Semantic Based Approach for Entity Matching on Noisy Semistructured Data

Nikhil, Acharya

Matriculation number: 3064518

May 16, 2021

Master Thesis

Institute of Computer Science

Supervisor:
Dr. Diego Collarana, University of Bonn

Examiners:
Prof. Dr. Jens Lehmann, University of Bonn
Prof. Dr. Elena Demidova, University of Bonn

INSTITUT FÜR INFORMATIK
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

Declaration of Authorship

I, Nikhil Acharya, declare that this thesis titled, 'Semantic Based Approach for Entity Matching on Noisy Semistructured Data' has been written independently and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- Where none other than the specified sources and aids were used.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

Acknowledgements

Firstly, I would like to thank my supervisor and mentor at work Dr. Diego Collarana for his support, ideas and encouragement. His constant review and feedback time to time has really helped me steer my work in the right direction and develop serious interest in the area of Semantic Web Technologies.

I am also grateful to my team in Fraunhofer IAIS based in Dresden who really have given me all the resources, suggestions and their valuable time for feedbacks whenever I needed it.

I would also like to extend my thanks to Dr. Carsten Winkelholz who is the Head of Research Group Information Visualization and Interaction at Fraunhofer FKIE and was my mentor at the Institute when I worked there as a student assistant. My interest in the area of Natural Language Processing and Data Visualization would not have been possible without his encouragement.

Also the success would not have been possible without constant support of my parents and my sister who always backed me throughout the tasks and the decision making. Lastly cheers to my friends who have been with me during the challenging times and were always available when in need.

RHEINISCHE
FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

Abstract

Institute of Computer Science

Master of Science

by Nikhil, Acharya

Entity Matching on Noisy Semi-structured Data, such as RDF graphs is an active field of research where many approaches have been proposed for interlinking individuals of Knowledge Graph datasets. These methods have included schema learning techniques, string matching on labels using SPARQL, attribute based approaches and methods that use Knowledge Graph embeddings. This thesis proposes a novel entity matching pipeline that parses Knowledge Graphs to fetch literals and label values, handles semantic interoperability conflicts and performs attribute based matching using data from literals by using a Deep Learning model. We test our approach on the Itunes dataset and Wikidata-DBpedia dataset. We believe that our technique can help interlink individuals of various RDF datasets and extend knowledge via entity matching. Our technique also promises to be robust to dirty, semi-structured data when literals have large texts.

Keywords: Entity Matching, Semantic Interoperability Conflicts, Interlinking, Link Detection

Contents

Chapter 1

Introduction

As per Domo [1] in the world 1.7 mb of data was generated every second in 2020. In terms of text that would mean a lot, i.e in the range of 100k characters per second. Analysis of such large texts or data is not easy because they are mostly unstructured/semistructured. However there is a need to expand knowledge consistently and filter redundant , unnecessary data. This data once processed is used in the fields of Artificial Intelligence (AI) which include image and speech processing, medical diagnosis, autonomous driving, robotics etc.

One of the aims of Artificial Intelligence is to reduce human intervention in order to perform the necessary tasks in our daily lives. These tasks highly on rely quality of data. One of the important ways to improve quality of data involves having good, well defined structures for representing data. The analysis in such a case would be seamless with AI tasks achieving maximum accuracy. There has been enough debate on improving the quality of representation of data. While its hard to agree on a single idea to unify heterogeneous data resources, we do have a World Wide Web. While World Wide Web has its drawbacks, to overcome these drawbacks link data web was created where each resource is represented via a Universal Resource Identifier (URI).

Linked data helps in linking heterogeneous data across different graph databases using URIs. We can apply state of the art Deep Learning/Machine Learning algorithms to these graphs. This helps us analyse, add more knowledge and remove redundant information in the linked data world. Data here is stored using knowledge graphs and stored in Graph Databases. Graph databases offer various advantages over conventional relational databases [2] and one of the them is that it offers a more flexible schema which helps us understand semantic relation between entities. Semantic relationships can now help us match resources referring to the same entity across different graph databases which helps expand knowledge and this is one of the biggest use case of our problem.

1.1 Motivation

One entity can be present across different databases with different kinds of information of our interest. Information for entities can be represented via properties. These properties comply with a schema of the graph database which can also be called an ontology. Our task is to match or link same entities across these databases. There have been various techniques developed to link or match entities which we will be discussing in the coming sections. Interlinking is key to knowledge expansion and deduplication.

Data can be unstructured, dirty, incomplete, redundant etc. These issues are not desirable but surely is a predominant problem in the data world including in the field of Linked Data. One of the advantages also of semantically linked resources is understanding of unstructured data can be simplified. We may have to manually resolve these issues or resolve by defining new semantic relations to transform data to a more structured or semi structured format.

