using spark MinHashLSH model, with 3 different random hash functions i.e. keeping the setNumHashTables attribute to 3. Find the similarity between the respective transformed hashes

for each of the entity mentions of the two entity collections with a spark `approxSimilarityJoin` method on an appropriate `threshold(t)`. This gives us the entities that are marked similar amongst the two entity collections `E1` and `E2` respectively. The two entities are marked similar only if the distance between them is less than the `threshold(t)`. This approach is depicted in Figure 4.4.

#### 4.3.1.1. Challenges involved in Approach 1

The knowledge graphs are typically incomplete. There are a lot of inconsistencies between the two different Entity collections to be compared. This missing information and noisy attribute information becomes a hinge. It results in huge distance between the two entity mentions, beyond the threshold being used for comparison, when we look for the similarities between the entity mentions in the two entity collections respectively. Thus, the two entity mentions are marked non-similar that are otherwise similar in the real world.

#### 4.3.2. Approach 2 - MinHash LSH method (1 or 2 attribute)

In this approach, we focus on using 1 or 2 main attributes for the entity mentions, instead of utilizing all the available knowledge. The inspiration for selecting the 1 or 2 attributes is attained from [3].

*Definition:- Let the entity collections in two databases be  $E1$  and  $E2$ , then an entity mention  $e1$  in a certain collection be recorded as a tuple  $\langle s, Ap \rangle$ , where  $Ap$  is a set of attributes, namely containing subject, only the selected 1 or 2 attribute names and its respective attribute values for the specific entity mention  $e1$  and  $s$  be the subject for the profile. The Entity collections  $E1$  and  $E2$  contains several entity mentions such that there are no duplicates within an entity collection.*

For each of the entity collections, for every single entity mention, we vectorize its attributes using an `HashingTF` or a `CountVectorizer` technique and then create hashes of the respective feature vectors using spark `MinHashLSH` model, keeping the `setNumHashTables` attribute to 3. Find the similarity between the respective transformed hashes for each of the entity mentions of the two entity collections with a spark `approxSimilarityJoin` method on an appropriate `threshold(t)`. The similar entities marked among the two entity collections are computed. This approach is depicted in Figure 4.5.

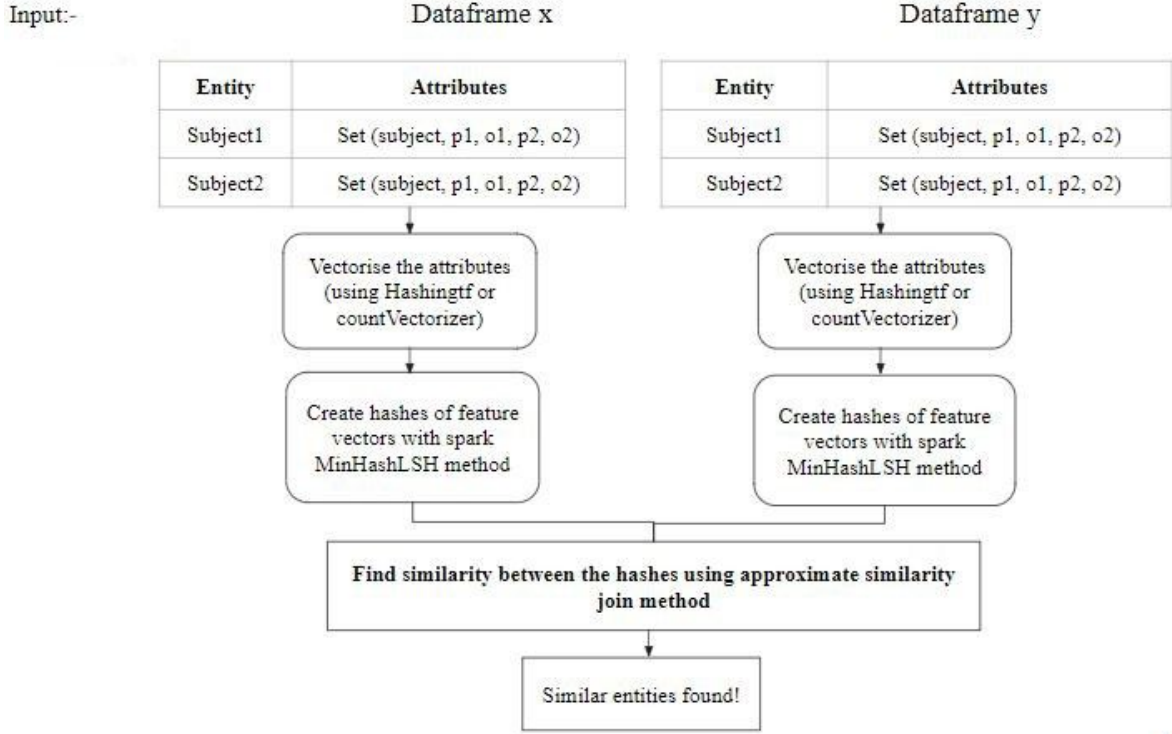


Figure 4.5: Spark MinHash LSH method with 1 or 2 attribute

#### 4.3.2.1. Aspects of Approach - 2

In this approach, selecting specific attribute information(1 or 2 attributes) amongst the heterogeneous, loosely schema bound information available with us from two sources forms an important aspect for comparison. Although this approach uses only selected attribute information about the entity mentions, it marks most of the similar entities correctly. The only hinge being less utilization of the available knowledge in the knowledge graphs. Thus, the maximum utilization of available knowledge from two sources becomes the inspiration for our next approach.

#### 4.3.3. Approach 3 - MinHash LSH subjects and Jaccard similarity attributes

For a broader comparison of entity mentions in two knowledge graphs, keeping an account of information from various attributes, we determine a 3-step process for entity resolution. The idea behind this approach is selecting common attributes of the two entity mentions based on the similarity of their subjects as depicted in Figure 4.6.

