UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Khalil Ur Rehman

Microdata Deduplication with Spark

Master's Thesis (30 ECTS)

Supervisor(s): Dr. Peep Küngas

Tartu 2016

Acknowledgment

All praises and thanks to Allah. Peace and blessing on His Messenger and Servant Mohammad.

I am thankful to Dr. Peep Küngas, supervisor of this thesis project, for his continuous support and guidance. The thesis would not have been possible without his guidance and courage. I present my gratitude to Dr. Marlon Gerardo Dumas Menjivar, program manager and my professor for his constant support and guidance throughout the Master program. I am grateful to all my teachers, professors, and all staff of the University of Tartu, specifically Institute of Computer science, who all play an important role in running smoothly the activities of University and helping in spreading of knowledge.

I am thankful to the IT Academy for their continuous financial support during the Master degree.

Finally, and most importantly, I would like to thank my wife for her patience, encouragement, love and support during these two years and the years before. I thank my parents for their kindness and prayers.

Microdata Deduplication with Spark

Abstract:

The web is transforming from traditional web to web of data, where information is presented in such a way that it is readable by machines as well as human. As a part of this transformation, every day more and more websites implant structured data, e.g. product, person, organization, place etc., into the HTML pages. To implant the structured data different encoding vocabularies, such as RDFa, microdata, and microformats, are used. Microdata is the most recent addition to these structure data embedding standards, but it has gained more popularity over other formats in less time. Similarly, progress has been made in the extraction of the structured data from web pages, which has resulted in open source tools such as Apache Any23 and non-profit Common Crawl project. Any23 allows extraction of microdata from the web pages with less effort, whereas Common Crawl extracts data from websites and provides it publically for download. In fact, the microdata extraction tools only take care of parsing and data transformation steps of data cleansing. Although with the help of these state-of-the-art extraction tools microdata can be easily extracted, before the extracted data used in potential applications, duplicates should be removed and data unified. Since microdata origins from arbitrary web resources, it has arbitrary quality as well and should be treated correspondingly.

The main purpose of this thesis is to develop the effective mechanism for deduplication of microdata on the web scale. Although the deduplication algorithms have reached relative maturity, however, these algorithm needs to be executed on specific datasets for fine-tuning. In particular, the need to identify the most suitable length of sorting key in sorted-based deduplication approach. The present work identifies the optimum length of the sorting key in the context of extracted product microdata deduplication. Due to large volumes of data to be processed continuously, Apache Spark will be used for implementing the necessary procedures.

Keywords:

Metadata, structured data, microdata, microformats, Apache Any23, deduplication, entity resolution

CERCS: P170

Struktureeritud andmetest duplikaatide eemaldamine Apache Spark'iga

Lühikokkuvõte:

Üha rohkem on avaldatakse veebis struktureeritud sisu, mis on loetav nii inimeste kui masinate poolt. Tänu otsimootorite loojatele, kes on defineerinud standardid struktureeritud sisu esitamiseks, teevad järjest rohkemad veebisaidid osa oma andmetest, nt toodete, isikute, organisatsioonide ja asukohtade kirjeldused, veebis avalikuks. Selleks kasutatakse RDFa, microdata jms vorminguid. Microdata on üks viimastest vormingutest ning saanud populaarseks suhteliselt lühikese aja jooksul. Sarnaselt on arenenud tehnoloogiad veebist struktureeritud sisu kättesaamiseks. Näiteks on Apache Any23, mis võimaldab veebilehtedest microdata andmeid eraldada ja linkandmetena kättesaadavaks teha. Samas pole struktureeritud andmete veebist kättesaamine enam suureim tehniline väljakutse. Nimelt on veebist saadud andmeid enne kasutamist vaja puhastada - eemaldada on vaja duplikaadid, lahendada ebakooskõlad ning hakkama tuleb saada kaebamääraste andmetega.

Käesoleva magistritöö peamiseks fookuseks on efektiivse lahenduse loomine veebisleiduvatest linkandmetest duplikaatide eemaldamine suurte andmekoguste jaoks. Kuigi deduplikeerimise algoritmid on saavutanud suhtelise küpsuse, tuleb neid konkreetsete andmekomplektide jaoks siiski peenhäälestada. Eelkõige tuleb tuvastada sobivaim võtme pikkus kirjete sortimiseks. Käesolevas töös tuvastatakse optimaalne võtme pikkus veebisleiduvate tooteandmete deduplikeerimise kontekstis. Suurte andmemahtude tõttu kasutatakse Apache Spark'i deduplikeerimise hajusalgoritmide realiseerimiseks.

Võtmesõnad:

Struktureeritud sisu, linkandmed, duplikaatide eemaldamine, microdata, microformats, Apache Any23

CERCS: P170

Table of Contents

List of Figures

List of Tables

1 Introduction

Microdata is the most recent format of embedding metadata, also called structured data, into web pages, which is developed in the context of HTML5 [1]. Microdata is a Web Hypertext Application Technology Working Group Standard (WHATWG) specification, which is used to embed metadata into HTML contents [2]. For example, using the HTML tag's properties, such as *itemtype*, *itemscope* and *itemprop* provided in microdata specification, common schema.org entities like a person, movie and location could be inserted into the HTML contents efficiently. To broad the concept of microdata provided by WHATWG, famous search engines, i.e. Google, Yahoo, Yandex, and Microsoft, initiated Schema.org [1]. Schema.org is an extension to microdata vocabulary and provides more strength to embed a broad number of data entities into the web contents. Schema.org also provides the vocabulary to depict the relationships among different entities and make the whole picture understandable as Creative Work [3]. Figure 1 shows the concept of describing Creative work using Schema.org microdata format.

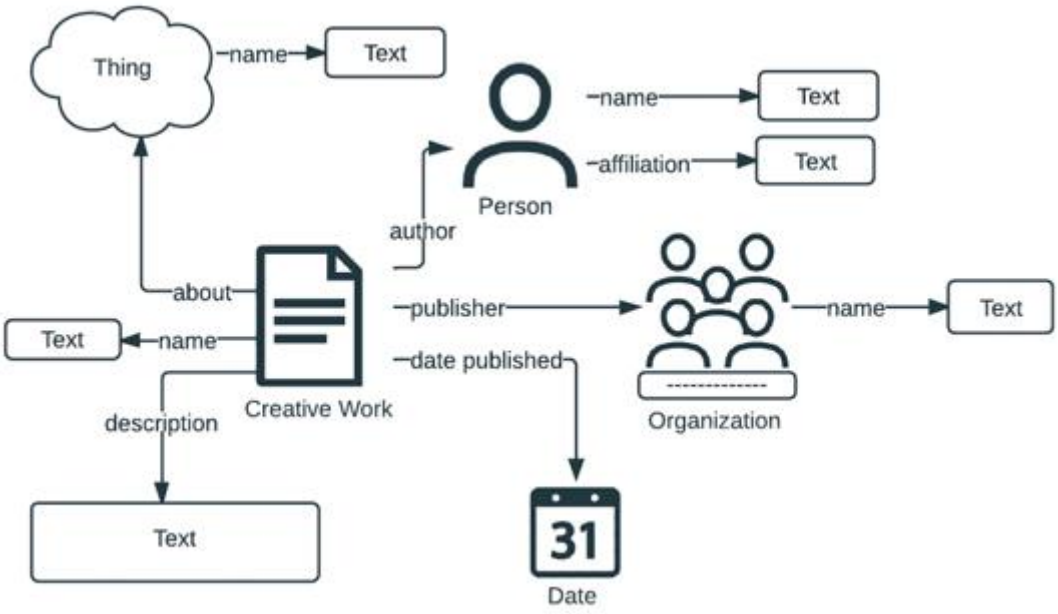

Figure 1: Relationships to depict Creative Work by Schema.org [3]

1.1 Motivation

The data that is embedded into web pages is of great importance in the research fields. For example, the microdata is been used to study how computers have changed the wage structure [3], longitudinal microdata is been used for understanding of productivity [4], microdata is also of great importance in price analytics [5] [6]. In addition, microdata is been used for research studies in the field of healthcare, business, understanding of customers' behaviour and demand etc. Every day the amount of useful data embedded into the web pages increases, which facilitates studies in all field of life.

One of the most common motivation for websites owners and writers to use Microdata in their pages is that it improve the awareness of the pages to potential users by providing structured data to major search engines. The popular search engines extract the embedded data to present richer results with more accuracy and specific details in response to a user search query [7].

Figure 2 shows how google.com use metadata to give very specific and precise information about University of Tartu when a query University of Tartu is searched[1]. Obviously, the user will find this information more useful and easy to understand than a traditional texture description provided under a link by such search engines. Besides it gives a precise short description of the entity (university in this case), its physical and online locations, it is also visually more attractive.

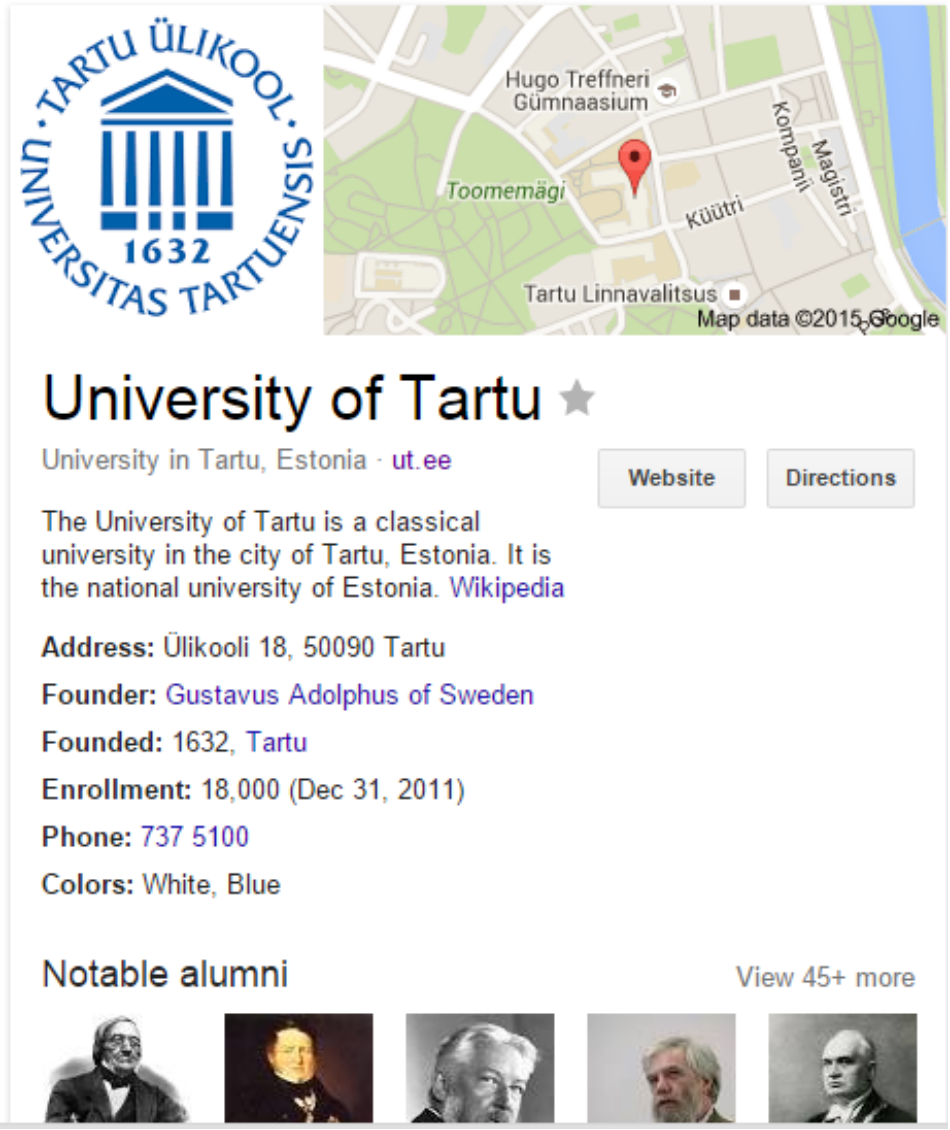

Figure 2: Using of microdata by google in response to query search of "Uninversity of Tartu"[1]

As mentioned in above lines, microdata is not only used by the search engines, it is also used in different research studies, specifically in the field of economics related to price analysis and consumer behaviour. M. Utku Özmen and Orhun Sevinç [5] has used microdata to study the duration of a typical price spell and examine the frequency, size, and distribution of price changes in Turkey. Bernardo Guimaraes, André Mazini, and Diogo de Prince Mendonça [6] has used microdata from different firms to distinguish between time

[1] **Google, "University of Tartu," [Online].** https://www.google.ee/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=University+of+Tartu.

dependent and state-dependent pricing. They have used Brazilian data to distinguish between the two types on the basis whether the time for price changes is affected by realized or expected inflation. There are other numerous papers which study prices and inflation in different countries based on evidence from microdata. Although, most of the papers have used other databases as the sources of data for their studies, yet web is one source of collecting the required microdata, and some have collected data from websites. We describe the different types of data sources for such studies in chapter 2 when we discuss the related work.