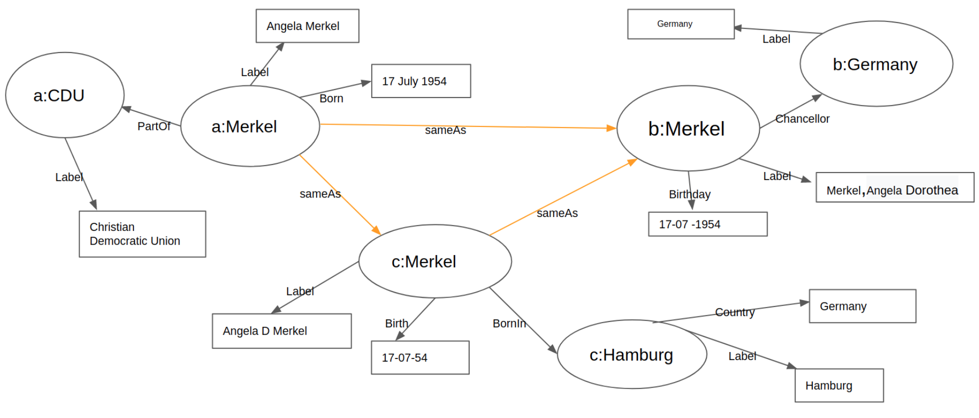


FIGURE 1.1: Interlinking Entities

Consider Figure 1.1 where lets say we are creating a question answering model on leaders around the world. We would like include information of Angela Merkel. For the model to perform well we need maximum information on leaders and here we have some information on Angela Merkel divided across different graphs (Databases). We can integrate information from 3 databases there. Each database has different knowledge regarding entity i.e., Angela Merkel. They have different properties we can learn from. So we have to propose techniques to map these entities together with the challenge where we see different formats for labels, dates for properties and hence the interlinking may not be so straightforward. The intent clearly is to design robust techniques where we can apply this across different graphs to different models. Such a technique can also reduce redundant data i.e in our example so many labels, birthday properties representing same entity. We can eliminate them if they are same. This is termed as deduplication.

1.2 Problem Statement and Challenges

Using the illustration above we define our problem statement relevant to our task. From there we move onto the challenges we faced during the problem solving.

Problem Statement

Given two sets of graph databases G_1, G_2 **having classes** $C_1 \in G_1$, $C_2 \in G_2$ **with each having its respective individual set** $I_1 \in G_1$, $I_2 \in G_2$ **and property set** P_1, P_2 **where** $domain(P_1) \in C_1$, $domain(P_2) \in C_2$ **with semantic interoperability conflicts s. We return a set of labels** $EM \in \{1, 0\}$ **which maps if an individual** $i \in I_1 \bigcup I_2$ **is in** $I_1 \bigcap I_2$ **or** $I_1 \coprod I_2$**.**

$$I_1 = resolve(I_1, P_1), I_2 = resolve(I_2, P_1) \tag{1.1}$$

$$T = Transform(I_1, I_2) \tag{1.2}$$

$$EM = f(T) \tag{1.3}$$

where f is the Entity Matching function, T is transformed dataset, resolve is the conflict resolution function and Transformation is the function to transform data from rdf properties to vectors.

1.2.1 Challenges

Challenge 1: Lack of appropriate data

In order to map individuals which refer to same entity we need to have enough individuals and properties to learn from. If that is not the case our function f in problem statement 1.2 cannot return the right labels. However it has been a big challenge in our experiments as properties are not complete for individuals.

Challenge 2: Schema Mapping

When we deal with schemas for graph databases each property can be bound by different rules depending on the schema. These overheads can be handled via creating links manually from property to property for properties we wish to consider for the analysis which we call mappings. Eg- In Figure 1.1 birth property is equivalent to birthday property. The more number of properties we deal with, the more manual intervention we need.

Challenge 3: Semantic Interoperability Conflicts

Dealing with semantic interoperability conflicts is another challenge. Our conflict function may not resolve each conflict. It is not possible completely because we need manual some intervention when values for properties are subjective for each case. However we have tried to define and resolve most of the conflicts. Eg- Different date formats for birthdays.

1.3 Contributions

In this section, we list down the key contribution of this work corresponding to the various challenges posed by the discussed problem:

Contribution 1

Define a pipeline which performs Entity Matching on RDF datasets by parsing through literals and labels. Our method specializes in dealing with literals that have really long text and has dirty data because the Deep Learning model phase is capable of handling such anomalies. The approach uses models like RNN, Attention that can compare literals which have long text and dirty data.

Contribution 2

Deduce a solution to handle semantic interoperability conflicts to achieve more accuracy on Entity Matching. Handling the conflicts for properties we consider really increases the accuracy.

Contribution 3

Define a method to parse a knowledge graph to extract values for literals using Depth First Search or Breadth First Search algorithms. Most existing methods use SPARQL wrappers to fetch value of literals. SPARQL wrappers are generally slow compared to graph parsers.