#### 4.3.3.1. Step 1 - Find matching entities based on LSH subjects

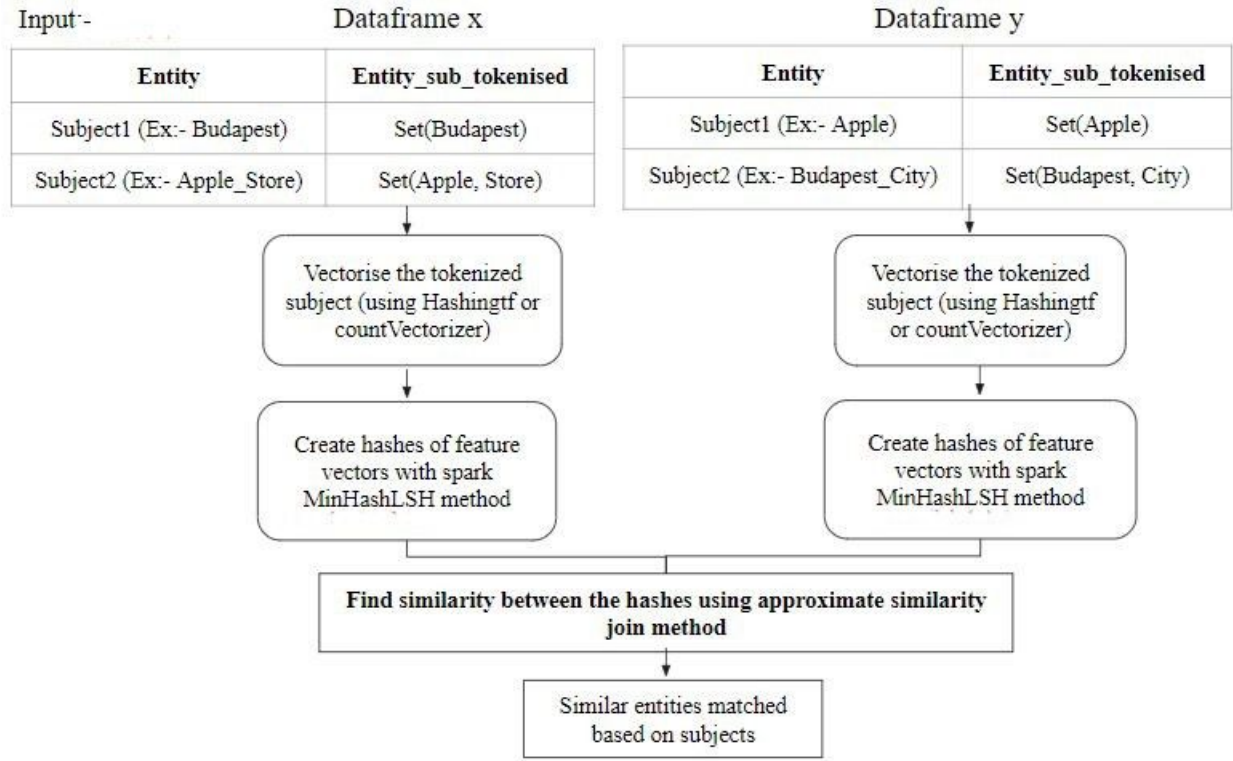


Figure 4.6: Spark MinHash LSH subject

*Definition:- Let the entity collections in two databases be  $E1$  and  $E2$ , then an entity mention  $e1$  in a certain collection be recorded as a tuple  $\langle s, Ts \rangle$ , where  $Ts$  is a set formed by the tokenization of the subject( $s_-$ ), for the specific entity mention  $e1$  and  $s_-$  be the subject for the profile. The Entity collections  $E1$  and  $E2$  contains several entity mentions such that there are no duplicates within an entity collection.*

For each of the entity collections, for every single entity mention, we vectorize its subject tokens using an HashingTF or a CountVectorizer technique and then create hashes of the respective feature vectors using spark MinHashLSH model, keeping the setNumHashTables attribute to 3. Find the similarity between the respective transformed hashes for each of the entity mentions of the two entity collections with a spark approxSimilarityJoin method on an appropriate threshold( $t$ ). The similar entities marked among the two entity collections are computed.

#### 4.3.3.2. Step 2 - Compare the attribute names for matched entities

Since our focus is on the maximum utilization of data, we compare the attribute names of only those two entities that are marked similar based on the similarity of their subjects. We extract the attribute names(i.e. predicates in RDF data) for the similar matched entities and compare their respective predicates by Jaccard similarity thresholds(Jp). Jp, being the Jaccard similarity threshold for predicate level comparison. The Figure 4.7 shows the implementation and Figure 4.8

```
def get_similar_predicates(similar_subj_rdd: RDD[(String, String, String, String)], jSimilarity: Double):
  RDD[(String, List[String], String, List[String], List[String], Double)] = {
    val refined_data_sub = similar_subj_rdd.map(f => {
      val sub1 = f._1
      val s_data1 = f._2
      val sub2 = f._3
      val s_data2 = f._4
      val pred_obj1 = s_data1.split(" , ").toList
      val pred_obj2 = s_data2.split(" , ").toList
      var list_pred1 = List[String]()
      var list_pred2 = List[String]()
      for (x <- pred_obj1) {
        list_pred1 = list_pred1 ++ x.split(":").head
      }
      for (x <- pred_obj2) {
        list_pred2 = list_pred2 ++ x.split(":").head
      }
      val intersect_pred = list_pred1.intersect(list_pred2)
      val union_pred = list_pred1.length + list_pred2.length - intersect_pred.length
      val similarity = intersect_pred.length.toDouble / union_pred.toDouble
      (sub1, pred_obj1, sub2, pred_obj2, intersect_pred, similarity)
    })
    return refined_data_sub.filter(f => f._6 >= jSimilarity)
  }
```

Figure 4.7: Filtering common predicates code snippet

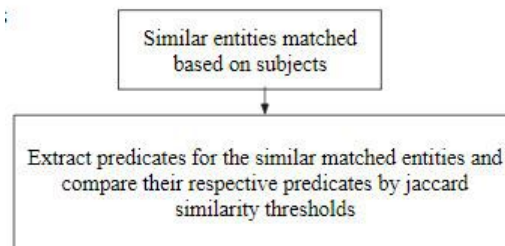


Figure 4.8: Compare the attribute names for matched entities

illustrates this step progressively.



#### 4.3.3.3. Step 3 - Compare the attribute values for intersecting attribute names in matched entities

```
def get_similarity_objects(ds_pred: RDD[(String, List[String], String, List[String], List[String], Double)], threshold_objects: Double):
  RDD[(String, String, Double)] = {
    val mapped_objects = ds_pred.map(f => {
      val sub1 = f._1
      val pred_obj1 = f._2
      val sub2 = f._3
      val pred_obj2 = f._4
      val common_pred = f._5
      var obj1: String = ""
      var obj2: String = ""
      for (x <- pred_obj1) {
        val pred = x.split(":").head
        val obj = x.split(":").last
        if (common_pred.contains(pred))
          obj1 = obj1 + " " + obj
      }
      for (x <- pred_obj2) {
        val pred = x.split(":").head
        val obj = x.split(":").last
        if (common_pred.contains(pred))
          obj2 = obj2 + " " + obj
      }
      val sub_obj1 = obj1.trim().split(" ").toList.distinct
      val sub_obj2 = obj2.trim().split(" ").toList.distinct
      val intersect_obj = sub_obj1.intersect(sub_obj2).length
      val union_obj = sub_obj1.length + sub_obj2.length - intersect_obj
      val similarity = intersect_obj.toDouble / union_obj.toDouble
      (sub1, sub2, similarity)
    })
    return mapped_objects.filter(f => f._3 >= threshold_objects)
  }
```

Figure 4.9: Compare the attribute values for intersecting attribute names code snippet

In the final step, we focus on reducing false positives. Thus, we extract attribute information(i.e. objects in RDF data) for only intersecting attribute names among the two similar marked entity mentions, fitted on Jp. We compare the objects of only intersecting predicates by the Jaccard similarity threshold of objects, namely Jo. This matches the similar entities in two knowledge sources. The Figure 4.9 illustrates the implementation and Figure 4.10 shows this step progressively.

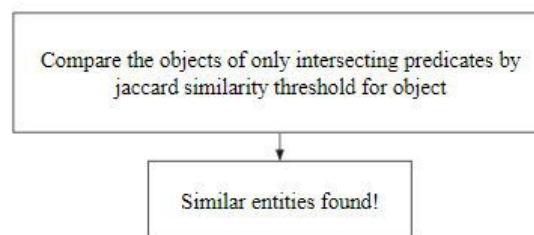


Figure 4.10: Compare the attribute values for intersecting attribute names

#### 4.3.3.4. Example Usecase

##### Step1: Find matching entities based on LSH subjects

Given two Dataframes(x and y), each with two entities as shown in Figure 4.11. The Dataframes x and y have columns indicating entities and their tokenized subjects. The Dataframe x has entities Budapest and Apple.Store and their tokenizations in their respective columns. Similarly, the Dataframe y has entities Apple and Budapest.city and their tokens in their respective columns. Our Approach -3 identifies the entities that are the same in real-world in these two Dataframes.