For many years, search engines were the only source knowing and giving insight into the published metadata on the web [8]. But as more and more websites started embedding metadata into web pages and the concept got popular, advancements are made in the extraction of the metadata from web pages. Open source tools are been developed, such as Common Crawl [10], Apache Any23 [11] etc. These tools mainly aim to extract the metadata in microformats, RDFa, and microdata from the web, and provide it for further studies. Common Crawl, one of the famous data extraction community, is a non-profit organization that extract metadata and publishes the extracted metadata publicly. Currently, Web Crawl has petabytes of data in the form of raw web page data, metadata extracts and text extracts collected and published over previous years [9] [10]. Any23 provides an API for developers to use it in software applications for extraction of microdata. Similarly, it provides the facility to be used as web service and or command line tool to extract metadata in different microformats embedded in web pages [8] [11].

1.2 Scope

The standards for embedding metadata in web pages is been developed with specifications defined and well-established vocabulary published by different communities. Specifically with the initiative of Schema.org microdata, which is backed by the giant search engines, the vocabulary for embedding metadata is made more precise and accurate. These specifications make it possible and very easy to embed data into HTML pages. In many different domains, the huge number of websites are adopting these standards to embed microdata into their pages. Similarly, extraction tools are been developed for common users in the form of command line and web interfaces, and for programmers as open source APIs that can be used in software applications to extract the embedded metadata. In this paradigm, the only question which is not completely answered yet is the study of duplicates detections in the extracted data. Data that is extracted and is supposed to be used for further process needs to be cleaned from duplicates.

In this research thesis, we will focus on this gap that is not currently filled by open source community. We will develop a linear mechanism to deduplicate in the extracted data. To cope with the issue of performance Apache Spark will be used for implementation of the mechanism.

1.3 Research Problem

As mentioned and described in above two sections, the main aim of this research is to provide a mechanism for deduplication of extracted microdata from the web.

In this research thesis, we answer mainly the following research questions;

- **Generic research question**: Which deduplication method to use for cleaning microdata at Web scale?
- **Specific research question:** How to use Apache Spark for deduplication and cleaning of microdata at Web scale?

1.4 Contribution

Our work mainly focuses on deduplication of microdata. Specifically, this thesis project contributes to the domain with the following objectives;

1) Develop a method and a Spark workflow for effective microdata deduplication.
2) Validate the method and the overall solution on a case of descriptive product pricing analytics with data from .ee domain.

1.5 Structure Description

In Chapter 2, we cover the related work done in the field of data deduplication. We also look into existing state of the art tools that is currently available for extraction and process of microdata. In chapter 3, we will have a look at the background and the already available work in deduplication of data. Chapter 4 explains the mechanism that we propose for filtering microdata from duplications. Also we will present the details of the implementation of mechanism developed for deduplication in extracted data. In chapter 5 we are looking into the experiment that is done to validate our developed mechanism and tool. In subsections of chapter 5 we are discussing the data, settings for experiment, evaluation and threats. Chapter 6 gives a conclusion to the paper and we discuss future potential directions for research in this domain.

2 Related Work

2.1 Data Deduplication

Data deduplication has mainly two themes, one is storage based data deduplication also called data compression or single-instance storage and second is record linkage also known as entity resolution or object matching.
Storage-based data deduplication is a specialized method of eliminating the repeating, i.e. duplicates, copies of the data [9]. This could be achieved by comparing and eliminating the identical copies of data files or the repeating data block (chunks) within the same data file [10] [11]. This technique is useful for improvement of data storage management. With storing only unique data, the deduplication decreases the required storage capacity. Hence, it decreases the size of data center [11]. A lot of research is been done in this field and many different methods are been proposed for storage-based deduplication. One of the common approach for deduplication in storage servers is that a cryptographic hash of the data, that is aimed to be stored, is created. Hash represents an arbitrary length of text in fixed length. Because the length of hash is considerably smaller to the size of the text it is representing, hash decreases the complexity of data chunk comparison. So for each stored or incoming data chunk first a hash signature is calculated, and this hash signature is searched in the already maintained hash index. If an entry for this signature is found then the data related to the hash is not stored, but a reference to it is created, which points to the location of the identical data block on the storage device. But if no entry in the hash index for this hash signature is found then the data chunk is stored on the storage device and an entry of its hash signature in the hash index is created [12] [11].

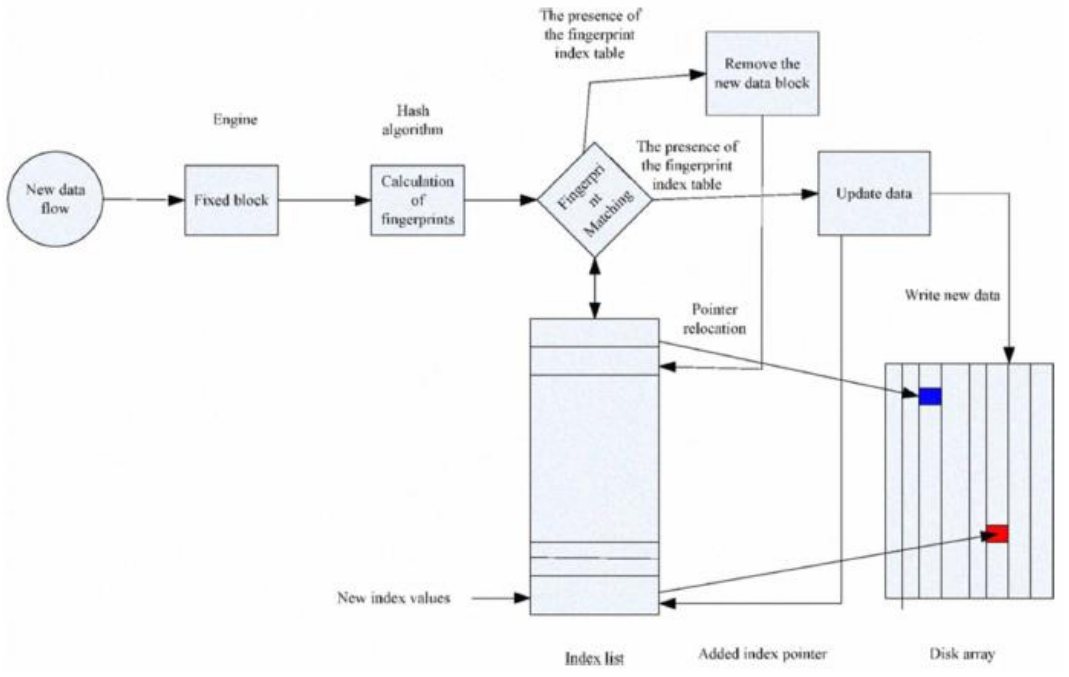

Figure 3: Logical Structure of data deduplication [14]

The second theme of data deduplication is known by many different names, such as record linkage, entity resolution (ER), or record/object matching. It is a deduplication technique, which deals with the identification of entities that refers to the same real-world entity [16] [17] [18] [19]. It is a problem of critical importance for integration of data from various data sources for data quality. The problem of duplicate entities can be found in a single database,

i.e. customers in an enterprise database, but mostly the duplication of data occurs when data from different external data sources are integrated into a single dataset for analysis or research study. As these data sources are independent, they may overlap information and follow different independent and practically incompatible standards [19]. The preciseness of the results based on the analysis of such integrated data set is critical, because important conclusions, sometimes critical business decisions, are based on these results [20]. For example, finding a duplicate customer, location or employee records in an enterprise database, or just like in the case of this research paper collecting data from 100s of different e-commerce applications and matching products for the available offers and price analysis. Generally, entity resolution techniques use multiple similarity measures to compare entities pair for making an effective match decision [13]. Because of the comparison step, where each entity is possibly a candidate pair for comparison with every other entity, the ER process may take hours or in the case of big dataset(s) days [21]. To make the process efficient, a common approach is the reducing of the search space with help of blocking [17] or indexing [18] technique. Through indexing or blocking technique generation of candidate records that are most likely true matches, refer to same real world entity are efficiently generated by use of data structures. For example, the standard blocking technique uses a blocking key value (BKV), generated from one or several entity attribute(s), to input the entities into many partitions, known as blocks, based on the similarity of BKV. Hence, the matching of subsequent entities are restricted to the same block [18]. Still ER remains a costly process and there is much work done to make it more efficient.
This paper mainly focuses on the entity resolution theme of data deduplication.

2.2 Deduplication with Hadoop

Apache Hadoop is an open source distributed computing framework. It provides simple programming model to process large data set across clusters of computers. Hadoop provides a specialized implementation of map reduce, called Apache Hadoop MapReduce. MapReduce is a parallel programming model, which has the map() and reduce() functions to process large sets of data. The map function processes key/value pair and generates an intermediate key/value pair. The reduce function combines all intermediate values which are associated to the same intermediate key.

Deduplication with Hadoop (Dedoop) [17] is a framework based on Apache Hadoop and MapReduce (MR) framework. It facilitates a parallel pair-wise similarity computation of records. Dedoop provides a web-based interface, where the user can specify the MR workflows and algorithms for blocking and classification of records pair from the data set. Then using cloud infrastructure Dedoop provides the results in a visualized form. Dedoop expects clean and structured input from users, other steps for deduplication of integrated data set is same as discussed in section 2.2. Dedoop performs three steps with three MR jobs. Dedoop in the first step applies blocking technique to insert more likely matched records into same block or partition. It generates the candidate record pairs for comparison using the blocking technique and calculates the similarity of the records pairs. In the final step, it classifies the records pairs into matched and non-matched pairs. For classification of record pairs it uses machine learning and includes the step of training Dedoop with label data. Figure 4 shows the depicts how Dedoop works.

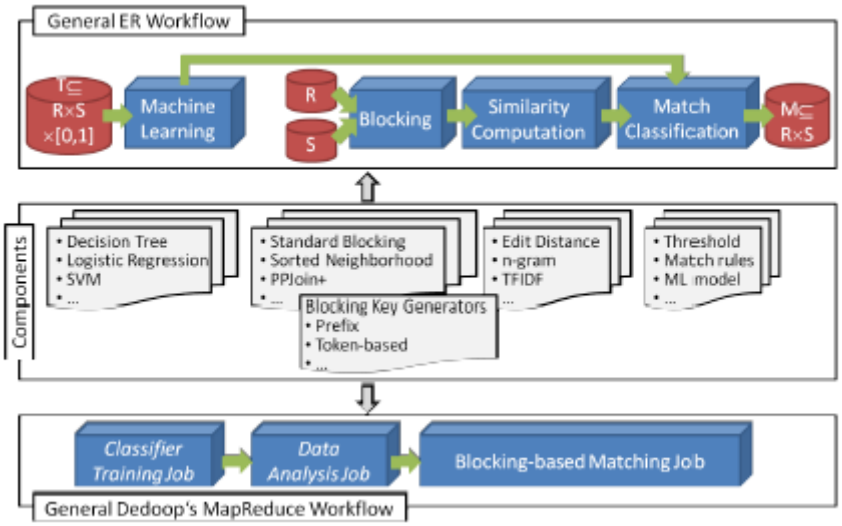

Figure 4: An overview of Dedoop Framework.

2.3 Current State-of-the-art Standards and Tools

Microdata and Schema.org

Out of the three most common used data embedding vocabularies on the web, the most recent is Microdata, the other two are RDF and microformats. First-time microdata shown up in 2009, it started getting popularity with the introduction of schema.org in 2011.

Microdata is a specification that is used to embed machine-readable data in web/HTML pages [13]. Microdata consists of name-value pairs, which describes a resources on the web. The resources in microdata terms are known as *items*. Each *item* is defined according to the microdata vocabulary. The microdata vocabulary is provided by WHATWG, whereas extended vocabulary, to broaden the scope of microdata, is provided by Schema.org [13] [14].

The vocabulary defined by microdata, includes *itemscope* describes an *item* and *itemprop* attribute defines the property or attribute of an item [14]. To specify a specific item or entity, *itemtype* attribute is used. *Itemtype* is the key for describing the type of the item, the different type of items could be a product, book, article, research work, person etc. Micordata does not provide vocabularies to specifically described all these different types items. Rather it gives a general idea of the item, which has a type, scope and some properties.