1.4 How the Thesis is Organised

The study is organised into the following chapters: The introduction chapter involves giving an overview of the linked data web and its application in problem solving. Then we define our problem statement and its challenges. Coming to the background chapter we give an overview of all concepts we have used to solve our problem. They include Entity Matching, Link Detection, Knowledge Graphs, Deep Learning, Semantic Interoperability Conflicts. Next in the related work we discuss all the techniques used in the field of Entity Matching for Knowledge Graphs, Link Detection Workflow, Schema learning. The techniques include LiteralE, KnoFuss etc. In the Entity Matching pipeline chapter we discuss our pipeline in detail which has pre processing, conflict handling, data transformation and the deep learning model. The Evaluation chapter includes presenting results we have got in the two sets of datasets we have used. In the final chapter of conclusion and future work we discuss our analysis, results inferred, drawbacks and how the work can be extended using existing approaches.

Chapter 2

Theoretical Background

In this section, we discuss the necessary background for understanding and solving our Entity Matching problem. We will start with an introduction and definition of Entity Matching, then present the concepts of Knowledge Graphs and RDF followed by an overview of types of Semantic Interoperability Conflicts. Lastly, we discuss Neural Network (NN) and other Deep Learning concepts which include character, word embeddings.

2.1 Entity Matching (EM)

Entity Matching refers to matching two entities or resources that refer to the same real-world object. The problem is predominant in numerous domains like Natural Language Processing, Image Processing, and Information Retrieval [3].

Entity matching can also be called record linkage, deduplication, or coreference resolution under different contexts:

- **Deduplication:** Refers to elimination and identification of duplicate blocks of records within dataset. Various Hashing methods are used for deduplication [4] .
- **Record Linkage:** Determining if records belong to the same entity if they come from multiple resources. A usecase we often see in the field of fraud detection [5], healthcare [6].

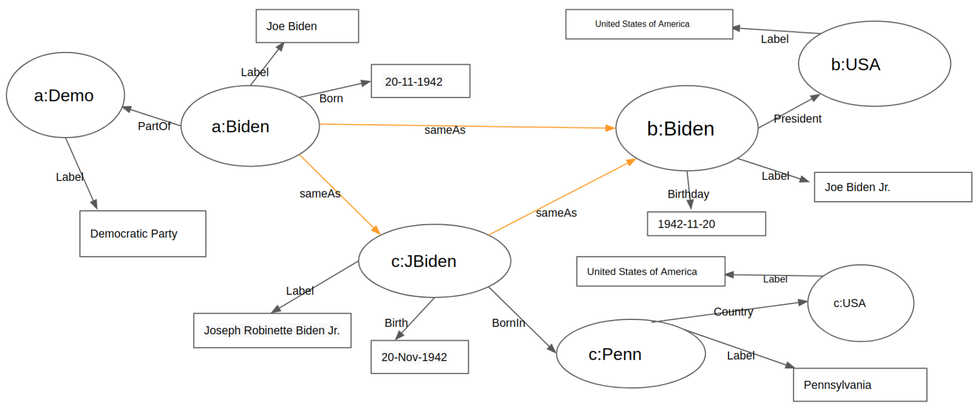


Figure 2.1: Sample Knowledge Base

- **Coreference Resolution:** To resolve all references to a given entity in a text corpora. The task involves replacing relevant pronouns, alias with the subject/entity.

Also, in the context of this thesis linking two resources because they are same will be referred to as Interlinking. This is like Paraphrasing the EM problem. Note that in general, interlinking need not always be about linking same resources.

Lets look Figure 2.1 of a sample Knowledge Base. In this case we match all resources of Joe Biden via Record Linkage by linking them. There are duplicate blocks now for birthdate and label which can be removed via deduplication. Replacing mutiple repetitive labels for resource Joe Biden with the correct label would be called Coreference Resolution. Inferring a link from USA to Pennsylvania would be a part of interlinking task eventhough they are not same but can be linked with a new relation "hasState".

2.2 Knowledge Graphs (KG)

Knowledge graphs are collections of interlinked entities with certain defined properties. Using these properties and descriptions forms a network that helps to learn the context of the information stored. A knowledge graph contains a triplet which includes a $< subject >$, $< predicate >$, $< object >$. Many publicly available knowledge graphs like DBpedia and Yago follow standards like OWL (Web Ontology Language) and RDFS (Resource Description Framework

Schema). These standards help define a structure, create axioms which in turn are used to create an Ontology. RDFS and OWL has many parallels with Object-Oriented Programming, but Web Ontology Languages can be considered more flexible as they change fast due to changing Internet data sources. Next, we define certain concepts of RDF, OWL and other foundations for our tasks. Most of the illustrations of these concepts are taken from W3C document [7] .