Input:- Dataframe x		Dataframe y	
Entity	Entity_sub_tokenised	Entity	Entity_sub_tokenised
Subject1 (Ex:- Budapest)	Set(Budapest)	Subject1 (Ex:- Apple)	Set(Apple)
Subject2 (Ex:- Apple_Store)	Set(Apple, Store)	Subject2 (Ex:- Budapest_City)	Set(Budapest, City)

Figure 4.11: Input Dataframes

##### Output :- Similar entities matched based on subjects

The Approach-3 minHashLSH on subjects with setNumHashTables as 3 identifies the below labelled entities as similar. Now they will further be evaluated based on their attributes to find true matches as shown in Figure 4.12.

Entity1	Entity 2
Budapest	Budapest_City
Apple_Store	Apple

Figure 4.12: Similar Entities by MinHash LSH subject

##### Step 2: Compare the predicates for matched entities

Utilizing the attribute level knowledge for each of the paired entities is a keen task for identifying the exact matches and justifying the validity of our pairs (that are paired based on the similarity of their subjects). The predicate level knowledge for each of the paired entities is in the Entity1\_predicates

and Entity2\_predicates respectively as shown in Figure 4.13. We now compute a paired match as valid, only if the Jaccard similarity of the predicates of the paired entities surpasses the user's threshold ( $J_p$  discussed above).

Entity1	Entity1_predicates	Entity2	Entity2_predicates
Budapest	Set(areaCode, rdf:type, country, timezone, populationDensity, postalCode, utcoffset, humidity)	Budapest_City	Set(areaCode, country, elevation, foundingDate, areaTotal, governingBody, populationDensity, timezone, rdf:type)
Apple_Store	Set(foundedBy, industry, keyPerson, product, parentCompany, rdf:type, numberOfLocations, subject, logo, rdf:label, foaf:name)	Apple	Set(genus, rdf:type, division, source, kingdom, class, carbs, fat, fibre, sugar, rdf:label, foaf:name, calcium, phosphorus, potassium, protein)




Figure 4.13: Attribute names comparison for matched entities

#### Output:- Similar entities matched with Jaccard Similarity predicates

Since, the Apple\_Store and Apple do not point to the same real-world object but were marked the same in our Step-1. The introduction of attribute level knowledge checks filter out misleading matches as shown in Figure 4.14. Now, we have Budapest and Budapest\_city with their intersecting predicates to be taken next into our consideration.

Entity1	Entity2	Intersecting_predicates
Budapest	Budapest_City	Set(areaCode, rdf:type, country, populationDensity, timezone)

Figure 4.14: Matched entities with intersecting attribute names filtered with Jaccard similarity on attribute names

#### Step3: Compare the objects for intersecting predicates in matched entities

To post check the true matches and further reduce false positives(if any), we utilize the attribute

values(i.e. objects) of only the intersecting predicates i.e Set(areaCode, rdf:type, country, populationDensity, timezone). Using, only the intersecting predicates overall predicates simplifies our task because we are dealing with heterogeneous information. The knowledge from different sources follow different schemata and have loose bindings. Thus, comparing the similarity of attribute values only for the intersecting predicates is appropriate to find true matches. The attribute values for the paired match are in Entity1\_objects and Entity2\_objects respectively as shown in Figure 4.15. The similarity of the attribute values for the paired match are compared against the user's threshold (Jo discussed above).

Entity1	Entity1_objects	Entity2	Entity2_objects
Budapest	Set (1, City, Place, Location, PopulatedCity, Hungary, 1558465, Central_European_Time)	Budapest_City	Set(1, City, Place, Location, PopulatedCity, Hungary, 1759407, Central_European_Time, Central_European_SummerTime)

Figure 4.15: Attribute values comparison for intersecting predicates

#### Output:- Similar entities found

The true identified matches are as shown in Figure 4.16

Entity1	Entity2
Budapest	Budapest_City



Figure 4.16: Similar Entities

## 4.4. OUR DESIGN

Since the Approach-1 suffers issues with the inconsistencies in the two knowledge bases (as discussed in section 4.3.1.1), we perform our detailed analysis with Approach-2 and Approach-3.

#### 4.4.1. Storage Unit

Storage forms an important part in the design and implementation of every system. We are using the Hadoop file system to store our input data and the computed output file. Our intermediate computed results are computed and stored in Spark RDDs and Dataframes, owing to their distributed fast in-memory computations.

#### 4.4.2. Input and Output for Approach-2 MinHash LSH method (1 or 2 attribute)

The input for our Approach-2 is in CSV format. Three dataset pairs are used, namely DBLP-ACM, DBLP-Scholar, and Abt-Buy datasets. These are available on the [15]. Each of the datasets has its true matches identified stored in a CSV ground truth file, which is used by us to evaluate the accuracy of our methodology. The detailed description of each of the datasets is discussed in section 5.2. The CSV data for each datasets comparison is loaded into Spark Dataframes through CSV data source support<sup>5</sup> for Apache Spark. The entity profiles to be compared are formed from the respective dataframes in the format discussed in section 4.3.2. On applying Approach-2 methodology, the true matches in the two dataframes are identified with the approx similarity join and stored in a spark rdd with their respective distances. Our predicted results are verified against the available ground truth to evaluate and ensure the validity of our approach. Our predicted results are finally written to the Hadoop file system.

#### 4.4.3. Storage and computation for Approach-3 MinHash LSH subjects and Jaccard similarity attributes

##### 4.4.3.1. Input data

The input data for Approach-3 is in the RDF format. The N-triples are loaded in the Spark RDDs and distinct triples are taken into consideration. We are working with Dbpedia 3.0rc and 3.4 datasets. The triples with predicates owl:sameas, wikiPageID, wikiPageRevisionID, wikiPageRevisionLink, wikiPageUsesTemplate, wikiPageHistoryLink, wikiPageExternalLink, wikiPageEditLink, wikiPageExtracted, wikiPageLength, wikiPageModified, wikiPageOutDegree, wikiPageRedirects,

---

<sup>5</sup><https://github.com/databricks/spark-csv>

image, imageWidth, imageCaption, relatedInstance are not used. Also, for the rdf:labels, we are using only labeled text with literal language as "en". The creation of ground truth for the datasets being compared is discussed in section 5.2, where we elaborate details of each of the datasets. The entity profiles are created from the loaded triples as discussed in section 4.3.3.1 and stored in Spark RDDs respectively. Spark framework partitions the data effectively so that it can be processed in a distributed manner.