To cope with the limitations of defining anything in terms of a specific object or an entity with specific attributes, Schema.org is initiated by popular web search engines, i.e. Google, Yahoo, Microsoft Bing and Yandex [3]. It was launched in 2011 to help search engines understand and interpret the embedded structured data in web pages. The search engines needed a wide variety of vocabulary to enrich their search results, for example identifying of a book or an article, its name, author, a link to the publisher, etc.

Schema.org vocabulary provides the URL *http://schema.org/Person* to identify an entity of real-world person. To embed a person data into the HTML contents one must combine the Schema.org vocabulary with microdata or other structured data vocabularies, i.e. RDFs and microformats. In the case of a person, the parent element in HTML should have *itesmscope* and *itemtype* vocabularies from microdata, whereas the *itmetype* will have value *http://schema.org/Person* from Schema.org to embed a person entity into the HTML pages.

The child elements must have all the properties related to the person. The following example shows this concept [13]. In the first snippet the person is described in traditional web and in second snippet the same person is described in semantic web using microdata and Schema.org vocabularies;

```
<!-- Traditional Web -->
<section>
    Hello, my name is Khalil, I am MSc Student at the University of Tartu. I live in Tartu, Estonia.
</section> [13]
```

```
<!-- Semantic Web -->
<section itemscope itemtype="http://schema.org/Person">
    Hello, my name is
    <span itemprop="name">Khalil</span>,
    I am
    <span itemprop="jobTitle">MSc Student</span>
    at the
    <span itemprop="affiliation">University of Tartu</span>.
    <section itemprop="address" itemscope
    itemtype="http://schema.org/PostalAddress">
    I live at
    <span itemprop="addressLocality">Tartu</span>,
    <span itemprop="addressRegion">Estonia</span>.
</section> [13]
```

Please note that the second part shows person entity with *http://schema.org/Person*, it also deals with address as a separate entity and uses *http://schema.org/PostalAddress*, to identify it.

The Schema.org standards provide a hierarchical set of types and their properties. On the top, the most generic type is *Thing*, which could be viewed as *Object* and *Entity* in programming and databases domains respectively. Subtypes of *Thing* include *CreativeWork*, *Event*, *Place*, *Person*, *Organization* and *Product*. Schema.org also provide few subtypes which are specific to an individual domain. For an educational domain, it provides *ScholaryArticle*, *Book*, *Review* and *WebPage* to describe the published research paper or work. To elaborate educational organization, it has *EducationEvent* and *EducationalOrganization* subtypes. In the hierarchy, each type has a set of properties plus the properties that are inherited from its parent type.

The properties, i.e. attributes, of Thing, which is inherited to each of its subtypes in the hierarchy; [14]

- description (Text): item's short description
- image (URL): URL of the item's image
- name (Text): item's name
- url (URL): URL of the item

These properties are quite useful, specifically in differentiating between different things and objects. In one way these properties make it much easier to extract the entities from structured data, combine all into a dataset and match them for duplicate detection.

Schema.org vocabulary is continuously evolving, from its introduction in 2011 till 2015, it has gone through more than 25 revisions. These revisions range from the typo in schema elements to introduction of a fully new vocabulary set for a specific domain, e.g. the *Music Ontology* to vocabulary for the convention of *GoodRelations.* Besides an addition of new vocabulary, the usage of existing elements are changed for better use. Similarly, the elements that shouldn't be used anymore are specified and marked as deprecated. Although the participating stakeholders in discussions are considerably small to its users, schema definitions are maintained in a community-driven process. The prospective changes made to the Schema is announced and discussed through public mailing lists.

Data-Vocabulary.org

Data-Vocabulary.org is the predecessor of Schema.org. Its vocabulary is very similar to that Schema.org. It is been deprecated since the introduction of Schema.org in 2011. It's mentioning was important since some web pages still have metadata structured using the vocabulary of Data-Vocabulary.org.a

Apache Any23

Any23 is the abbreviation for "Anything to Triples". It is a command line tool, an API and a web service [15]. It extracts web data from a variety of micro formats. At the movement, it supports the following formats;

- Microdata and Schema.org
- RDF/XML, Notation 3, Turtle
- RDFa
- CSV with comma separated values
- Micro-formats with many different standards, like Adr, Geo, hCalendar, XFN etc.

Any23 is written in Java, and it is used in the major web of data applications, i.e. sindice.com and sig.ma. Services of Any23 can be utilized in many different ways;

- In java applications as a library to consume metadata from the web.
- It can be used as command line tool for extracting and converting the data to different supported formats.
- It is also available as online service API at any23.org.

Common Crawl

Common Crawl is a non-profit organization, which extract metadata using its crawler software and publish this data for further processing on the internet. The published data can

be used by any person or entity to carry out their own research or processing. The common crawl has published, till now, petabytes of data[2] [8].

Apache Spark

Apache Spark is a fast and general cluster computing framework for the processing of Big Data [16]. It was initially developed in AMP Lab at University of California, Berkeley. Currently, it is maintained by Apache Software Foundation. Spark provides in-memory data processing, in contrast to Hadoop disk-based processing. The in-memory processing makes Spark much faster for Big Data tasks with some limitations, i.e. data should not increase from available memory.

As we will be working on large web-scale data, to avoid performance bottleneck we use Apache Spark for deduplication of microdata.

2.4 Price Analytics using microdata

To study price rigidity in Turkey, M. Utku Özmen and Orhun Sevinç uses microdata [5]. They have investigated micro price data of around 6000 items over four years. The analysis focuses on the duration of price spell and the frequency, size, distribution and synchronization of price changes.

Generally, three types of sources are used for data in the studies of micro-level price spells. The data for item-level prices from the compiled Consumer Price Index (CPI). This type of data, which is available mostly once a month, is generally official and compiled by the statistical agencies. CPI data also deals with representative goods by construction, i.e. not all the price for the specific brands of product, e.g. milk, available in a store is collected. Except the prices for brands which are more common brands through a specific area, e.g. municipality, city or country, is collected. For efficiency of analysis, the second type of data source is the scanner data that is used in such studies. Scanner data is the scanner readings or the registry records of a supermarket. This data is normally available on the weekly basis and it includes both quantity and price data. The third type of data source that is used in micro-level price spell studies is the scrapped data. This type of data is been collected from the online sources by crawling the websites. Normally identifying a unique product and recording price and other characteristics of the product. The major benefit of such data is that it is available in real time, but it covers a relatively low portion of the CPI.

The data M. Utku Özmen and Orhun Sevinç has used for their study is mostly consist of scraped data, the data that is been collected from online resources. The scraped data, in other words, is the microdata that is available on the internet. The data included in this study [5] is been collected from retailers' e-commerce application, i.e. automobile distributors, airline companies etc. They have also used CPI data which was collected manually by visiting different stores in the stores.

Two approaches are used to calculate the price spell. A direct approach, which states that each observed complete and censored spell is recorded as a single duration for an item [5]. An indirect approach which observes the number of price changes for an item over a specific period and the number of time intervals in this specific period in which the prices for the item might have changed. Both of the approaches are interrelated, the lower frequency of price changes means the longer price spell duration, so more rigid prices.

The study [5] conclude that there is a great degree of heterogeneity among the sub-groups of consumer prices. It also suggests that there is a mixed time and state dependent pricing

[2] **Common Crawl** - https://commoncrawl.org/

[5] strategy in Turkey. The mixed strategy of state and time dependent pricing is mostly common in the developed economies but the study shows that it is true for emerging economies too. The study states that the prices in Turkey is more flexible but the synchronization is being low as compared to peer countries of Turkey [5].

2.5 Summary

In this chapter, we discussed different standards, tools, and methods, each has a direct or indirect use in our research project. The data deduplication discussed gives us an understanding of deduplication. Although we will use the entity resolution theme of data deduplication in our project, but discussion on encryption theme of deduplication was important to understand the difference between the two and when each of them is used. Obviously from Schema.org vocabulary we understood that each real world object is structured into web contents as a thing that is an object or an entity. The standards, i.e. microdata and Schema.org, discussed gives us an understanding on how structured data is implanted into web pages. Hence, it helps in developing a valid mechanism to extract the embedded data and transform it to a valid format efficient for data comparison and deduplication. Our focus in this paper is on microdata. Currently, on average 35% of websites use microdata to implant structured data into their pages. Because schema is backed on popular search engines, its popularity increase and it is becoming more prominent over other metadata vocabularies. The tools that are mentioned gives us an understanding of how data is extracted from the web pages. Our focus is on deduplication of microdata, so we will reuse the current extraction tools and methodologies these tools use for extraction of microdata from websites. A suitable option for such extraction is the API provided by Apache Any23 to extract data from web pages programmatically. The discussion on Dedoop is specifically important to understand that the currently existed solutions for deduplication are using the MapReduce approach for deduplication. For our approach, we will use Apache Spark which is much faster than Hadoop. The last section, talks about the possible use of microdata. Although the papers presented in the last section do not completely depends on microdata collected from web, but they do use data collected from websites. Similarly, as this data is freely available on the web, it could be of great use in such studies in coming years. It need a process which could be used from collection microdata over web to its cleaning and comparison for different research study. And this is the purpose we will work in this paper to achieve.

3 Background

In this chapter, we will give a general idea of data matching and deduplication technique. We will discuss some of the important concepts and methods that are widely used for deduplication of data.

3.1 Deduplication process

Duplicate detection is an essential task for data integration and data cleaning. Because of its importance and time-consuming process, a large amount of research is been conducted on this topic covering different aspects of the subject, and a number of solution have been proposed [17]. These proposed solutions cover both sequential and parallel approaches. In this subsection, we will discuss the general deduplication process.

Peter Christen [18], has discussed in details the deduplication and data matching process. He proposes mainly 5 steps for deduplication of data, which are;

1. Preprocessing
2. Indexing
3. Record pair comparison
4. Record pair classification
5. Results evaluation

Peter Christin's [18] duplicate detection states data preprocessing as the first step. The standardization of attributes to the same structure and its value into a standard format is dealt with in the preprocessing step. When data is cleaned and standardize, it is ready for duplicate detection through record pair comparisons. To find duplicates each record in the dataset should be compared to each other record for calculating the similarity of the records pair. This makes the similarity calculation is quadratic to the size of the dataset(s). Most of these comparisons are among the records which are not similar because all records are not similar to each other. Indexing, also known as blocking, technique is used to reduce the record pairs which need to be compared. For detail comparison, indexing technique generates record pairs that are most likely similar to each other. After indexing of the records, the record pairs need to be compared to calculate the similarity and classify them as duplicate or nonduplicate pairs. The final step is evaluating the quality of classification, where the classified pairs are evaluated to identify how many of them correspond to real-world entities. Similarly, matching completeness is checked by looking at how many real-world entities that appeared in the dataset are correctly matched. A common accuracy measure for this step is precision and recall.

Although, Peter Christin [18] explains these steps systematically and in details, this approach seems to be common among other work done in the entity resolution domain with some variations. The papers [19] [20] uses nearly the same kind of approach. A very famous paper [21], which is a survey about deduplication techniques, dicusseses the main process in 3 main steps and indexing technique as an additional step. It refer to the preprocessing step as data preparing, that is cleaning the data from unwanted characters, its transformation, and standardization to a uniform format. The second step then includes matching of the entities based on its attribute values, and based on the similarity of the attribute values, the decision of the entity pair as duplicate or non-duplicate. Finally, the paper discusses the blocking techniques for making the process faster and efficient. It skips the evaluation of the process.

3.2 Pre-processing

Pre-processing also known as the data preparation [21] is a crucial step for successful data deduplication [18]. It is the initial step taken in the deduplication process. Data commonly have attributes, such as name, description, price, provider etc. in the case of a product. It is important to make sure that the data, which comes from different data sources, is properly standardized. It is also important to confirm that the attributes which are used for comparison of the data records are in one standard format. Data entry errors and the structure in which data is presented in the dataset(s) are the two main factors, among other, which affects data quality [18].

The main steps that are involved in data pre-processing are the removal of unwanted characters and words, expanding of the abbreviations, correcting misspellings and dividing attribute values into well-defined text segments [18]. Normally the data has unwanted characters or words, such as meta-information which explains the formatting of the text to a web browser, information that explains the language used for the specific attribute, or some unwanted spaces, special characters etc. These unwanted words and characters need to be removed and the actual value of each attribute needs to be cleared. The attribute values should also be checked for misspelled text.