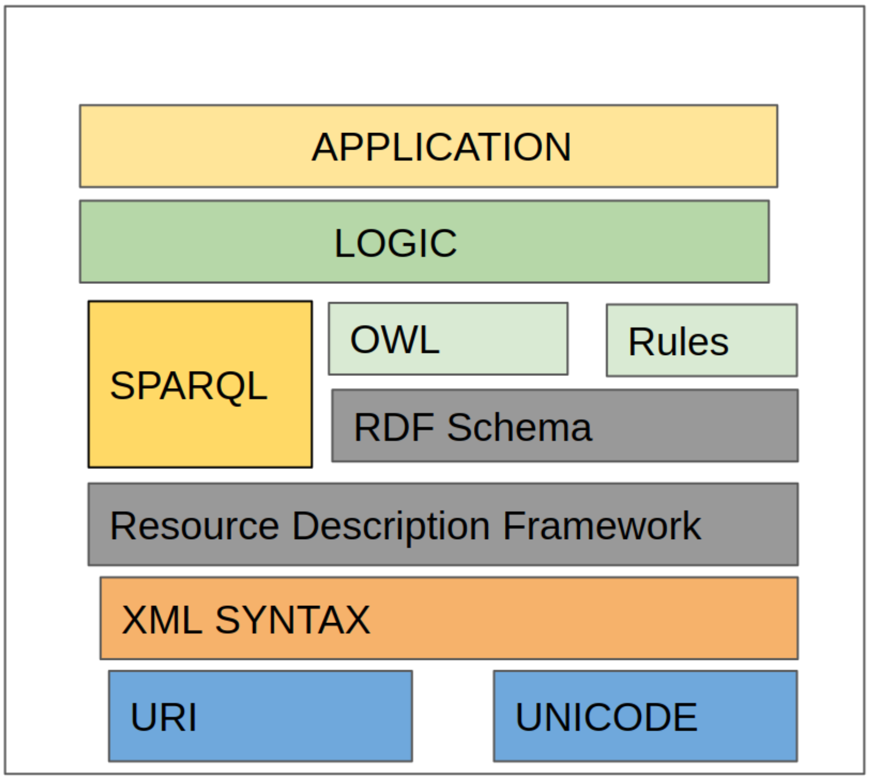


FIGURE 2.2: Semantic Web Stack

Semantic Web is an extension of World Wide Web set by standards of World Wide Web Consortium. We define the aspects of semantic web from the paper "The XML and Semantic Web Worlds: Technologies, Interoperability and Integration. A Survey of the State of the Art" [8]. In Figure 2.2 we see a semantic web stack which explains semantic web applications layer by layer. At the root level we have the URIs which are used to represent the resources. It is used to identify real world objects, concepts and information on web sources. These URIs can be represented by semi structured data using XML syntax. These XML structures use the resources to define and instantiate web ontologies which is done using Resource Description Framework Schema. It provides constructs for the description of types of objects (classes), type hierarchies (subclasses), properties that represent object features (object properties) and property hierarchies (subproperty). Above these layer of resources and schema we would like to retrieve and modify data at a query level. This is done using SPARQL. These feature like data fetch, data retrieval and data definitions helps us define logic for our Semantic web applications. These Semantic Web applications can help perform the desired Artificial intelligence tasks.

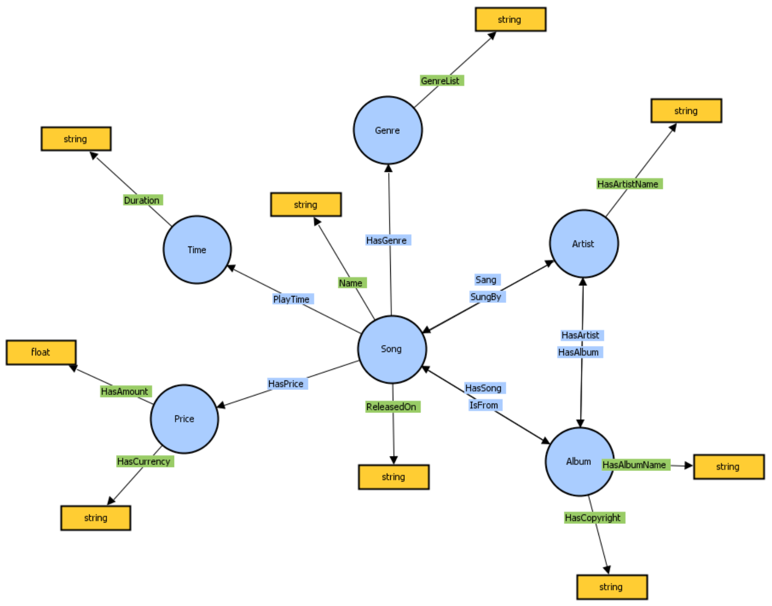


Figure 2.3: Itunes OWL Ontology

Below we define some of the relevant concepts from RDF schema and OWL(Web Ontology Language) using the standard w3c document [9]. We cite examples from Figure 2.3