#### 4.4.3.2. Model description

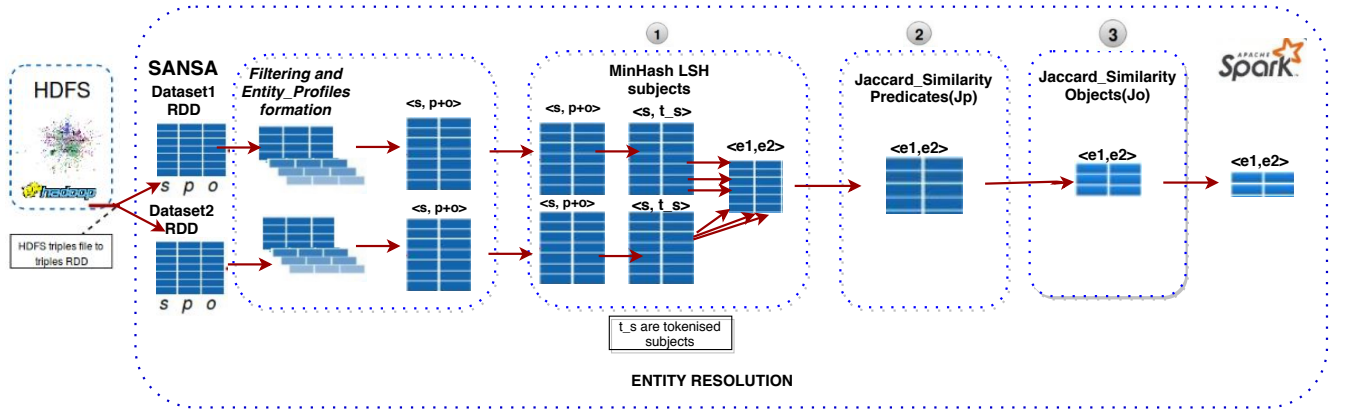


Figure 4.17: Entity Resolution execution pipeline

Figure 4.17 shows the execution pipeline, where we take RDF triples as input, perform data cleansing to form entity profiles subsequently, then apply a series of transformations and actions, and return entities marked similar as output.

1. For each of the entity profile formed, the entity subject data is tokenized and stored in Entity\_sub\_tokenised column in Dataframes x and y respectively as shown in section 4.3.3.4. The tokenized sets in this column are vectorized through HashingTF or CountVectorizer model. The hash values are computed on the generated feature vectors using the MinHashLSH method and approximate similarity join is applied to identify matches based on subject similarity.
2. The paired entities are now deduced and we join the attribute level information for these entities using our initial RDD. The attribute names for each of the paired matches are compared and intersecting predicates are extracted only for the paired matches that surpass the user

threshold ( $J_p$ ) for Jaccard similarity on their attribute names i.e. predicates.

3. The paired matches now filtered out have their intersecting predicates and their respective attribute values identified. We compare the attribute values through Jaccard similarity of these paired matches against the user threshold ( $J_o$ ) to identify the final matches as our marked similar entities.

The source is available at the Github page<sup>6</sup>. The marked similar entities are further evaluated against our created ground truth and stored in the Hadoop file system. The run time for each of the approaches is recorded for their respective datasets and discussed in chapter 5.

---

<sup>6</sup><https://github.com/amrit1210/LSH-based-Entity-Resolution>

## CHAPTER 5

### EVALUATION

In this chapter, we discuss the evaluation of the two successful approaches, namely Approach-2 (MinHash LSH method(1 or 2 attribute)) and Approach-3 (MinHash LSH subjects and Jaccard similarity attributes). We discuss:-

1. The various datasets used in evaluating each of the two approaches respectively.
2. The parameters used for configuration.
3. The comparison of our results with the existing state of the art results based on precision, recall and F-measure values recorded.
4. The execution time involved in our approaches with HashingTF and CountVectorizer model.

#### 5.1. CLUSTER CONFIGURATIONS

To test out the performance of our Scalable Entity Resolution model, based on Spark and Scala, we deploy it on a Spark cluster. We use Spark version 2.2.1 deployed on a Spark Standalone mode with Scala version 2.11.11. A small private cluster of Smart Data Analytics (SDA) consisting of 4 servers is used. The servers have 256 cores in total. Each server runs Xeon Intel CPUs at 2.3GHz, 400GB of disks space, and 256GB of RAM. Each server has installed Ubuntu 16.04.3 LTS (Xenial) and is connected in a Gigabit Ethernet2 network. Each Spark executor has 250GB of memory assigned.

#### 5.2. DATASETS

For the Approach - 2, we are considering three different datasets, namely DBLP-ACM, DBLP-Scholar, and Abt-Buy dataset. These datasets are used in the [3] publication. The link to the datasets is attached at<sup>1</sup>. We compare our results with their result for 1-2 attribute approach. The dataset DBLP-ACM and DBLP-Scholar are bibliographic datasets while Abt-Buy is from the

---

<sup>1</sup>[https://dbs.uni-leipzig.de/research/projects/object\\_matching/benchmark\\_datasets\\_for\\_entity\\_resolution](https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution)



e-commerce domain. The number of entities and attributes involved in each of the datasets is described in table 5.1. The comparisons in these three different datasets possess different levels of difficulty based on their dataset characteristics.

1. The DBLP-ACM [3] match entity task is expected to be of low difficulty because it deals with publication entities from two well-structured bibliographic data sources (the Digital Bibliography Library Project [DBLP] and the ACM digital library) that are at least partially under manual curation. The selected DBLP and ACM entities cover the same computer science conferences and journals.
2. The second dataset i.e DBLP-Scholar [3] matches DBLP publications with publications from the entity search engine Google Scholar (Scholar). Scholar automatically extracts its publication entities from full-text documents crawled from the Web. This data has many quality problems particularly, heterogeneous representations of author lists or venue names, misspellings, and extraction errors.
3. The e-commerce tasks(Abt-Buy [3]) deal with sets of related product entities from online retailers Abt.com and Buy.com.

Table 5.1: Evaluation dataset description

Match task	Source size (Number of entities)		
Attributes	Sources	Source1	Source2
Title, Authors, Venue, Year	DBLP-ACM	2,616	2,294
Title, Authors, Venue, Year	DBLP-Scholar	2,616	64,263
Name, Description, Manufacturer, Price	Abt-Buy	1,081	1,092

For Approach-3, we are using DBpedia datasets. The medium and large-sized datasets are taken to evaluate the scalability of our code. We have taken different versions of Dbpedia for comparison, in particular - for medium size dataset, we are comparing the two infobox versions, namely Infobox 3.0rc (containing 9,91,933 entities) with Infobox 3.4 (containing 19,40,631 entities). The Infobox 3.0rc is 2.7 GB in size while Infobox 3.4 is 5.8 GB in size. The ground truth for calculating precision, recall, and F-measure is constructed by considering the matches as entities with exactly the same URL. The inspiration for the dataset and ground truth construction is from [11]. Another

large-sized DBpedia dataset, we are comparing DBpedia 3.0rc (containing 47,81,249 entities) with Infobox 3.4 (containing 19,40,631 entities). The DBpedia 3.0rc is 19GB in size while Infobox 3.4 is 5.8GB in size as aforementioned. The ground truth constructed for the medium-sized dataset is used here also for evaluation. The summarised details of these datasets as represented in table 5.2.

Table 5.2: Evaluation DBpedia dataset description

Dataset Categorization	Dataset Size	Entities in Dataset1	Entities in Dataset2
Medium size	8.5GB	Infobox 3.0rc (9,91,933)	Infobox 3.4 (19,40,631)
Large size	24.8GB	DBpedia 3.0rc (47,81,249)	Infobox 3.4 (19,40,631)

### 5.3. CONFIGURATION PARAMETERS

The adjustment of configuration parameters is of utmost importance to achieve the desired results from any algorithm. In our setting, we have the following three configuration parameters discussed.

#### 1. Similarity threshold for ApproxSimilarityJoin

As discussed in section 4, the ApproxSimilarityJoin returns two entities with the calculated distance between them, if the distance between them is less than the Similarity threshold set by the user. Thus, the similarity threshold set by a user plays an important role in determining similar entities. Both, the similarity threshold and the calculated distance between two entities is a real number ranging between zero and one. The smaller the threshold set by the user, the ApproxSimilarJoin returns more closely related entities. In our task, for different datasets, we have set different values for the Similarity threshold.