3.3 Indexing

The most expensive step in data deduplication is the comparison of records. Comparison of each record to every other record in the dataset(s) is resource consuming. In such case, most of the comparisons correspond to not true matches, that means not each record is similar to every other record in most cases. To compare two data sets of m and n size for duplicate detection, the total record pairs for comparison will be m x n pairs. To reduce the number of record pairs for comparison, records indexing is an important step in data deduplication. The aim of indexing is to remove pairs that are unlikely true matches. Whereas the pairs that are possibly true matches (two records which refer to the same entity) need to be kept for detail comparison. Therefore, indexing is a filtering step, which is based on a type of index data that is a base for bringing 'similar' valued records from the different data sets or within the same data set together. In this step, all the records of each dataset are processed and then each record is inserted into one or several blocks for comparison. The index data should be formalized in a way that the possible similar records are inserted in the same block. For example, one case is sorting of the records to move the possible similar records close to each other. The index data that is used for indexing is called blocking key, also known as sorting key in case of using sorting approach for indexing. The blocking key, in general, is generated from a single or multiple attributes in the record. For example, a name, and price attributes could be a good option for developing a blocking key for indexing products in order to bring the most similar products into the same index list. In the same way, if the records are been sorted based on this blocking, here better to say sorting, key in sorting-based indexing technique, it will lead to records with same name and price next to each other. Defining the indexing key is a critical part of each indexing technique.

Following are some techniques used for indexing or blocking;

1. **Standard Blocking/Indexing**: It is a traditional approach for data matching and deduplication process for many years. One blocking key value (BKV) is generated for each record in the dataset(s). Based on the BKV the record is inserted into a block, with the same BKV into the same block. To match two data sets all possible pairs of candidate records for comparison are formed from all the records which have same BKV across both data sets. In case there is a BKV only for records of one data

set then no pairs are made, as the other data set do not have any similar BKV, and no comparison is done for such BKV.

2. **Sorted Neighbor Approach**: It is an alternative to the standard blocking. Instead of generating blocks based on BKV, it uses 'sorting key'. The sorting key is generated in the same way as BKV, to sort the records in the dataset. After sorting the records, a fixed length of sliding window (w>1) is then moved over the sorted records, all possible records pairs in the window at any given step are candidate pairs for comparison. Following pictures, from a lecture on sorted neighbor techniques from Felix Nauman [22], shows an example of sorted neighbor approach. The rectangle in red color depicts the sliding window concept.

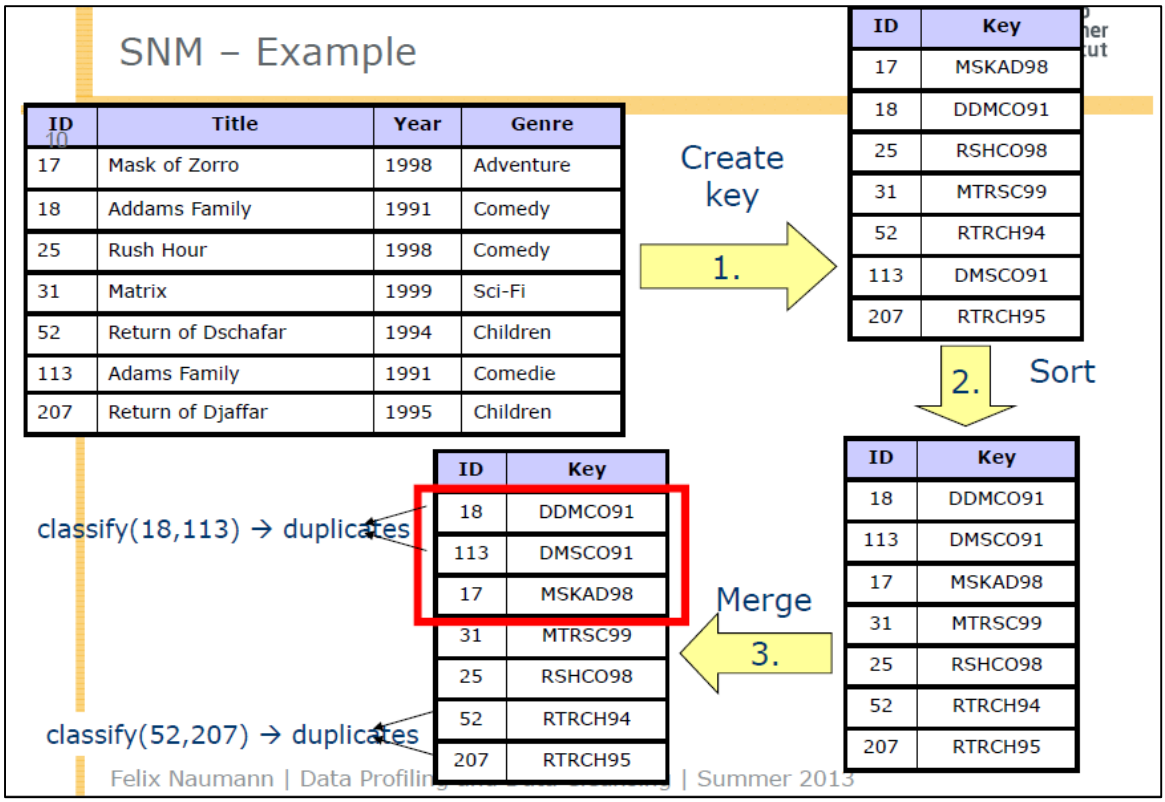

Figure 5: The concept of sorting key generation and data sorting [22].

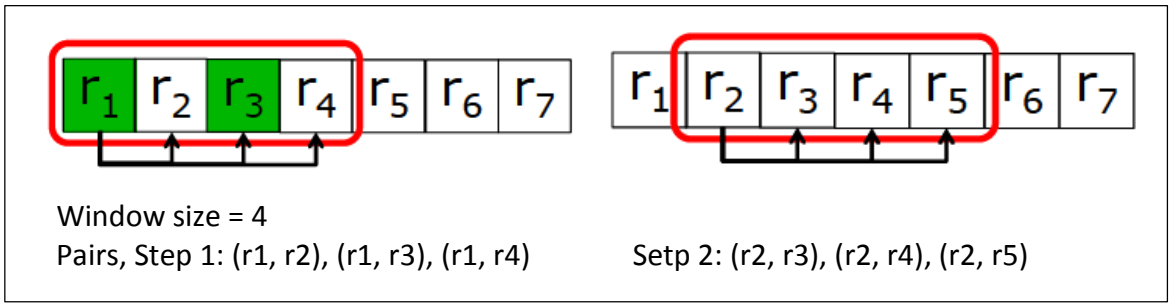

Figure 6: The concept of sliding window over the records, two steps are shown here [22].

The window size and how the windows should slide has many variations based on the data and number of pairs that needed to be compared. The bigger the window size, the larger the number of candidate pairs for comparison and more computational resources is needed. Bigger window size also makes the comparison more accurate, as more records are compared to each other and there are limited chances to miss any candidate pair which may

be a true matching pair. The maximum widow-size could be n (the number of records in the dataset(s), in such case, each record will be compared to each other record, which is not at all aim of indexing technique. The minimum window size can be 2, that is each record is compared to the record next to it only.

Other well-known algorithms for indexing are Canopy Clustering, Suffix-Array Based Indexing, Q-Gram Based Indexing, and Mapping Based Indexing, to name some.

The results provided by Peter Christen [18] states that the slowest technique is mapping based indexing, followed by q-gram indexing, and canopy clustering. The large number of candidate record pairs are produced in mapping based indexing. Sorted neighborhood and suffix-array indexing are faster and require less memory. These two approaches are simpler too.

3.4 Resource Description Framework:

RDF, Resource Description Framework, is a World Wide Web Consortium (W3C) framework which enables encoding of metadata in the form efficient for exchange and reuse. RDF imposes XML structured constraints on its document creation to provide an unambiguous method of semantics expression. The constraints also ensure encoding and exchange of standardized metadata. RDF provides human and machine-readable vocabularies to facilitate and encourage the reuse and extension of metadata semantic among the information communities [23].

RDF framework provides a model for describing resources which is identifiable by Universal Resource Identifier (URI). A resource has attributes and characteristics [23]. The properties, i.e. the attribute associated with a resource are marked with property-type, each property-type has a value. The collection of the properties corresponding to a resource is called description of the resource.

N-Triples is a specific plain text syntax for RDF. It is an easy to parsed line-based syntax [24]. N-Triples consist of a sequence of RDF terms, representing subject, predicate and object of RDF simple triples. The triples may be separated by spaces or tabs. Each sequence is presented in one line terminated by '.' (dot) and new line character [24]. Following is an example of N-Triples;

_:subject1 <http://an.example/predicate1> "object1" . [24]

_:subject2 <http://an.example/predicate2> "object2" . [24]

When a label of topic is added to N-Triples it becomes N-Quads. Following is an example of N-Quads;

_:subject1 <http://an.example/predicate1> "object1" <http://example.org/graph1> . [25]

_:subject2 <http://an.example/predicate2> "object2" <http://example.org/graph5> . [25]

RDF is an important concept in Linked Data. As discussed above, RDF facilitate encoding of structured data about an entity or object, i.e. resource. Each resource is identified by an identifier. Normally, in the real world each resource has relationship to any other resource. RDF also facilitate this recording of this relationship. N-Triples and N-Quads are very important simple syntax to record these RDF concepts or linked data in plain text. For example two famous fiction characters are Spiderman and Green-Goblin, to present the linked data of these two characters following N-Triple statement could be used;

<http://example.org/#spiderman>
<http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin> . [24]

Please note that there is no line break between the three triples. In very similar case to above example any resource could be presented in terms of N-Triples. To the above example if we add a label or topic say <http://example.org/graphs/spiderman> [25] which may shows the source of the is information the triples will become quads, hence from N-Triples to N-Quad.

We are using N-Triples and N-Quads to store the extracted microdata. These formats are quite efficient for processing, storing and transferring of the data.

4 Design and Implementation of Deduplication and Anomaly Removal Methods

In this section, we will give an overview of the complete microdata deduplication process. It starts at extraction of data from HTML contents and ends with obtaining a unique result set of the records. In the following chapter, we evaluate this method with real world data to present the evidence of its efficiency and accuracy.

4.1 Deduplication Technique

The method that we have used for microdata deduplication is very similar in many steps to the methods used in related research papers [21] [19] [18]. Following are the step we see necessary for microdata deduplication;

1. Data preparation
2. Indexing or blocking
3. Record comparison and duplicate elimination
4. Results evaluation

Data Preparation

The Data preparation step is also commonly known as data pre-processing, see section 3.2 for details. We agree with the authors of *Duplicate Record Detection* [21] on saying this step as data preparation, because in the case of microdata this step is more like an initial process followed by the deduplication process. Deduplication, as mentioned in the introduction, is the process of finding all those entities which refer to the same real-world entity. Before the start of the deduplication process, the data goes through some standardizing steps during which the data is converted into a uniform format, reducing the structural heterogeneity [21]. The sub-steps data preparation requires are data transformation and data standardization steps.

By microdata, we mean, specific to this paper, the structured data embedded into web pages. In such case, it is less obvious that the data under study comes in a format suitable for starting the process for deduplication. Mostly the data is in the form of web pages or HTML contents. Following are steps required for transformation and standardization of microdata into a form suitable for input into duplicate detection process.

1. **Extraction of Microdata**: For microdata, the first required step is extraction of microdata from HTML contents to a format supported by the extraction tools, such as Apache Any23. Most common formats are N-Triples, N-Quads, Turtle, and JSON etc. This step could only be skipped if the data is already extracted and available. In my case, we extracted the data into RDF N-Quads format.
2. **Conversion of the data to entities**: The second step that we preferred in data preparation is conversion of data from N-Quads or other formats supported by the extraction tools to tuples. In tuples each record is presented in a single row and the attributes of each record is separated by a comma. One good reason for converting the extracted data to tuples is that microdata and schema.org embed the structured data into web pages in a very similar way to an object or entity data. That is each resource on the web has a type and characteristics, i.e. attributes. So it is quite easy to identify the microdata resource as an object, e.g. a person or a product and its attributes. Once the data is in tuples, each tuple could be used as a separate entity. In my case, we present each record in a single line, where each property of the record is within the brackets (e.g. <property>) and the properties are separated by semicolon. Semicolon separation is good when it comes to the properties that have

commas in the text. All attributes of the attributes of each record is presented in one standard format, i.e. String/Text.

3. **Removing unwanted characters**: The data comes from different web pages, hence with much different quality and mistakes. Many pages have formatting data mixed with the attribute values' text. The most common unwanted characters we had with the data was space formatting characters, e.g. \n, \t etc., and metadata, e.g. @en for presenting language of the text etc. The only way to get rid of such character was to remove it from the text.
4. **Encoding correction**: As the data comes in the different language, specifically the data that we worked on was from .ee domain. It had many characters in the form of Unicode, e.g \\u00E4 for Estonian ä character etc. Such type of character needs to be corrected and converted to the standard character or alphabet. We used the java library to convert such the Unicode back to UTF-8 characters.
   The removal and correction of text should be performed after cleaning and transforming the data into a standard form, in my case we performed this step after formatting the data into tuples. We performed all the cleaning steps sometime over the whole record as a single string and in other cases for each individual attribute in the record.