- **Resource** Resources are the subset of everything. It is an instance of class. Every resource is represented as an URI. All the nodes and relations in the graph correspond to a Resource.
- **Property** Property is an instance of class. It can be partly similar to a method in object oriented terms. All relations correspond to a property. They include HasPrice, ReleasedOn, Name etc.
- **Class** Classes are set of resources that are RDF classes. In lay man terms it corresponds to a type or category of an entity. Every Node in the RDFS graph corresponds to a class. Eg. Song rdf:type rdfs:Class

- **rdfs:domain** Domain is used to indicate that the property applies to instances of a certain class. Refer to figure 2.3 we see for the property ReleasedOn Album class is the domain. Also HasAmount property Domain belongs to Price class.
- **rdfs:range** Range is used to indicate that the values returned by the property are instances of a certain class or a datatype. For the property duration the range is string. Similarly for property HasPrice, range is Price class.
- **owl:DatatypeProperty** A type of property which can be considered a literal whose domain corresponds to a class while range corresponds to a primitive datatype like string or int or float. Eg- HasAmount, HasCurrency
- **owl:ObjectProperty** A type of property whose domain and range both correspond to a class resource. Eg- HasAlbum, HasPrice.
- **Individuals** Individuals are instances of class. Individuals in turn take property values which can be literals or resources. Eg- Luis Fonsi is an individual of type Artist. Eg: despacito rdf:type Song .
- **owl:sameAs** Individuals can be linked to one another using sameAs property. It indicates both individuals belong to same entity.
- **owl:differentFrom** Individuals can be linked to one another using differentFrom property. It indicates both individuals belong to different entities. Eg: Itunes:despacito owl:sameAs spotify:Despacito .
- **owl:equivalentProperty** The owl:equivalentProperty construct can be used to state that two properties have the same property extension. They are sematically equivalent [7].

In above figure 2.3 the schema is designed using OWL and RDF Schema. We here describe Song, Album, Artist, Genre, Price and Time class. Next there are Data Properties for classes like SongName , ArtistName , GenreList , AlbumName etc. Object properties connect one class to another class. Itunes ontology has object properties like HasAlbum, HasSong, HasArtist, HasGenre. While defining properties we need to define their range and domain. They define what value the triplets can take for $< Subject >$, $< Object >$ slot while defining properties for individuals. The domain of the property refers to the values the Subject can take and Range refers to the Object field of the triplet respectively. Lets look at an example of object property below

Lets say Despacito is of type Song described by triplet
$< itunes : Despacito > < rdf : type > < Itunes : Song >$
Lets say Luis Fonsi is of type Artist
$< itunes : LuisFonsi > < rdf : type > < Itunes : Artist >$
Now we define the property $< Itunes : HasArtist >$
$< itunes : Despacito > < Itunes : HasArtist > < itunes : LuisFonsi >$

2.3 Link Discovery in Knowledge Graphs

The knowledge graphs can be expanded via creating links between set of resources. This helps knowledge expansion and hence the more links we discover or infer, the more we learn. Entity Matching problem can be called a subset of link discovery problem. Links can be inferred, learned or added manually. If two resources are the same, then a link "sameAs" between them represents the Entity Matching problem as mentioned in section 2.1. We test all pairs of resources for a pair of similar classes from different schema using a Cartesian product and analyse the result using a similarity function. Using the result of the function we determine if a relation can connect two classes. Its is represented in equations 2.1 and 2.2 below.

Link Discovery Problem

Given two resources S and T plus a relation R. Find all pairs

$$(s, t) \in S \times T \mid R(s, t) \tag{2.1}$$

The result is represented as a set of links called a mapping

$$M_{\mathbf{S,T}} = (a_{\mathbf{i}}, R, b_{\mathbf{j}}) \mid a_{\mathbf{i}} \in A, b_{\mathbf{j}} \in B) \tag{2.2}$$

The above Link Discovery Problem(formula 2.1 and 2.2) is formulated using the paper "A survey of current link discovery frameworks" [10]. Lets observe figure 2.4 where we want to know if we can deduce a link between "Itunes:Despacito" and "Spotify:Despacito". Here in accordance to the Link Discovery Problem

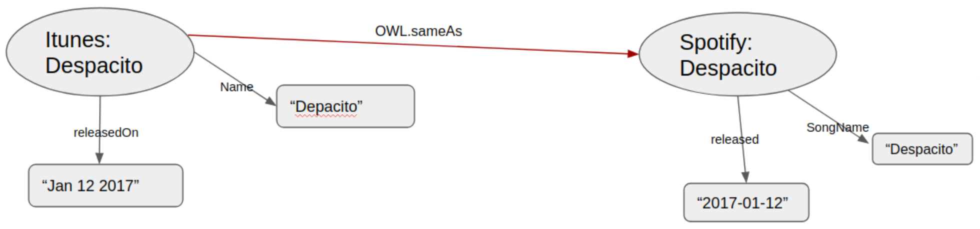


Figure 2.4: Link Discovery for sameAs property

our R is "OWL.sameAs" while our resources (s and t) are Spotify and Itunes. Our classes A and B are "Itunes:Song" and "Spotify:Song" with class members $a_i \in A$ as Itunes:Despacito and $b_i \in B$ as Spotify:Despacito. We can calculate confidence of an expected link using $(a_\mathrm{i}, R, b_\mathrm{j}, sim(a_\mathrm{i}, b_\mathrm{j}))$. In the next section we see various methods that can be used to infer a link.