#### 2. Jaccard Similarity threshold for Predicates(Jp)

This parameter is only set for Approach-3. As discussed in section 4, for the MinHash LSH subject similarity retrieved entities, we check the Jaccard similarity of the predicates and retrieve only those pairs that have Jaccard similarity more than the user-determined parameter Jp. This parameter is also a real number with the value ranging between zero and one. The more the Jaccard similarity for predicates(Jp) in our model, the closer the matched pair of entities. In our task, for

a medium and large dataset, we have adjusted the similarity predicate to be 0.15 for both medium and large-sized DBpedia datasets. Although, we have tried resolving two different versions of the same dataset i.e. DBpedia, but it is not an easy task because for medium-sized dataset 9,48,698 entities very newly added between infobox 3.0rc and infobox 3.4. Only, 5 percent of attribute-value pairs are common between these two different versions of datasets.

### 3. Jaccard Similarity threshold for Objects(Jo)

This parameter is set only for Approach-3. As discussed in section4, for the entity pairs fitted to the Jaccard Similarity of predicates(Jp) with their intersecting predicates, we calculate the Jaccard similarity of their attribute values i.e. objects and retrieve similar entities as the entities whose calculated similarities surpass the user-defined Jaccard Similarity of objects(Jo). This parameter and the calculated Jaccard similarities for the intersecting predicates are both real numbers ranging between zero and one. In our task, we have adjusted this parameter to 0.25 for both medium and large-sized datasets.

## 5.4. EVALUATION PARAMETERS

Figure 5.1 represents the confusion matrix<sup>2</sup>, that helps in the understanding of precision and recall equations described in the following sections.

		Actual	
		Positive	Negative
Predicted	Positive	<b>True Positive</b>	<b>False Positive</b>
	Negative	<b>False Negative</b>	<b>True Negative</b>

Figure 5.1: Confusion Matrix<sup>2</sup>

<sup>2</sup><https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c>

#### 5.4.1. Precision

The Precision<sup>2</sup> is a measure that identifies the percentage of our results that are relevant. It is defined as the number of true positives divided by the number of true positives plus the number of false positives. True positives are data point classified as positive by the model that actually are positive (i.e. they are correct) and False positives are the cases that model incorrectly labels as positive, but are actually negative.

$$Precision = \frac{true\_positive}{true\_positive + false\_positive} \quad (5.1)$$

#### 5.4.2. Recall

The Recall<sup>2</sup> is a measure that identifies the percentage of total relevant results correctly classified by our algorithm. the number of true positives divided by the number of true positives plus the number of false negatives. False negatives are data points the model identifies as negative that actually are positive (i.e. they are incorrect). Recall can be thought as of a model's ability to find all the data points of interest in a dataset.

$$Recall = \frac{true\_positive}{true\_positive + false\_negative} \quad (5.2)$$

#### 5.4.3. F-measure

The F-measure<sup>2</sup> is the harmonic mean of precision and recall taking both metrics into account in the following equation. The harmonic mean is used instead of a simple average because it punishes extreme values.

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \quad (5.3)$$

## 5.5. EVALUATION RESULTS

The prime focus of this section is on examining the implementation results of our Approach-2 and Approach-3. We will compare and validate the accuracy of our approaches with the state of the art methods. Both, approaches are implemented with HashingTF and CountVectorizer model. So, we will be comparing execution time involved in each of the strategies on different datasets.

### 5.5.1. Discussion of results for Approach-2

We have completed our experiments on Spark client mode with the cluster configuration described in table 5.3.

Table 5.3: Spark configuration parameters in cluster mode

Configuration parameters	Value
Driver Memory(GB)	50
Executor Memory(GB)	80
Total number of cores	190

The DBLP-ACM dataset contains 2,616 entities and 2,294 entities respectively with the attributes title, authors, venue and year. For the 1-attribute approach, we have taken the attribute(title) and for 2-attributes, we have considered the attributes(title and authors), as considered in the state of the art approaches. A similarity threshold of 0.25 is considered in 1-attribute comparison and 0.28 for 2-attribute comparison during ApproxSimilarityJoin. The precision, recall, and f-measure are as shown in table 5.4 for 1-attribute and 2-attribute comparison respectively. Although the state of the art methods attain slightly high precision, recall and F-measure values for 1-attribute comparison, our methodology still achieves comparable results in less time with no effort involved in labeling data for training the model and no need of multiple iterations. Our methodology directly processes the input Dataframes and actively returns similar entities with efficiency.

Table 5.4: Evaluation results for DBLP-ACM dataset

	1-attribute(Title)		2-attribute(Title, Authors)	
Number of attributes	State of the art	MinHash LSH	State of the art	MinHash LSH
Precision (%)	94.9	85.88	96.9	92.05
Recall (%)	97.3	95	87.8	93
F-measure (%)	96.1	90.21	92.1	92.52

The DBLP-Scholar dataset contains 2,616 entities and 64,263 entities respectively with the attributes title, authors, venue and year. Since, it is again a bibliographic dataset, for a 1-attribute approach, we have taken the attribute(title) and for 2-attributes, we have considered the attributes(title and authors), as considered in the state of the art approaches. A similarity threshold of 0.15 is considered in 1-attribute comparison and 0.42 for 2-attribute comparison during ApproxSimilarityJoin. The precision, recall, and f-measure are as shown in table 5.5 for 1-attribute and 2-attribute comparison respectively. Here, we attain similar precision, recall and F-measure values as state of the art methods. This adheres to the validity and accuracy of our approach and also goes in line with aborting the blocking stage and directly processing the entities for comparison.

Table 5.5: Evaluation results for DBLP-Scholar dataset

	1-attribute(Title)		2-attribute(Title, Authors)	
Number of attributes	State of the art	MinHash LSH	State of the art	MinHash LSH
Precision (%)	74.1	79.0	77.5	79.86
Recall (%)	91.7	87.0	84.8	83
F-measure (%)	82.0	82.31	81.0	81.40

The Abt-Buy dataset is an e-commerce website crawled dataset containing 1,081 entities and 1,092 entities respectively with the attributes name, description, manufacturer and price. For the 1-attribute approach, we have taken the attribute(name) and for 2-attributes, we have considered the attributes(name and description), as considered in the state of the art approaches. This dataset contains a lot of inconsistencies, noisy and missing values. A similarity threshold of 0.5 is considered

in 1-attribute comparison and 0.685 for 2-attribute comparison during ApproxSimilarityJoin. The precision, recall, and f-measure is as shown in table 5.6 for 1-attribute and 2-attribute comparison respectively. Although the state of the art methods attain slightly better F-measures for both 1-attribute and 2-attribute approaches, we argue that our model is better because it does not suffer huge variations in precision and recall parameters. This explains that the LSH model is an effective and efficient model to remove block building and training overheads with small compromises on accuracy depending on datasets.