Indexing

After the step of data preparation and preprocessing the data is transformed to a form appropriate for comparison of the records pairs in order to identify duplicate records. But, as discussed in section 3.3, the comparison of records is quadratic to the size of the dataset(s) in nature, when each record is compared to each other record in the dataset(s). To avoid the waste of resources and make the process efficient, the records are divided into blocks in a way that the records which are more likely duplicates are inserted into similar blocks. Hence, the comparison of records which resides in the same block is carried out.

One famous approach for indexing is Sorted Neighbourhood technique, we have chosen this technique for deduplication of microdata. We put efficiency, simplicity and compatibility to Apache Spark for selecting an indexing technique. An evidence for Sorted neighbourhood technique could be found in Peter Christen work [18]. He states that among the indexing techniques, the Sorting Neighbourhood and Suffix Array indexing techniques are the faster techniques and require less memory. Sorted neighbourhood technique is much simpler too. It needs the generation of sorting key and based on the sorting key all records are sorted to bring the most similar records near to each other. A sliding window is used to compare the records resides next to each other. This technique is more compatible to Apache Spark in the sense that it uses less memory, Apache Spark also relies on memory for data processing. Hence, it is much safer to use Sorted Neighbourhood technique on Apache Spark for the large dataset containing millions of records. One other reason, that is much specific to choosing Sorted Neighbourhood over Suffix array indexing, is that Apache Spark's MLlib library implements the sliding window concept. It makes selecting the records pair for comparison much simpler.

The sorting or indexing key is a crucial part of any indexing technique. It is not important which indexing technique is used for indexing of data, but important is the definition of the indexing key [26]. Because this is the indexing key which brings similar valued records to the same block. The sorting key could be developed of a single attribute or multiple attribute value. When defining a sorting key, it is important to take care of the data that needs to be de-duplicated and the indexing technique that is in use.

Keeping in view the guidelines mentioned above we prefer name, price, and provider as the necessary attributes which could be used for defining the sorting key. Name is always an important aspect of a record. Whether it is a person, place or product etc., name plays the important role in identifying an entity. Similarly, if we are talking about the product the price and provider of the product are much important too. Because these two attributes play the important role in differentiating the product entities from one another. For example, if we say Galaxy SI, it is enough for identifying a product, but not for differentiating with other products that may have the same name. If we add provider or manufacturer to the product name, e.g. Samsung, then the product is clearer. And with adding price information, we come closest to identify and differentiate it with other products. The more attributes added the more will the sorting will be perfect. But not in all cases, because while choosing the attributes for a sorting key definition, we should also keep in mind that the attributes should have the maximum or complete frequency and good quality. By frequency, we mean the attribute should have values for all or maximum records. Microdata is not collected from a single website, rather it comes from multiple websites. Each website provides only a limited set of data about any product record. While studying the data we came to know that these three attributes are mostly complete attributes as for as the data that we have collected from .ee domain. We also understand that add all attributes to defining the sorting key will not benefit, while in some cases it may affect the indexing in the negative way. As we select limited attributes, similarly we do take part of the attribute text, e.g. 10 or 20 characters. The selection of characters to add to sorting key also depends on the length of attribute value through all the records. So my way of defining sorting key is combining first 30 characters of the name with the price of the product and the domain name of the product provider. Choosing of only 30 characters from the name of the product is based on experiments, detail of which is given in chapter 5.

```
String key= EMPTY_STRING;
if(name.length()<30)
     key+=name;
else
     key+=name.substring(0,30);
```

Record comparison and duplicate elimination

After sorting the records based on the sorting key, same valued records are moved closer to each other and are put in the same partition. Arranging of the records in partitions is handled by Apache Spark. Now when the records are arranged, they are ready for comparison. A sliding window will slide over the records and will pick the nearest neighbours for comparison. We keep the sliding window size 3, that means at each iteration three records next to each other are picked, making two pairs of records for detail comparison. Working of sliding window is shown in Figure 6 for a window size of 4, with details in section 3.3.

After picking the records for comparison, the first record is paired with second and with third separately, making two pairs of record in case of sliding window size 3. We compare the two pairs and calculate the similarity between the pair. The result of the similarity calculation is either duplicate (1) for or no duplicate (0) records. We add all unique records to the resultant set and for all sets of duplicate records, we add only one representing records to the resultant record list. For example, if we have P1, P2, P3, P4, P5 and P6, a sliding window of size 3 will have 5 iterations and 10 record pairs for comparison as follow;

Data Set (Products)	P1	P2	P3	P4	P5	P6
Window Size	3					

Table 1: A data set of 6 products and the window size 3

Iterations	Pairs	Comparison	Comparison result	Resultant Data Set
Iteration 1	P1, P2	Yes	Not Duplicates	P1
	P1, P3	Yes		
Iteration 2	P2, P3	Yes	Duplicates	P2
	P2, P4	Yes	Duplicates	
Iteration 3	P3, P4	No		
	P3, P5	Yes	No Duplicates	
Iteration 4	P4, P5	Yes	No Duplicates	
	P4, P6	Yes	No Duplicates	
Iteration 5	P5, P6	Yes	No Duplicates	P5, P6

Table 2: Pairs and Iteration of dataset with 6 products and window size 3

The table shows a window size of 3 will have 5 iterations, by principle it should have only 4 iterations but the last one is extra for comparing the 'window size' -1 remaining products. Similarly, the actual number of comparison are shown in the table which were the only 8, details for the process are given bellow. The last column shows the resultant data set which contains a representing product for the duplicates and the unique product which has no duplicates.

Record Pairs List		
P1		
P2	P3	P4
P5		
P6		

Table 3: List of duplicate record pairs

This table represents a list which is updated on each iteration, recording the duplicate pairs. In case of products having no duplicates, the list will have only unique product.

Shown in the above three tables is the process of comparison of record pairs. We see necessary clarifying of few points. The last table (Table 3), this table shows a list of a list which keeps records of a product or set of products. In case a product has no duplicates, e.g. P1, P5, and P6, it comes alone in the first column of this list. In case there are multiple records which correspond to a similar product entity, e.g. P2, P3 and P4 all represent one product and are duplicates, then the list contains all these products in one row. This list is updated on completion of each iteration, whether a duplicate product is added to the already

recorded product set or a unique product is added to the last row of the list. The product in the first column does not come with a specific formula, rather it is the product that comes first from top to bottom on the original sorted data set. This same product, i.e. P2, represents the other two similar/duplicate products, i.e. P3 and P4, in the resultant data set.

One other benefit of the list of list shown in Table 3 is that on each iteration we first look at the last row of the list and take all the records on the row, e.g. at iteration 3 of the sliding window the list already has P2, P3, and P4 from the previous iteration, we retrieve this list. We then compare the entity at the second last position/index of the row, i.e. P3, with the first entity of the record pair picked by sliding window, i.e. also P3, and the entity at the last index, i.e. P4, with the second entity of the first pair picked by the sliding window, i.e. P4 too. In this case we the pair picked by sliding windows is completely similar to the pair that exists on the last row of the list, so no need for detail comparison. Because we already know from the previous iteration of the sliding window, with the help of this list, that P3 and P4 are pairs. By this way, detail comparison of the entities that are already known duplicates are been avoided. In the case of big data sets, which has a high number of duplicate entities avoiding such comparisons may effects the overall time of the deduplication. This is also the reason for *No* in the first row of the third iteration at the second column of Table 2, titled as *Comparison*, which means no detailed comparison is been done for the pair (P3, P4).

We follow document based approach, the data set for deduplication is read from files on disk, and after the deduplication process, the resultant set is been write back to files. During the process, the data sets and intermediate results are managed by Apache Spark.

Results evaluation

Evaluation of the method is explained in chapter 5.

4.2 Implementation of the deduplication technique

The implementation of the method is been done in three main steps.

1. **Extraction of Microdata:** In the first step, we process HTML files and extract data to N-Triples format. The N-Triples are further formatted to N-Quads. The N-Quads contains a key, subject, predicate and object. The key contains a time stamp on which the data is been collected, the domain name from which the data is collected and the HTML page address which contains the collected data. The key is been combined with the standard subject, predicate, and object that have been extracted from the HTML contents using the Any23 API. The N-Quads are stored into files.

```
extractMicroData(String htmlContents) throws Exception{

      Any23 runner = new Any23("html-microdata");
      File file=createTempFile(htmlContents);
      DocumentSource source= new
             FileDocumentSource(file);
      ByteArrayOutputStream out = new
              ByteArrayOutputStream();
      TripleHandler handler = new NTriplesWriter(out);
      String result = runner.extract(source, handler);
      result = out.toString("UTF-8");

      handler.close();
}
```

The above code snippet received HTML contents, it then writes the contents to a temp file and the extract microdata from HTML contents with the help of Apache Any23 library. The extracted data is further processed to N-Quads and stored in files.

```
setNQuadStatements(String key, String result) {
     String[]
statements=result.split("(\\s\\.)(\\r?\\n)");
     StringBuilder stat=null;
     for(String statement: statements){
     stat=new StringBuilder("");
        System.out.println(statement);
        String[] statParts = statement.split (
                    "\\s(<|\"|_)");

        String subject = statParts[0]
                    .replaceAll("(<|>|\")", "");
        String predicate = statParts[1]
                    .replaceAll("(<|>|\")", "");
        String object = statParts[2]
                   .replaceAll("(<|>|\")", "");

        stat=new StringBuilder("").append("<"+key+">,")
               .append("<"+subject+">, ")
               .append("<"+predicate+">, ")
               .append("<"+object+">");
        statementsList.add(stat.toString());
     }
     writeToFile(statementsList);
}
```

As could be seen in above code that we split the N-Triples with String.split() method and extract the RDF triples. There are some libraries, specifically a method by Any23 which does the separation. The problem with using the available libraries are that the N-Triples statement order is not preserved in the way it is extracted from HTML contents. The order of statements is quite important for extracting entities from N-Quads.

2. **Transformation of N-Quads to Tuples:** After processing the HTML pages and formation of N-Quads, in the next step we extract product entities from the N-Quads.

```
SparkConf sparkConf =
     new SparkConf().setAppName("NTriples To Entity");
JavaSparkContext ctx = new JavaSparkContext(sparkConf);

Configuration conf=
     new Configuration(ctx.hadoopConfiguration());
conf.set("textinputformat.record.delimiter",
          "22-rdf-syntax-ns#type>, ");
JavaPairRDD<LongWritable, Text> lines=
               ctx.newAPIHadoopFile
               (
                    readFilePath,
```

```
                        TextInputFormat.class,
                        LongWritable.class,
                        Text.class,
                        conf
                );
JavaRDD<String> bLines=lines.map(
          new Function<Tuple2<LongWritable, Text>,
     String>(){
     @Override
     public String call(
             Tuple2<LongWritable, Text> line) throws
     Exception {
          return line._2.toString();
          }
      }).filter( new Function<String, Boolean>(){
          public Boolean call(String bLines){
               return (
                    bLines.contains("org/Product")||
                    bLines.contains("org/Offer"));
            }
     });
```

The above code snippet initializes an Apache Spark process which reads N-Quads from files. The files are split into chunks, a group of lines, based on the predicate which contains 22-rdf-syntax-ns#type. The predicate <http://www.w3.org/1999/02/22-rdf-syntax-ns#type > has always a schema.org class as subject, e.g. <http://schema.org/Product>, <http://schema.org/Offer> or <http://schema.org/Organization>. Then filtering all those groups of statements which neither represent product nor and offer, again the product and offer always come after one another. So the filter uses quite efficiently remove the microdata which presents other than product and its offer.

After the splitting and filtering of the data, we read the remaining N-Quad statements line by line and look for the product attributes, i.e. name, SKU, description, product URL, image URL etc. A product object is been initialized and data for each attribute is assigned to the corresponding variable of the product. The list of products received from each file is converted to String and saved in files called Entities. Each entity is a tuple of name, SKU, description, product URL, image URL, price, currency, availability, provider and time stamp.