We list some of the approaches used for link detection below

- String based similarity for link detection [11]
- Use semantic neighborhood of a resource [12]
- Reinforcement learning for path traversal along the graph and inferring links to answer questions [13]
- Learning the literals for corresponding entities

2.4 Semantic Interoperability Conflicts

This section is referred from the PhD thesis by Irlan Grangel-González [14]. Our semantic approaches must be able to exchange data with unambiguous, shared meaning. Our algorithms should be able handle these conflicts. These are called semantic interoperability conflicts.

- **STRUCTUREDNESS** This interoperability conflict occurs whenever data sources are described at a different level of structuredness, e.g., structured, semi-structured, and unstructured. In Knowledge graphs for a data property like name of an Artist we can either have a name field or the field split into first and last name. For example- David Guetta for name field, Guetta for last name and David for first name.

- **SCHEMATIC** Data Sources can be modelled using a different schema. We can represent age as a number or a string or an integer. Eg- 52 as XSD.integer or ”52” as XSD.String.
- **DOMAIN** The conflict where different interpretations of the same domain is represented. Different interpretations include: i) Homonym: the same name is used to represent concepts with a different meaning; ii) Synonym: distinct names are used to model the same concept; iii) Acronym: different abbreviations for the same concept are employed. Eg- Name for singer field ”Flo Rida” also can be called ”Tramar Lacel Dillard”
- **REPRESENTATIONS** This interoperability conflict is described when different representations are used to model the same concept. Representation conflicts include: i) Different scales or units; ii) various values of precision; iii) incorrect spellings. Eg- Time field in minutes and seconds can be represented as ”4:30” or ”4 minutes and 30 seconds”
- **GRANULARITY** This interoperability conflict appears when various interpretations of the same domain are represented. Different interpretations include: i) Intra-aggregation: the same data is divided differently, e.g., full person names against first-middle-last ii) Inter-aggregation: appears when there exist sums or counts as added values. Eg- Price field may have currency in Euros or Cents or weight field with kilogram or grams
- **MISSING ITEM** This interoperability conflict occurs whenever different items in distinct data sources are missing. Missing Item comprises: i) Missing attributes; ii) Missing content. Eg- We may have values missing in a non mandatory field like say Fax.

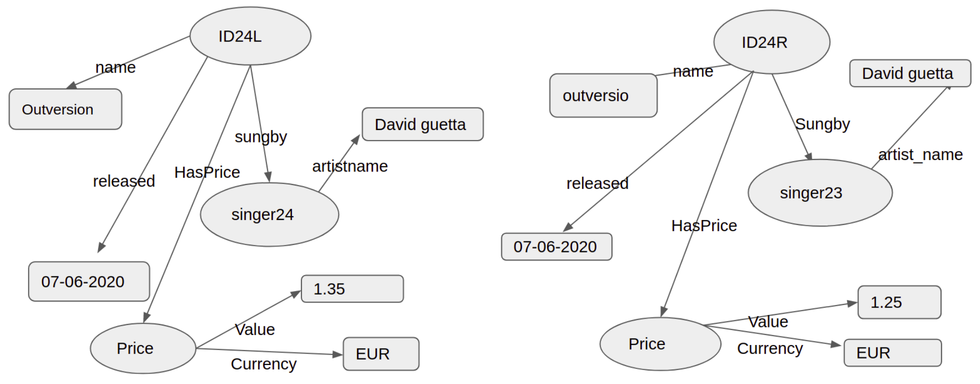


FIGURE 2.5: Semantic Interoperability Conflicts Example

In the Figure 2.5 we need to conciliate semantic interoperability conflicts in order to perform a link detection or an entity matching task on the above 2 rdf graphs. We can observe some conflicts there like domain conflict for artistname relation, representation conflict for released relation and value relation. To infer a link from ”ID24L” to ”ID24R” we need to handle these conflicts.

2.5 Deep Learning

2.5.1 Neural Network (NN)

In this section we will give an overview of Neural network adapted from the book “Neural-Network-Design” [15]. Neural Network is one of the most commonly used Machine Learning Algorithms. Figure 2.5 represents a typical single layer perceptron. In case of this perceptron, we have X inputs connected to S neurons with each input weighted by W and each neuron has a bias function. The result of each neuron which is weighted input plus bias is passed on to a transfer function which gives us the required output. Mathematically we denote it as $\sigma(WX + b)$.