Table 5.6: Evaluation results for Abt-Buy dataset

	1-attribute(Name)		2-attribute(Name, Description)	
Number of attributes	State of the art	MinHash LSH	State of the art	MinHashLSH
Precision (%)	78.4	35.80	90.65	27.63
Recall (%)	36.4	48	17.6	31
F-measure (%)	49.7	41.01	29.5	29.21

### 5.5.2. Execution Times for Approach-2 datasets with HashingTF and CountVectorizer models

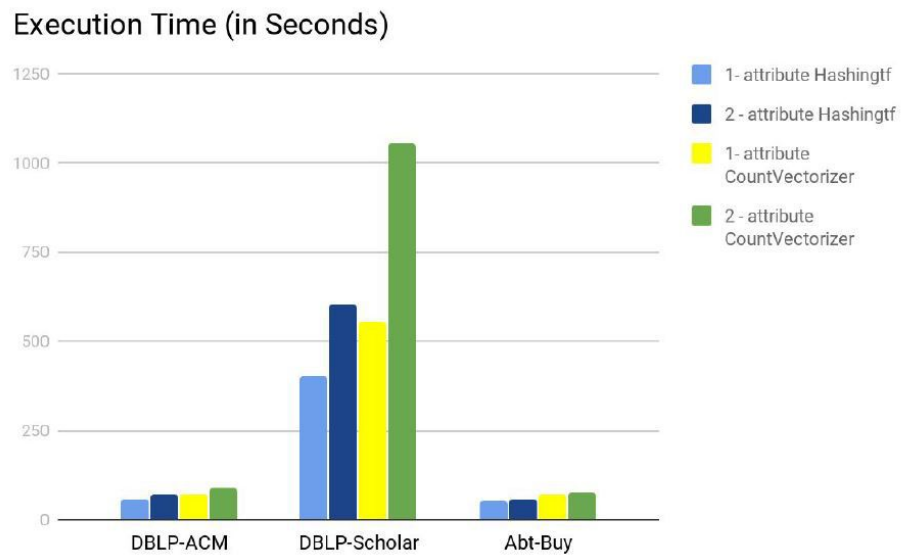


Figure 5.2: Execution Time (in seconds) with Approach-2

The execution time is a crucial parameter to analyze the effectiveness and efficiency of our algorithm. We have implemented and evaluated our approach by both HashingTF and CountVectorizer model. As we have closely discussed the pros and cons of both the models in section 4.2, Here we can see its effect on the execution times for our different datasets. For approach-2, we have considered three small-sized datasets discussed above. Figure 5.2 shows the execution time (in seconds) involved in the 1 and 2 attribute approach with both the models. The DBLP-Scholar dataset takes more time than the other two datasets because the Scholar dataset has 24 times more entities to be processed than DBLP dataset. Clearly, 2-attribute processing takes little more time than 1-attribute owing to the increase in the size of data to be processed. This is also more evident with the DBLP-Scholar execution time variations for 1 and 2 attributes. Thus, with the increase in the number of entities and data to be processed, the execution time also increases. The HashingTF model executes much faster in comparison to the CountVectorizer model as it directly transforms the tokenized text to feature vectors and does not need to scan the data twice unlike CountVectorizer, once for building the model and then later to transform it to feature vectors. With little collisions into the same bucket, it works fast and best.

### 5.5.3. Discussion of results for Approach-3

As discussed in detail in section 5.2, for medium-sized dataset, we are comparing Infobox 3.0rc dataset (containing approx 9 million entities) against Infobox 3.4 dataset (containing approx 19 million entities) and for large-sized dataset, we are considering entire DBpedia 3.0rc dataset (containing approx 47 million entities) and Infobox 3.4 dataset. The same ground truth is used for evaluating both. The precision, recall, and f-measure values recorded as shown in table 5.7. To attain these results, the configuration parameters used are as discussed below:-

1. A 0.10 similarity threshold for ApproxSimilarityJoin is adjusted for the initial phase to determine the subject similarity between the entities in two Dataframes. The lower the threshold set here by the user, the more similar are their subjects.
2. A Jaccard Similarity threshold(Jp) of 0.15 is considered among the predicates for the entities paired on the subject similarity basis. Although we attain good precision, recall and f-measure scores at this level, we move one step further intending to reduce the false positives in our



results. This would vary with different datasets.

3. A Jaccard Similarity threshold(Jo) of 0.25 is considered for computing the similarities among the objects of the intersecting predicates for the entities matched at phase - 2 of our algorithm. We do not attain huge viable variations in precision, recall, and f-measure values at this level (comparing to previous stage values recorded) of our algorithm (we do not have many false positives), but it depends entirely on the dataset.

Table 5.7: Evaluation Results for DBpedia datasets

	Jaccard Similarity Predicates	Jaccard Similarity Objects
Precision (%)	99.75	99.86
Recall (%)	86	85
F-measure (%)	92.36	91.83

#### 5.5.4. Discussing Example cases from Evaluation Results

This section discusses the results for Approach-3, applied on Dbpedia(medium) dataset cross-referencing entities between Infobox 3.0rc and Infobox 3.4. We have closely observed the similar entity matches found at each step of the algorithm and how they are filtered out in successive steps.

##### 5.5.4.1. Analyzing Step-1- Find matching entities based on LSH subjects

Table 5.8: Evaluation Results - Similar Entities matched by minHash LSH subject

Infobox 3.0rc	Infobox 3.4
Entity1	Entity2
Ali_Baban	Ali_Baba
Ali_Babacan	Ali_Baba
Ali_Baba	Ali_Baba
Blackberry	Blackberry
Blackberry	BlackBerry
Superman	Superwoman

The table 5.8 shows some of the match pairs formed after the Step-1 of the algorithm. The Entity1 column contains entities in Infobox 3.0rc referenced to Entity2 column, containing entities from Infobox 3.4. These results are recorded when a similarity threshold of 0.10 is set for Approx-SimilarityJoin during MinHashLSH subjects. Now they will further be evaluated based on their attribute names to find true matches in the subsequent step.

#### 5.5.4.2. Analyzing Step 2 : Compare the predicates for matched entities

Table 5.9: Evaluation Results - Predicate knowledge for entity matches by MinHash LSH subjects

Infobox 3.0rc		Infobox 3.4	
Entity1	Entity1_predicates	Entity2	Entity2_predicates
Ali_Baban	Set(title, before, years, after)	Ali_Baba	Set(nihongoProperty)
Ali_Babacan	Set(title, before, years, after, birthDate, birthPlace, office, termStart, termEnd, religion, party)	Ali_Baba	Set(nihongoProperty)
Ali_Baba	Set(nihongoProperty, forProperty)	Ali_Baba	Set(nihongoProperty)
Blackberry	Set(twoOtherUsesProperty, classis, ordo, familia, genus, subdivisionRanks, regnum, subfamilia, subdivision, color, width, subgenus, divisio)	Blackberry	Set(subdivison, familia, genus, subgenus, subfamilia, classis, divisio, regnum, name, subdivisionRanks, ordo, twoOtherUsesProperty)
Blackberry	Set(twoOtherUsesProperty, classis, ordo, familia, genus, subdivisionRanks, regnum, subdivision, subfamilia, color, width, subgenus, divisio)	BlackBerry	Set(thisProperty, name, screen, ringtone, memory, connectivity, weight, size, commonsProperty, networks, pdfinkProperty)
Superman	Set(powers, aliases, alliances, baseOfOperations, penciller, publisher, caption, realName, debut, characterName, homeworld, date, issue, title, story, creator, artist, writer, volume, inker, id)	Superwoman	Set(panel, page, publisher, date, issue, title, sortkey, villian, hero, subcat, cat, alterego, debut, noimage, alliances, creator, powers, characterName)

The predicate level knowledge for each of the paired entities in Step-1 is as shown in the columns Entity1\_predicates and Entity2\_predicates in Table 5.9 respectively. A Jaccard Similarity threshold(Jp) of 0.15 is adjusted over the predicates of matched pairs in Step-1. We now compute a paired matches valid, only if the Jaccard similarity of the predicates of the paired entities surpasses the user's threshold (Jp discussed above).