3. **The de-duplication process**: Having product entities in from of tuples makes it easy to read these entities and process them for duplicate detection and removal. First, an Apache Spark process is initialized and the data is read. While reading the data the entities are checked of completion. An entity is not a complete entity if it has no name or its SKU, description, product and image URL, and currency all together are empty. Following code represents the initial process of reading an cleaning of entities;

```
SparkConf sparkConf =
     new SparkConf().setAppName("Product Unification");
JavaSparkContext ctx =
     new JavaSparkContext(sparkConf);
```

```
JavaRDD<String> lines = ctx.textFile(args[0], 16)
          .filter(new Function<String, Boolean>(){
               @Override
               public Boolean call(
                    String line) throws Exception {

                    if(!line.isEmpty()){
                         return isRecordValid(line);

                    }
                    return false;
               }
});
```

The method isRecordValid() looks for the completion of the product entity with the criteria mentioned in above lines.
After creating the RDD which has all cleaned and valid entities, the next step is creating of the Sorting Key for each entity and appending it to the corresponding entity.

```
JavaPairRDD<String, String> keyEntity =
          lines.mapToPair(
               new PairFunction<String, String,
          String> () {
     public Tuple2<String, String> call(
               String entityTuple){
          String key=generateKey(entityTuple);
          return new Tuple2<String, String>(key,
               entityTuple);
     }
});
```

Whereas method generateKey() has the following logic;

```
String key= EMPTY_STRING;
if(name.length()<30)
     key+=name;
else
     key+=name.substring(0,30);
key+=price;

if(!(provider.isEmpty())){
          key+=provider.split("\\.")[1];
}
```

Using sortByKey() method of Apache Spark we can easily sort the entities. While sorting the entities, Apache Spark will also handle partitioning the entities based on the sorted order. With sorting the entities more similar entities will move near to each other. After sorting the entities with the sorting key, we no more need the key, so we will get the product record only.

```
JavaRDD<String> sortedEntity =
          keyEntity.sortByKey().values();
```

Entities are sorted and placed into partitions, i.e. blocks. So now we will fix a window size and will start picking product record pairs for comparison.

```
final int windowSize=WINDOW_SIZE;
RDD<Object> slidingRDD =
                RDDFunctions.fromRDD(sortedEntity.rdd(),
                sortedEntity.classTag()).sliding(windowS
                ize);
JavaRDD<Object> recordPairsRDD =
     new JavaRDD<>( slidingRDD, r.elementClassTag());
```

In above code snippet, we selected a window size and then create an RDD which will handle the sliding window over the complete data set. The sliding RDD is implemented by Apache Spark's MLlib library.
Now detail comparison of the records pair will be done, as follow;

```
final long size= recordPairsRDD.count();
JavaRDD<String> result =
     recordPairsRDD.map(new Function<Object,String>() {
          @Override
          public String call(Object recordsPairs)
                    throws Exception {
               iteration++;
               String record="";
               if(iteration==size){
                    record=handleLastPairs(
                                   recordsPairs);
               }
               else
                    record= Util.matchPairs(
                         (Object[]) recordsPairs);

               return record;
          }
          Private String handleLastPairs(recordsPairs){
               Object[] lastPairs=
          (Object[])recordsPairs;
               for(int i=0; i<lastPairs.length; i++){
                    Object[] newArray=
                         new Object[lastPairs.length-
                              i];
                    newArray= Arrays.copyOfRange(
                              lastPairs,
                              i,
                              lastPairs.length);
                    record+=Util.matchPairs(newArray);
               }
          }
     }).filter(new Function<String, Boolean>(){
          @Override
               public Boolean call(String arg0) throws
Exception {
```

```
                return !arg0.isEmpty();
            }
        });
```

The size variable gets the number of sliding window iteration. Then we start counting the iterations up to the last iteration. At the last iteration, we take all the remaining records and give the record pairs for comparison through a loop, which decreases the records one by one and sends the records pair for comparison. This way we handle matching the records at index and bellow window size minus 1. As the sliding window is not able to pick these boundary records.
Detail comparison is handled by the Util.matchPairs(Object[]) method in the a custom Util class. The method looks like;

```
public static String matchPairs(Object[] recordsPairs)
{
      String recordToReturn="";
      ArrayList<Long> prePairsIDsList=
                  new ArrayList<Long>();
      if(recordsPairs!=null){
            List<Product> productList=
                        new ArrayList<Product>();
            for(Object recordObject:recordsPairs){
                  String[] record= recordObject.toString()
                        .replaceAll("<|>", "").split(";");
                  Product product=setProduct(record);
                  productList.add(product);
            }
            Product product=new Product();
            int lastRecordIndex=
                  recordsComparisonList.size()-1;

            if(!recordsComparisonList.isEmpty())
                  prePairsIDsList=
                        recordsComparisonList.get(
                              lastRecordIndex);
            if(prePairsIDsList.size()<2){
                  product=productList.get(0);
                  recordToReturn=product.toString();
                  prePairsIDsList.add(product.getId());
            }else{
                  product=productList.get(0);

if(!(prePairsIDsList.contains(product.id))){
                        prePairsIDsList=new ArrayList<>();
                        prePairsIDsList.add(product.id);
                        recordToReturn=product.toString();
                  }
                  else{
                        recordsComparisonList
                              .remove(lastRecordIndex);
                  }
```

```
            }
            for(int i=1; i<productList.size(); i++){
                  Product productOther=productList.get(i);

if(!prePairsIDsList.contains(productOther.id)){
                        if(product.equals(productOther))
                              prePairsIDsList
                                    .add(productOther.getId()
                              );
                        else if(product.compare
                                    (productOther))
                              prePairsIDsList
                                    .add(productOther.getId()
                              );
                  }
            }
            recordsComparisonList.add(prePairsIDsList);
      }
      return recordToReturn;
}
```

The recordsComparisonList is the list of list which keeps track of the records with its duplicate copies. The details about this list is given in section 4.1 and Table 3 corresponds to this list.

5 Validation Experiment

In this chapter, we will discuss the data set for our experiment. The experiment setting and evaluated results will also be included in this chapter, besides the challenges to the validity of the data.

5.1 Data

As mentioned earlier the data for this research work is collected from the .ee domain. The data was already collected by a crawler application and it was stored in sequence files. It was collected from the pages that are publically available as e-commerce stores over the web. Hence for our data set, we use thousands of pages which had embedded structured data. The data set that we use embed all the data directly in the HTML contents using the schema.org and microdata vocabulary. In order to make this data usable for our experiment, we first read the HTML contents from the sequence files and by the use of Any23 API extract metadata embedded in the HTML contents. The extracted data is stored as RDF N-Quads which combines the key for the page with standard RDF triples, i.e. the subject, predicate, and object. The key for the page is made up of the domain name, specific page URL from which the data is collected and the time and date on which the data is stored (into the sequence files).

We were interested in all microformats which are used to embed metadata into HTML pages. But all the websites that were been crawled was using a very similar type of vocabulary for embedding the data into their web pages. The extracted microdata shows the use of mainly two microdata vocabularies, i.e. Schema.org and Data-Vocabulary.org. Both of the two vocabulary sets has very similar vocabulary used together with microdata and other microformats to embed structured data into HTML contents. Figure given bellow shows the top ten vocabulary classes;

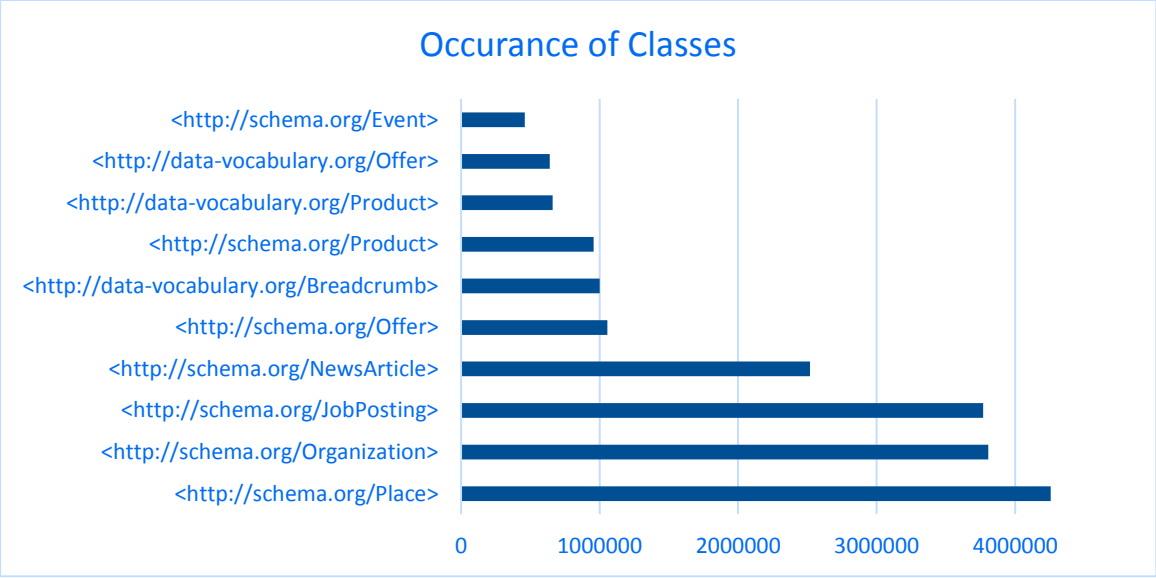

Figure 7: The most used entity classes in the data set

A class represents an individual entity in the microdata domain. As can be seen in above graph the data we have processed contains microdata for the organization, place, product, offer, event etc. The classes are placed as the object in the RDF triples. The predicate for such object is mostly <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>.

For this project, we are only interested in product and offer classes. Therefore, we filtered out all vocabulary and its data that is not related to product or offer. After this, we will refer to product related vocabulary only.

A corresponding entity to the class, e.g. schema.org/Product, is described by its corresponding vocabulary called properties, i.e. attributes, which are been the predicates in RDF triples. Whereas the value of the properties are placed in objects of the triples. The following graph shows the most used predicates or vocabulary for describing a product and its corresponding offer.

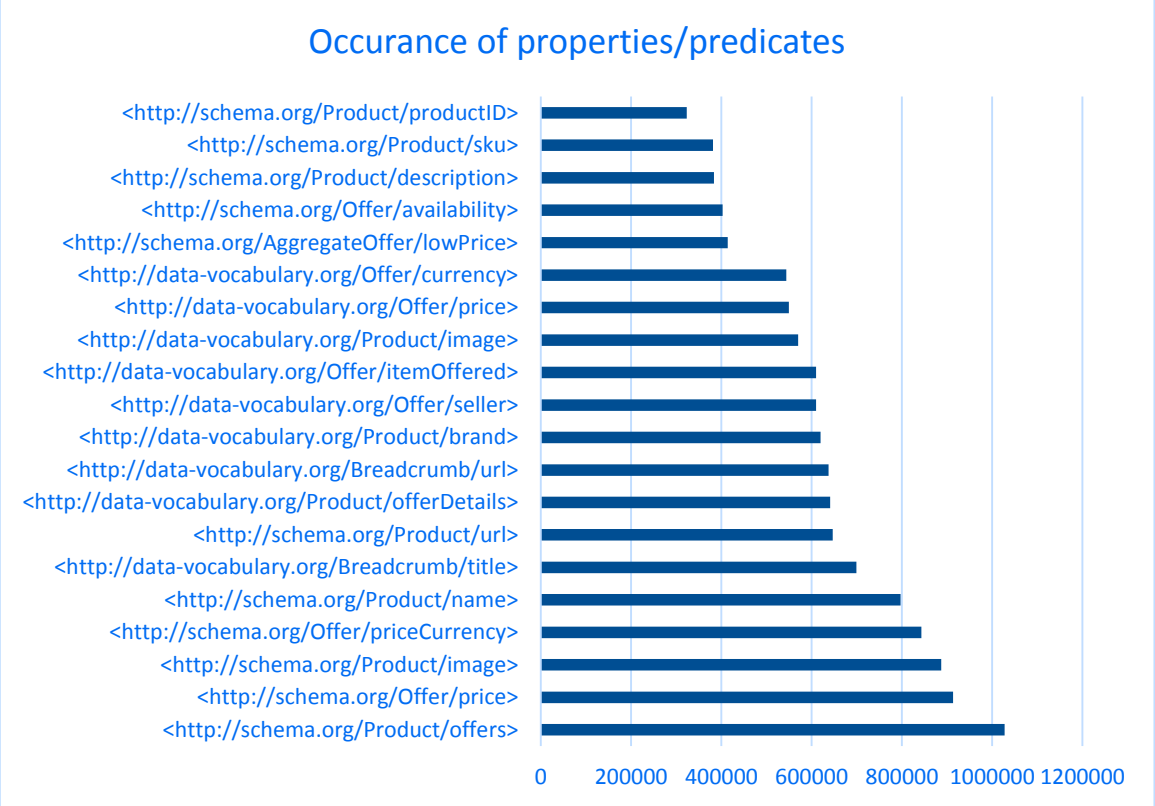

Figure 8: The properties of microdata Product and Offer entities from data.