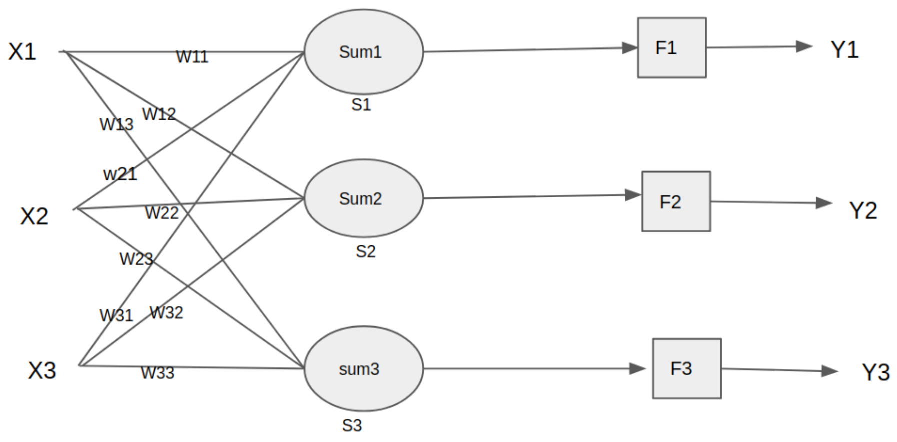


FIGURE 2.6: A typical Single-Layer Perceptron (MLP)

Next we look at the multi layer perceptron as represented in Figure 2.7 . Here we extend the single layer perceptron by adding more layers and neurons. Lets say

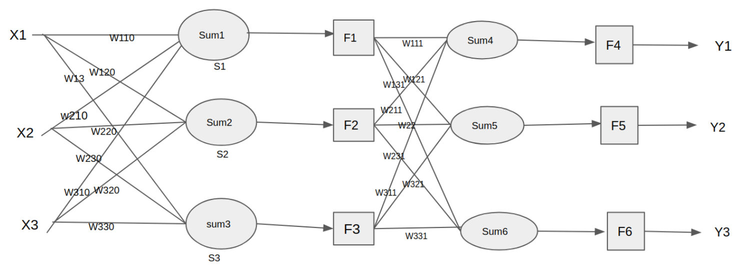


FIGURE 2.7: A typical Multi-Layer Perceptron (MLP)

outputs of the first layer are further fed into new set of neurons and their output to next set and so on. We would have a weight matrix of all the neurons plus bias for each neuron and transfer functions. The first layer includes first set of neurons,transfer functions and second layer includes the second sets of functions and weights. This can further be extended until n layers but our illustration and Figure 2.7 will be restricted to 2 hidden layers.

Let n_1 and n_2 be the number of the number of neurons in the first and second hidden layers. Similarly W_1 and W_2 be weight matrix for both hidden layers and now let b_1,b_2 be the bias respectively. There are non linear activation functions f_1 and f_2 for each layer. We can say the input $X \in \mathbf{R}^{dim_x}$ where dim_x is dimension of input x. Next for weight matrix of first layer $W_1 \in \mathbf{R}^{dim_x \times n_1}$. Similarly for weight matrix of $W_2 \in \mathbf{R}^{n_1 \times n_2}$. Next the bias terms $b_1 \in \mathbf{R}^{n_1}$ and $b_2 \in \mathbf{R}^{n_2}$.

$$\begin{aligned} NN_{Singleperceptron}(x) &= f(Wx + b) \\ NN_{Multiperceptron}(y) &= f_2(W_2(f_1(xW_1 + b_1)) + b_2) \end{aligned} \tag{2.3}$$

We need to map the weighted inputs plus bias to the desired output(typically between 0 to 1). The most common way of doing it is by using transfer functions. Some of the most commonly used transfer functions are listed below.

- Rectified Linear Unit: This function will output the input directly if it is positive, otherwise it returns a zero. This can be used to resolve vanishing gradient issues in the networks

$$ReLU(x) = \max(0, x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.4}$$

- Sigmoid Function: This function helps map the input to any value between 0 and 1. This function is smooth, continuous facilitates smooth backpropagation.
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

- Hyperbolic Tangent funcion (tanh): This function helps map the input to any value between 1 and -1. The function has a threshold of 0 with any value above 0 is considered high and value below 0 is mapped to negative values.
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.6}$$

- Step Function: Here we choose a threshold to determine a high or low output. This function however is non differentiable, hence cannot be used for backpropagation.

$$hardtanh(x) = \begin{cases} 0, & \text{if } x <= T \\ 1, & \text{if } x > T \end{cases} \tag{2.7}$$

- Softmax Function: We use softmax functions to normalize the outputs which are weighted sum values to probabilities that sum up to one. If we were to apply multi level classification where we are classifying more than binary outputs($k > 2$ classes) we use softmax to determine the probability of the membership to each class. In case the output dimension $d_{output} = 1$, softmax is used for regression or binary classification.