**Observations:-**

1. As, we can clearly see the predicates for the entity Ali\_Baban and Ali\_Babacan from Infobox 3.0rc are dissimilar with the entity mention Ali\_Baba from Infobox 3.4. Thus, they does not surpass the threshold Jp and are eliminated.
2. Similarly the attribute-level knowledge for Blackberry and BlackBerry can be seen as two disjoint sets and thus, they are also eliminated.

Now, at this level we are sure that these entity matches from Step-1 are not the exact true matches for us to resolve. So, the misleading entity matches are eliminated at this point. Thus, the other paired matches that surpass the Jp for their attribute-level knowledge are as shown in Table 5.10, with their common predicates identified in the Intersecting\_predicates column. To identify the true matches from these paired entities, we will further evaluate the attribute values of the intersecting predicates in our final step.

Table 5.10: Evaluation Results - Intersecting predicates for similar entities matched with Jaccard Similarity predicates threshold(Jp)

Infobox 3.0rc	Infobox 3.4	
Entity1	Entity2	Intersecting_predicates
Ali_Baba	Ali_Baba	Set(nihongoProperty)
Blackberry	Blackberry	Set(subdivison, familia, genus, subgenus, subfamilia, classis, divisio, regnum, subdivisionRanks, ordo, twoOtherUsesProperty)
Superman	Superwoman	Set(publisher, debut, date, issue, title, alliances, creator, powers, characterName)

### 5.5.4.3. Analyzing Step3: Compare the objects for intersecting predicates in matched entities

To post check the paired matches and further reduce false positives(if any), we evaluate the similarity of attribute values(i.e. objects) of only the intersecting predicates. The attribute values for the paired match are in Entity1\_objects and Entity2\_objects columns respectively as shown in Table 5.11. A Jaccard Similarity threshold(Jo) of 0.25 is adjusted to check the similarity of the attribute values for the paired matches.

Table 5.11: Evaluation Results - Values of Intersecting predicates for entity matches by Jaccard Similarity threshold for predicates (Jp)

Infobox 3.0rc		Infobox 3.4	
Entity1	Entity1_objects	Entity2	Entity2_objects
Ali_Baba	Set(Ali Baba and the Forty Thieves, Aribaba to Yonjuppiki no Tozoku)	Ali_Baba	Set(Ali Baba and the Forty Thieves, Aribaba to Yonjuppiki no Tozoku)
Blackberry	Set(the wireless e-mail device BlackBerry, other uses, Rosaceae, the fruit, Rosoideae, Magnoliopsida, Flowering_plant, Rubus, Eubatus, Plant, Species, Rosales, Blackberry (disambiguation))	Blackberry	Set(the wireless e-mail device, BlackBerry, Rosaceae, other uses, the fruit, Rosoideae, Magnoliopsida, Flowering_plant, Rubus, Rubus (formerly Eubatus), Species, Plant, Rosales, Blackberry (disambiguation))
, Superman	Set(DC_Comics, February, 1968, Superman_%28comic_book%29 Infinite_Crisis, 2003-12, Batman, Action Comics, JSA: Classified, 2007-07, Action Comics Annual, Justice League of America, Superman:_Whatever_Happened_to_the_Man_of_Tomorrow%3F, All Star Comics, Joe_Shuster, All-American_Publications, Powers_and_abilities_of_Superman, Team_Superman, Justice_League, Daily_Planet, Jerry_Siegel, Action_Comics_1, Superman)	Superwoman	Set(DC_Comics, May 2, 52_%28comic_book%29, Crime Syndicate of America, Invulnerability, Superwoman, Gardner_Foy, Mike_Sekowsky, Superhuman strength, *Super strength, speed, and endurance * Flight *, Superhuman speed, Superior hand-to-hand combatant * reflexes, stamina, Heat Vision, Lasso of submission , a weaponized [[Skills and resources)

### Observations:-

Clearly, the attribute values of Superman and Superwoman doesnot surpass the Jaccard Similarity threshold for objects(Jo). Thus, our approach attempts to diminish the possibility of declaring two different entities as same. So, this pair is eliminated and the other pairs are identified as true matches. The identified true matches are as shown in Table 5.12.

Table 5.12: Evaluation Results - Similar entities found

Infobox 3.0rc	Infobox 3.4
Entity1	Entity2
Ali_Baba	Ali_Baba
Blackberry	Blackberry

#### 5.5.5. Execution Times for Approach-3 datasets with HashingTF and CountVector-izer models

##### Execution Time (in Minutes)

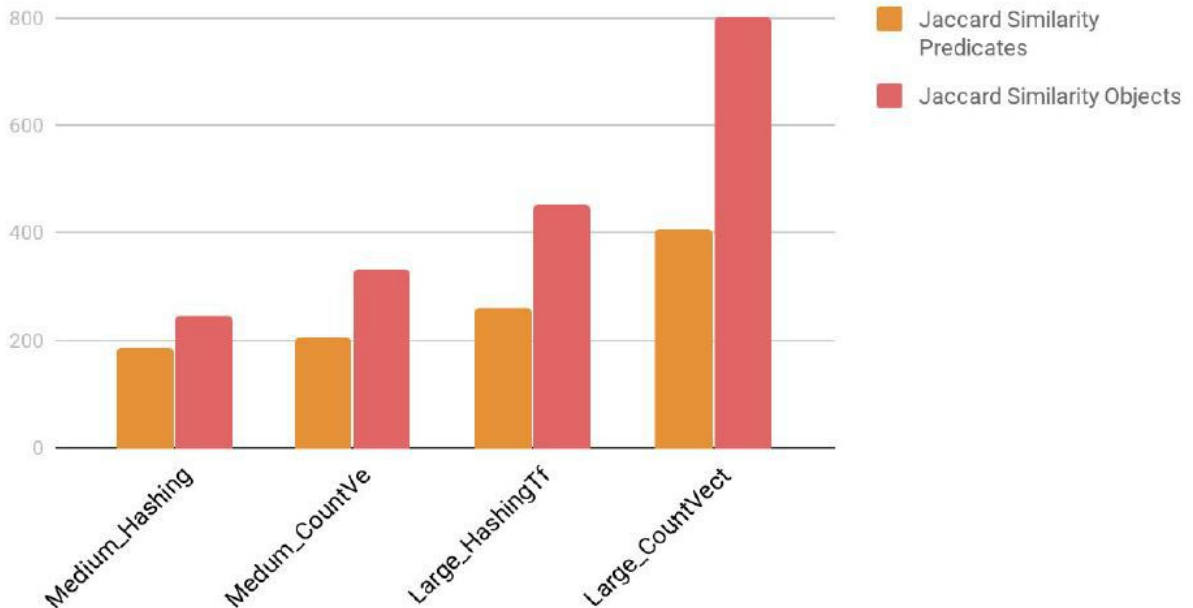


Figure 5.3: Execution Time (in minutes) with Approach-3

We have implemented and evaluated our approach for both medium and large-sized DBpedia datasets with HashingTf and CountVectorizer models respectively. In our algorithm, the most time is consumed at the approxSimilarityJoin in step-1. Figure 5.3 shows the execution time (in minutes) for both the datasets on level-2 and level-3 our approach i.e with the Jaccard Similarity threshold(Jp) set for the predicates of subject similarity-based pairs and with the Jaccard Similarity threshold(Jo) for objects for finally computing the true matches. The execution time varies with the size of the dataset and with the model used for forming the feature vectors with the text. The HashingTF model with large sized dataset executes incomparably less time than the CountVectorizer model with the medium-sized dataset. The CountVectorizer model takes twice as much time as the HashingTF model for large-sized dataset. Thus, the HashingTF model based solution is the fastest prototype for our algorithm. For larger datasets evaluating step-3 of our algorithm i.e. filtering with the Jaccard Similarity for Objects takes much higher time(especially with the CountVectorizer model). So, it depends on the user's choice and the dataset to proceed to step-3 after step-2. With prior knowledge of the dataset and the precision, recall and f-measure values after step 2, the user can abort the step-3 compromising with a small number of false positives and reducing the overheads of time consumption.