To unify the data we identified properties that are similar in meaning and present same attribute of product or offer. The following table gives the mapping for the vocabulary that is similar in meaning.

Schema.org	Data-vocabulary.org
http://schema.org/Product	http://data-vocabulary.org/Product
http://schema.org/Offer	http://data-vocabulary.org/Offer
http://schema.org/Product/name	http://data-vocabulary.org/Product/name
http://schema.org/Product/description	http://www.w3.org/1999/xhtml/vocab#description
http://schema.org/Offer/price	http://data-vocabulary.org/Product/price / http://data-vocabulary.org/Offer/price

http://schema.org/Offer/priceCurrency	http://data-vocabulary.org/Product/currency
http://schema.org/Product/productID / http://schema.org/Offer/productID	--
http://schema.org/Product/sku / http://schema.org/Offer/sku	--
http://schema.org/Product/url	http://data-vocabulary.org/Product/url
http://schema.org/Product/image	http://data-vocabulary.org/Product/image

Table 4: Mapping of vocabularies

Table 4 shows mapping for the vocabularies that are used for embedding structured data into HTML formats. Mapping is done between the vocabularies which express similar meaning.

For the unification, we have selected Schema.org as the main vocabulary, as it is recent and more popular currently. After mapping the data vocabulary following graph shows the statistics for unified vocabulary.

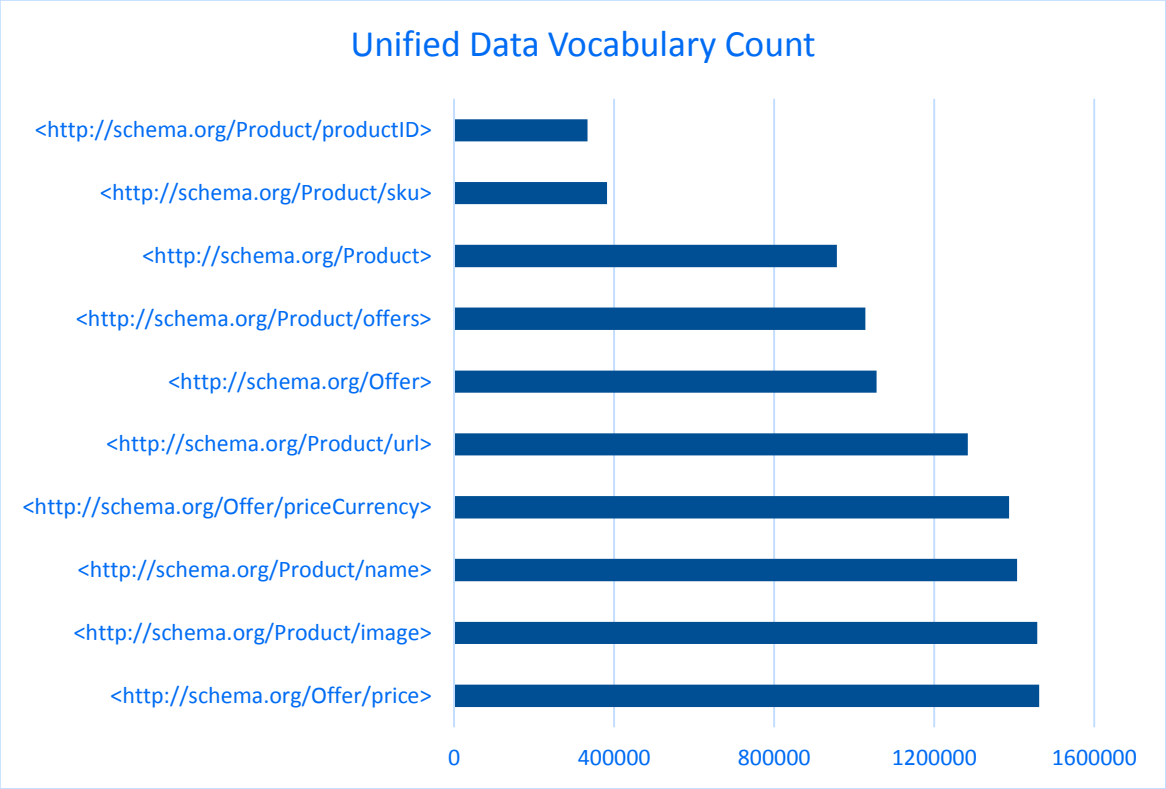

Figure 9: Statistics for the vocabulary, after unification that is used in the collected data.

The number of sequence files, that are processed, are 26500, totally acquiring a size of 1.2 TB on HDFS. The data, when converted to N-Quads, has a size of 51.7 GB with the number of files similar to the sequence files, i.e. 26000. The data is further processed and converted to product entities in tuples. Each tuple was consist of 11 product attributes, i.e. id, name, SKU, description, image URL, product URL, price, currency, availability, provider, and the time stamp. The id to each product was assigned sequentially during the creation of the

product entities from N-Quads. Overall 1.8 millions of product entities were been extracted from the collected microdata.

Before going to the next step of the evaluation process, we see necessary to explain what we call duplicates in the context of the product, specifically in the context of this paper. Duplicate entities (records) are all those entities which refer to the same real-world entity, in our case it is a product. So all those entities which refer to the same real-world product are detected and eliminated. While working on real world data from .ee domain, we noticed that there at two types of duplicates, one the exact duplicate entities and second are those entities which may differ in some attributes but overall they all refer to the same entity. Following are the two types with examples;

1. **Exact Duplicates:** We categorize these entities if they all have exactly similar, 100% equal, name, price, and provider. By provider here we mean the domain, website or online store, which offers the product.
2. **Duplicate Entities (not exactly same attributes):** These are the entities which may differ in attributes but are duplicates. The evidence could be seen from different attributes, e.g. the image address which plays a vital role to identify any duplicate. Obviously, not only image should be same, it may be a mistake. But if the name of the product is very near to each other in similarity and both are presented by one image.

5.2 Experiment Settings

The experiment to evaluate the deduplication process is been done with yarn-cluster. The hardware information on the machines available in the cluster is given as follow;

Quantity	RAM	CPU	Disk	Network
4	29 GB	3 VCPU Intel Xeon E5-1650 v3	1800 GB	1 Gbit/s

Table 5: The hardware specification of machines in yarn-cluster

5.3 Evaluation

Golden Data Set:

The first step in evaluation process was the preparation of Golden data set. Following are the step we performed for the generation of golden data set;

1. We selected 1000 out of 26000 sequence files randomly.
2. The sequence files were processed to URLs, from the URLs we extracted HTML contents.
3. The HTML contents were then processed for extraction of microdata. Microdata was converted to N-Quads format.
4. N-Quads were processed for the formation of entities. A total number of **41163** product entities were been formed from microdata.
5. To eliminate the exact duplicates of entities, see chapter 4 for the definition of exact duplicates, an Apache Spark process were initialized. The Process discarded all exact duplicates from the entity set. A total number of **14000** product entities were eliminated in this step.

6. In the next step, we eliminate all those entities which have no name, or all together has no information of description, product URL, image URL, price, and currency was discarded. A total number of **5000** entities are lift at the end of this step, all other were discarded.
7. In the final step, we checked all remaining **5000** entities and discarded all those entities which were referring to same real world product. My main focus during this step was the name of the entity, its description, image, and product URL. All these were giving quite enough evidence to mark a pair product as duplicate or non-duplicate. The resultant data set contains 3600 unique records.

The resultant data set with 3600 out of 41000 records are named as Golden Data Set.

Evaluation Method

We used document based approach to find the precision, recall and f-score for evaluating the results of the deduplication process explained in chapter 4. Golden data is used as expected or relevant data set and the result of the process is as derived data set. Then by the following formulas, we can calculate precision, recall, and f-score.

$$Precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|}$$

$$F - Score = 2 \times \frac{precision \times recall}{precision + recall}$$

Experiments for Evaluation of the Method

Mainly two things affect the result of deduplication process, specifically in the context of sorted neighbourhood approach. First is the sorting key, obviously selection of attributes are very important but the length and how many characters should be taken from each attribute has the same importance and effect over the results. For example, if we want to define sorting key by combining name, price, and provider in case of a product entity. Because these attributes are more important in recognizing and differentiating a product. Then how many characters or portion of the values of name, price and providers should be combined, all or a part to the attribute value. Second is the selection of sliding windows size, the sliding window size represents how many products next to each other should be compared at any given iteration. The sliding window selects the number of products based on its size and slides over the records by 1 increment in its starting index.

To clarify what size of windows and how many characters should be suitable for the most efficient process of our proposed deduplication process. Following experiments are carried out. During these experiments, we kept the size of other attribute values, i.e. price and

provider, constant, whereas the size of name attribute varies, i.e. 10, 20, 30 or full characters. Similarly, the window size is been varied through the experiments too, i.e. 3, 5, 7 and 9.

Following graph shows the experiments for 10, 20, 30 and Full characters selected from value of name attribute to form sorting key. The sliding window size for each selected characters varies 4 times at each experiment, e.g. for character 10 4 experiments at window size 3, 5, 7, and 9 are done and results are shown.

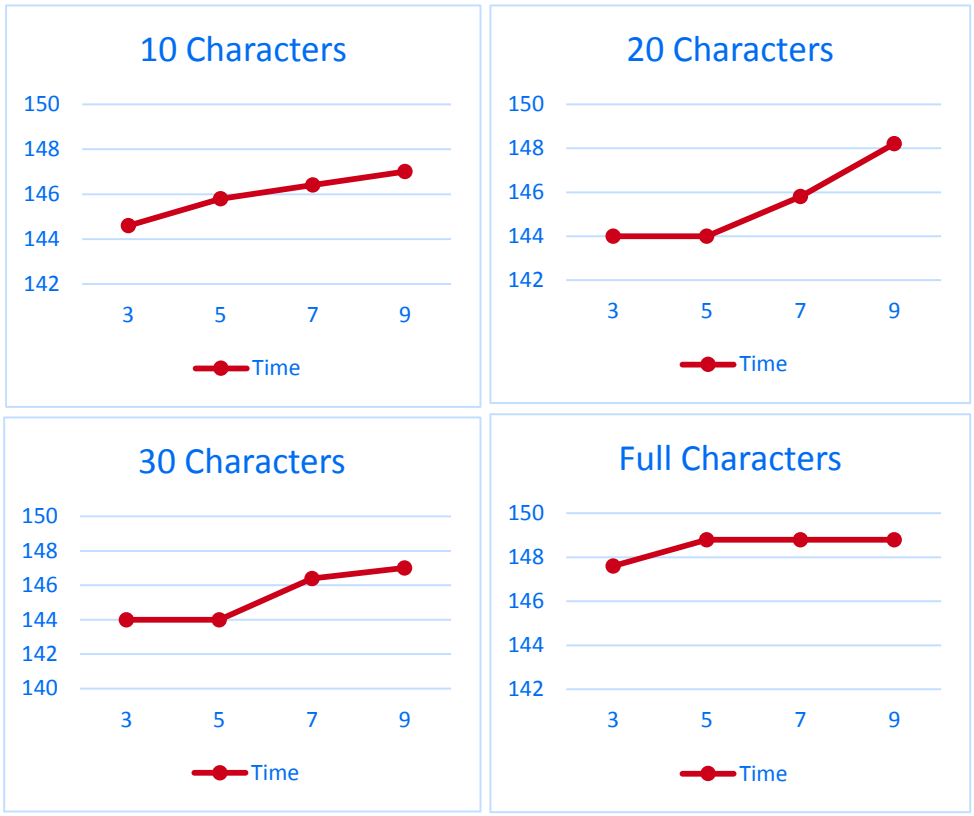

Figure 10: Time variation for different windows size and name characters

Figure 10 shows the CPU time for the different lengths of sorting key characters with respect to different windows sizes. As can be seen from the picture the difference in time with respect to characters is very less. The average in all four cases is 146-147. Whereas in the case of window sizes the difference is more visible, but still less.

Figure 11: Precision of the resultant data set at different windows sizes and name characters.

In Figure 11 the precision has a very visible difference with respect to sorting key's length of characters. For 10 characters it is 0.658 which increases to 0.90 in the case of 30 and full characters. Whereas windows size effects the precision only when the sorting key has 20 characters from attribute name's value.

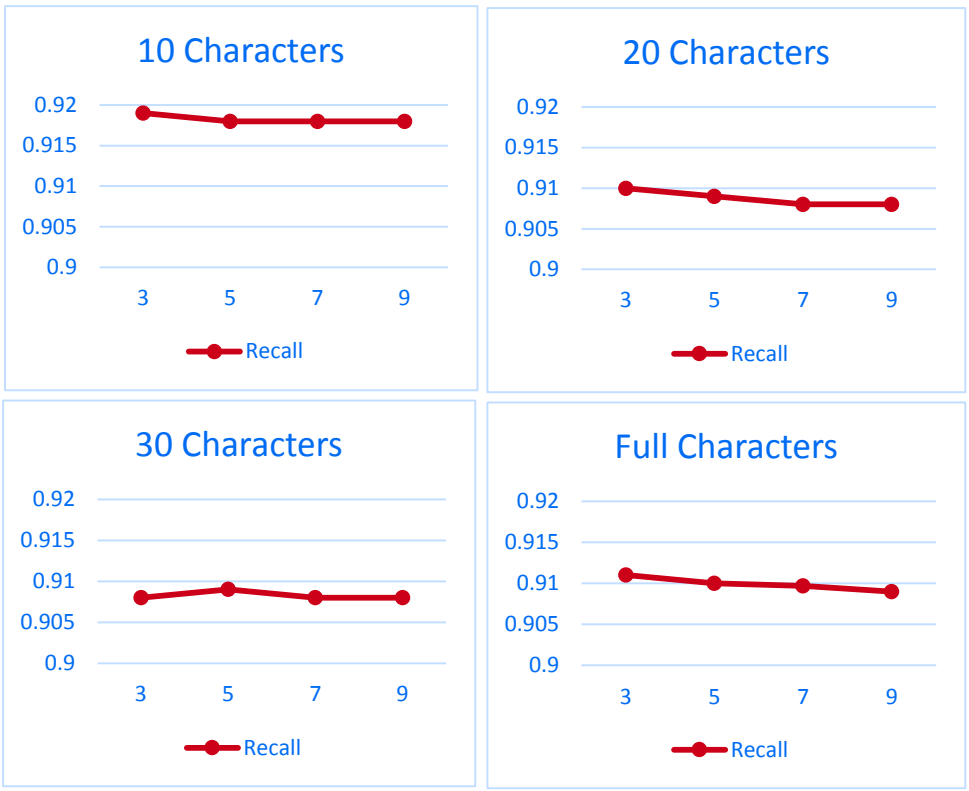

Figure 12: Recall of the resultant data set at different windows sizes and name characters.

When the sorting key length is small the resultant set has more records, obviously more duplicates too. In such case, the precision is low, because of the duplicates, but recall is high as some of the products that were expected is not dropped. So the resultant set has more of the expected results but also more unexpected too. This could be seen from Figure 12 for recall and Figure 11 for precision.

Figure 13: F-score of the resultant data set at different windows sizes and name characters.

This figure shows one of the most important aspect, i.e. f-score, of the evaluation process. And it is clear from the figure that at 30 and full characters the f-score is quite fine, i.e. 91%. In the case of 10 it is 77% only and in the case of 20 it is 87%. From this, we can conclude that the optimum f-score is in 30 characters. So if a sorting key has 30 characters from the name attribute of the product and it has some limited characters from price and provider attributes, one will have the best resultant unified dataset. Hence, increased from 30 characters do not benefit and decrease from 30 will have less precision and less f-score.

Figure 14: Ratio of the f-score to the time of the resultant data set at different windows sizes and name characters.

The higher the ratio f-score to time is, the higher f-score the resultant set will have, and i.e. the resultant dataset is more optimal. The figure shows that as the sorting key increases in length the ratio increases too. But after 30 characters the ratio is constant. Again 30 characters are the best length with respect to time and f-score.

5.4 Threats to Validity

The data was not collected from well-established e-commerce websites, rather it was collected from random websites over the .ee domain. Beside product, it has data about organizations, jobs, event, news articles etc. Similarly, the data was not specific to one language. Because of the very general nature of the collected data, there are some threats to the validity of the process. In the following paragraph, we will discuss these treats one by one.

One of the issues with the data was multiple languages, the data was mainly in Estonian and Russians, and sometimes in English too. So in some cases a single product was been described in two languages, mostly Estonian and Russians. On close, investigation we found that the number of data that has this problem is not in huge amount. During developing the Golden dataset we found nearly 100 such cases, where one product data was provided twice in two different languages.

Because of data in different languages, there was the issue of character encoding. In data preparation process, we tried to solve this issue with decoding back to a UTF-8 character.

But in some cases when there was a spelling mistake or missing character in the Unicode, then it was not possible to decode it back to a valid character.

Developing and organizing the Golden dataset manually may have affected the validation process. Developed with the human perspective, the set is subjective and at some places, it was difficult to decide whether a pair of records are duplicate or non-duplicates. During the development of the set, we mainly consider the similarity of name, description, provider, product and image URLs. Some of the issues during eliminating of duplicates manually for Golden dataset are;

- In cases when the pair of products has names, but no descriptions. Image URL plays an important role in identifying the pair as duplicate or non-duplicate. Sometimes both the products have reference to the same image while the URLs of both products' images has a very little difference, i.e. same image is duplicated but there is a minor difference in the URL text. Such cases could easily be identified by humans but it is difficult to be detected by machines.
- Products with very general names, e.g. clothes or bikes, and very general description. On the resale online web stores, mostly common people put their 2nd hand belongings for sales. In such cases, the owner of the product writes a very general description in her/his own words and give the product a very general name. On investigation, this could be found by the human eye, but again it is difficult for the machine to identify such cases.
- Description and other attributes, e.g. colour and price, included in the name of the product. It was also an issue while developing the Golden dataset. Some products have attributes given within the name of the product. In some case these products are same but in other cases the products are different with different offers.

One more treat that may affect the validity of the process is size of Golder dataset. The set is been formed of 1000 randomly selected files out of 26000, these files may not completely represent the whole dataset.

The treats that are explained above may have an effect on the efficiency of overall deduplication process. But these treats do not have a big impact on the process. The data that is affected with such treats is a percent or less than that. In the data preparation process, we have handled incomplete and inconsistent data and has discarded all the records which may have the greater affect, negative, over the validation of the process.

6 Conclusions

Data deduplication is an important process, and it has application in many aspects of life. Similarly, microdata has great importance in many research domains. In this paper, we have combined the two approaches and presented a linear approach for the extraction of the microdata data from different sources over the web to its unification. Unification of data is vital for achieving better quality and making the data usable in any research work. Our proposed process has the ability to be used in the context of whole microdata and Schema.org domain, whether that is products, persons, organizations, data related to the health sector, or news articles.

One of the important aspects of data deduplication process is the definition of sorting, i.e. blocking, key for indexing of records. In this paper, with practical experiments, we have found the optimal length of sorting key with respect to f-score. There is much written about the quality of attributes and choosing the value of attributes for defining of sorting key, but how much the characters of key effects deduplication is not considered.

Spark is a distributed framework developed on top of MapReduce. By using Apache Spark the deduplication process is made faster and quite efficient. But still there is a space for the improvement. The results of our proposed deduplication process could be made more accurate and the process faster by putting more concurrency into the method. With increasing the number of executors in Apache Spark the method could be made much faster. To achieve this, there is need of a well-established and less weighted communication mechanism between the executors.

We believe that with creating of big Golden dataset for validation of the process may make the process more efficient and fine-tuned. The development of big dataset requires more time and language understanding of the .ee domain.

The deduplication process could also be improved with putting more effort into the data preparation phase of the process. Methods like phonetic encoding and splitting the values of attributes to well-defined and meaningful words. By doing these techniques the data comparison could be improved.

Our project is pure research oriented, so the implementation of the process could be improved and made more usable by providing a user-friendly interface. By using the interface the user of the application should be able to locate HTML pages or web addresses and the application should provide the resultant unified data set.

7 References

[1] J. Ronallo, "HTML5 Microdata and Schema. org.", *Code4Lib Journal 16,* 2012.

[2] WHATWG, "Microdata - HTML Draft Standard", WHATWG, [Online]. Available: https://html.spec.whatwg.org/multipage/microdata.html. [Accessed 2016].

[3] Krueger and Alan B, "How computers have changed the wage structure: Evidence from microdata, 1984-1989", *National Bureau of Economic Research,* 1991.

[4] Bartelsman, Eric J. and Mark Doms., "Understanding productivity: Lessons from longitudinal microdata", *Journal of Economic literature,* 2000.

[5] M. U. Özmen and O. Sevinc, "Price rigidity in Turkey: Evidence from micro data", *Emerging Markets Finance and Trade*, 2015.

[6] Guimaraes, Bernardo, André Mazini, and Diogo de Prince Mendonça., "Time-dependent or state-dependent pricing? Evidence from firms' response to inflation shocks. ",2015.

[7] R. Meusel, C. Bizer and H. Paulheim, "A web-scale study of the adoption and evolution of the schema. org vocabulary over time.", *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics. ACM,* 2015.

[8] Mühleisen, Hannes and Christian Bizer, "Web Data Commons-Extracting Structured Data from Two Large Web Corpora", *LDOW 937*, 2012.

[9] J. Singh, "Understanding Data Deduplication", 01 09 2009. [Online]. Available: http://www.druva.com/blog/understanding-data-deduplication/.

[10] T. T. a. N. L. T. Thwel, "An efficient indexing mechanism for data deduplication.", *Current Trends in Information Technology (CTIT), International Conference on the. IEEE,* 2009 .

[11] Q. Z. L. a. X. Z. He, "Data deduplication techniques.", *Future Information Technology and Management Engineering (FITME), International Conference on. Vol. 1. IEEE,* 2010.

[12] F. M. G. A. a. M. S. Shieh, "De-duplication Approaches in Cloud Computing Environment: A Survey.", *International Journal of Computer Applications 120.13,* 2015.

[13] G. Developers, "Microdata", [Online]. Available: https://developers.google.com/gmail/markup/reference/formats/microdata?hl=en.

[14] Barker, Phil and Lorna M, "What is schema. org?", *Centre for Educational Technology, Interoperability and Standards,* 2014.

[15] Any 23, "Introduction to Apache Any23", 03 11 2013. [Online]. Available: https://any23.apache.org.

[16] Spark, "Spark Overview", Apache, [Online]. Available: https://spark.apache.org/docs/latest/index.html.

[17] H. A. T. a. E. R. Köpcke, "Evaluation of entity resolution approaches on real-world match problems.", *Proceedings of the VLDB Endowment 3.1-2,* 2010.

[18] P. Christen, Data Matching, Springer, 2012.

[19] L. A. T. a. E. R. Kolb, "Dedoop: efficient deduplication with Hadoop.", *Proceedings of the VLDB Endowment 5.12,* 2012.

[20] K. a. Dr.R.Rangarajan, "An Approach to Duplicate Record Detection Using Similarity Metrics and Anfis", *Journal of Computational Information Systems,* 2012.

[21] A. K. P. G. I. a. V. S. V. Elmagarmid, “Duplicate record detection: A survey.”, *Knowledge and Data Engineering, IEEE Transactions on 19.1,* 2007.

[22] F. Naumann, *Sorted Neighborhood Methods*, Hasso Plattner Institute, 2.7.2013.

[23] E. Miller, “An Introduction to the Resource Description Framework”, D-Lib Magazine, [Online]. Available: http://www.dlib.org/dlib/may98/miller/05miller.html.

[24] D. Beckett, “RDF 1.1 N-Triples: A line-based syntax for an RDF graph”, World Wide Web Consortium, 25 02 2014. [Online]. Available: https://www.w3.org/TR/n-triples/.

[25] G. Carothers, “RDF 1.1 N-Quads,” W3C Recommendation, 25 02 2014. [Online]. Available: https://www.w3.org/TR/n-quads/.

[26] P. Christen, “A survey of indexing techniques for scalable record linkage and deduplication.” *Knowledge and Data Engineering, IEEE Transactions on 24.9* , 2012.

[27] Culotta, Aron and Andrew McCallum, “Joint deduplication of multiple record types in relational data”, *Proceedings of the 14th ACM international conference on Information and knowledge management,* 2005.

[28] Chandola, Varun, Arindam Banerjee and Vipin Kumar, “Anomaly detection: A survey”, *ACM computing surveys (CSUR) 41.3,* 2009.

[29] R. a. H. P. Meusel, “Creating Large-scale Training and Test Corpora for Extracting Structured Data from the Web”.

[30] C. Crawl, “Common Crawl,” Common Crawl Project, [Online]. Available: http://commoncrawl.org.

[31] S. Chaudhuri, “Robust and efficient fuzzy match for online data cleaning.”, *Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM,* 2003.

Appendix

I. Implementation Code

The java applications for the complete deduplication process could be found at the following git hub repository;

https://github.com/KhalilRehman/microdeduplication.git

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Khalil Ur Rehman** (date of birth: 02/03/1989),

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Microdata Deduplication with Spark,

(title of thesis)

supervised by Peep Küngas,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **19.05.2016**