## CHAPTER 6

### CONCLUSION

In our work, we have developed an effective, efficient, generic and scalable approach for entity resolution. We have implemented and experimented with two different approaches, namely Approach-2 and Approach-3 discussed in section-4 with different datasets and conclude that the HashingTf model performs faster than the CountVectorizer model in all cases and thus, works the best for our approaches. Our approaches reach our prior mentioned goals dealing with the heterogeneity of data and handling structured as well as unstructured data. With the advent of finding intersecting predicates in approach-3 and comparing the similarities in their attribute values, we solve the problem of missing and noisy information to a great extent.

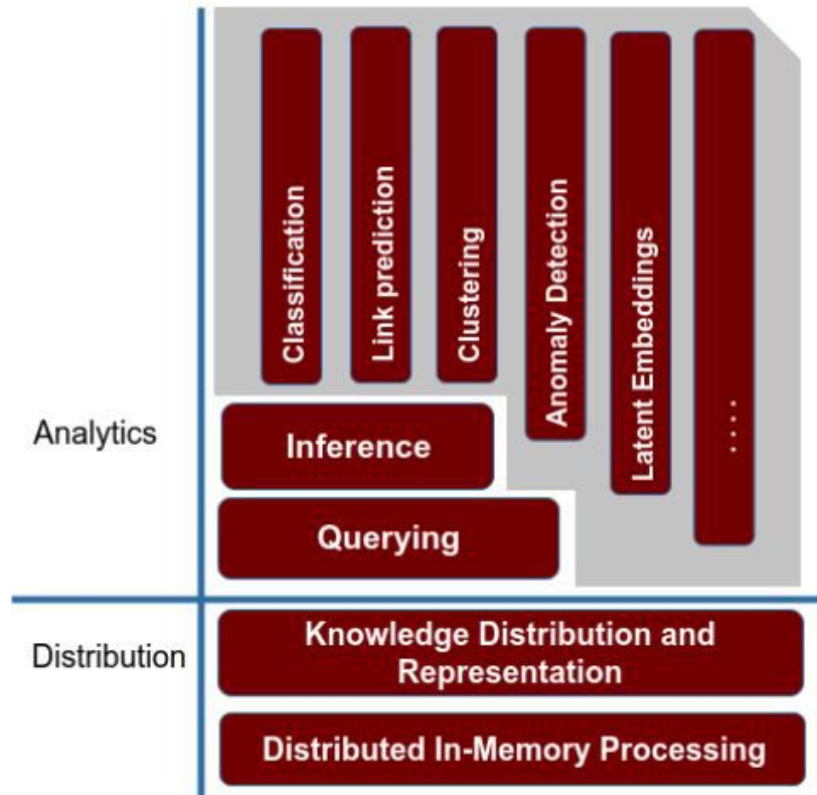


Figure 6.1: SANSA stack<sup>1</sup>

With little i.e. 5-7% or no compromises in accuracy (depending on the dataset), our approach performs entity resolution in linear time competing with the existing approaches. Our methodology successfully removes the block building stage in entity resolution. Thus, completely reducing the time incurred in this step. Also, we abort the effort of labeling data for supervised training methods and the time involved in multiple iterations of training the model. Hence, the task of entity resolution in RDF data could be simplified and leveraged with our suggested approach.

Our thesis work will be an addition for Semantic Analytics Stack(SANSA)<sup>1</sup>, a distributed computing framework (specifically Spark and Flink) with the semantic technology stack, shown in figure 6.1. A module named as "Scalable Entity Resolution" will be added to the ML layer in SANSA Stack. This will help in identifying duplicate entities in RDF data, simplifying the task of data integration effectively.

The current thesis work can be further extended. Datasets with ids as subjects instead of entity names in URL are difficult to compare with our approach. In our future work, this can be extended by taking into consideration the rdf:labels for such datasets. We are currently performing our work on DBpedia datasets and can be extended to other RDF datasets like Freebase, YAGO, and Wiki data, etc. A major challenge occurs in evaluating the scalability of our approaches as there are no ground truths available for large datasets. Thus, the creation of ground truth for large-sized datasets is another major task to enhance the current work. We can also improve the evaluation of our approach by performing analysis of entity resolution with different DBpedia language datasets. It would also be exciting to perform the current work on other distributed frameworks like Flink as it automatically manages operations like data partitioning and caching that are otherwise set manually in Spark. Performing a comparative analysis of our work on the Spark and Flink framework will provide better insights into our algorithm.

---

<sup>1</sup><http://sansa-stack.net/>



## REFERENCES

- [1] G. Pandey, *The Semantic Web: An Introduction and Issues*. International Journal of Engineering Research and Applications, 2012.
- [2] P. Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.
- [3] H. Köpcke, A. Thor, and E. Rahm, *Learning-Based Approaches for Matching Web Data Entities*. IEEE Internet Computing, July/August 2010.
- [4] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl, *A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces*. IEEE Transactions on Knowledge and Data Engineering, 2013.
- [5] G. Papadakis, V. Efthymiou, T. Palpanas, G. Papastefanatos, and K. Stefanidis, *Parallel Meta-Blocking for Scaling Entity Resolution over Big Heterogeneous Data*. Information Systems - Elsevier, 2017.
- [6] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl, *Eliminating the Redundancy in Blocking-based Entity Resolution Methods*. Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries, 2011.
- [7] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl, *Beyond 100 Million Entities: Large-scale Blocking-based Resolution for Heterogeneous Data*. WSDM, 2012.
- [8] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis, *Scaling Entity Resolution to Large, Heterogeneous Data with Enhanced Meta-Blocking*. EDBT, 2016.
- [9] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl, *Meta-Blocking: Taking Entity Resolution to the Next Level*. IEEE TKDE, 2014.
- [10] J. Fisher, P. Christen, Q. Wang, and R. Erhard, *A clustering-based framework to control block sizes for Entity Resolution*. Proceedings of the 21th ACM, 2015.
- [11] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser, *Efficient entity resolution for large heterogeneous information spaces*. WSDM, 2011.

- [12] O. Benjelloun et al, *Swoosh: A generic approach to Entity Resolution*. VLDBJ 18(1), 2009.
- [13] <https://spark.apache.org/docs/latest/ml-features.html>, *Extracting, transforming and selecting features*.
- [14] [eng.uber.com/lsh/](http://eng.uber.com/lsh/), *Detecting Abuse at Scale: Locality Sensitive Hashing at Uber Engineering*.
- [15] [https://dbs.uni-leipzig.de/research/projects/object\\_matching/benchmark\\_datasets\\_for\\_entity\\_resolution](https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution), *Benchmarked Entity Resolution Datasets used in Approach-2*.