$$\begin{gathered} \boldsymbol{z} = z_1, z_2, z_3 ..., z_k \\ softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \end{gathered} \tag{2.8}$$

2.5.2 Training Feed Forward Networks

The Pattern Recognition books by Christopher M. Bishop [16] [17] have been referred for the below section. Readers are recommended to refer these books for in-depth knowledge about the subject. We have train our Neural Network for our learning task. The idea is to use target labels from the dataset to accomplish the task. Our learning algorithm estimates the underlying distribution $p(Y|X)$ where y_i is the target label or output of an input example x_i in the training set D.

$$p_{model}(y_i|x_i;\theta) = \hat{Y_i} \tag{2.9}$$

We would like to have model estimation p_{model} which is closest to $p(Y|X)$ i.e the true distribution of data. For the input $x_1, x_2, x_3...x_m$ with m samples parameterized by θ and respective target labels $y_1, y_2, y_3...y_m$ we want to estimate $\hat{Y_i}$ on unseen data. One of the ways of doing so is by using Maximum Likelihood Estimation(MLE).

$$\begin{gathered}\theta_{MLE} = \operatorname*{argmax}_{\theta} p_{model}(Y|X;\theta)\\ \operatorname*{argmax}_{\theta} \prod_{i=1}^{m} p_{model}(y_i|x_i;\theta)\\ \theta_{MLE} = \operatorname*{argmax}_{\theta} \sum_{i=1}^{m} \log p_{model}(y_i|x_i;\theta)\end{gathered} \tag{2.10}$$

Suppose we use a Gaussian to model the conditional distribution. The conditional log likelihood would look like below.

$$\begin{gathered}\theta_{MLE} = \operatorname*{argmin}_{\theta} -p_{model}(Y|X;\theta)\\ \sum_{i=1}^{m} \log p_{model}(y_i|x_i;\theta) = -m\log\sigma - m/2\log 2\Pi - \sum_{i=1}^{m}(f(x;\theta)-y_i)^2/2\sigma^2\end{gathered} \tag{2.11}$$

We would be only interested in theta hence we can write the equation as

$$\theta_{MLE} = \operatorname*{argmin}_{\theta} \sum_{i=1}^{m}(f(x;\theta)-y_i)^2 \tag{2.12}$$

So in a cost function where $p_{model}(y|x)$ is a gaussian $\mathcal{N}(y; f(x;\theta), I)$, we would like to define a cost function

$$\begin{gathered}J(\theta) = E_{x,y\ \hat{p}} f(x;\theta) - y_i)^2\\ \theta_{MLE} = \operatorname*{argmin}_{\theta} \mathbf{E}_{x_i,y_i\sim\hat{p}_{data}} \mathcal{L}(f(x_i,\theta), y_i)\end{gathered} \tag{2.13}$$

The above generalised loss function is based on pre sample loss based on predicted output and expected output. Now we need to understand that we have no idea about true distribution of the data, hence we work on empirical distribution $\hat{p}_{data}$. Now we will define empirical risk over m training samples. We use a loss function to minimize empirical risk over these training samples. One thing to note is that this idea can be very prone to overfitting as it may try to memorize the training set and perform poorly on unseen data.

$$\mathbf{E}_{x_i,y_i \sim \hat{p}_{data}} \mathcal{L}(f(x_i,\theta),y_i) = 1/m \sum_{i=1}^{m} \mathcal{L}(f(x_i),\theta),y_i) \tag{2.14}$$

Empirical risk over all training samples can be computationally very expensive. Modelling over each value can consume a lot of computational load. Minimizing the loss function over the parameter space does not seem possible. Hence Gradient Descent is used to optimize the loss function. Gradient descent makes us move small steps in the direction opposite to gradient function and reach a minimum. From this we can estimate the local parameters. Over the course of steps to minimums the function can reach a local minimum and those should be avoided.

$$\begin{aligned} \theta_{MLE} &= \operatorname*{argmin}_{\theta} \mathbf{E}_{x_i,y_i \sim \hat{p}_{data}} \mathcal{L}(f(x_i,\theta),y_i) \\ &= \operatorname*{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i,\theta),y_i) \\ &\min_{u,u^T u=1} \quad u^T \nabla_\theta \mathcal{L}(f(x,\theta),y) \end{aligned} \tag{2.15}$$

The directional derivative of $\mathcal{L}$ along u is minimized when u points opposite to direction as $\nabla_\theta \mathcal{L}$. We can decrease L when we move in the opposite direction of negative gradient by method of steepest descent.

$$\theta' = \theta - \eta \nabla_\theta \mathcal{L} \tag{2.16}$$

where, η is the **learning rate** which can also be called the step size. It is more common to evaluate the model parameters only on sampled mini-batches of data. Most optimization algorithms converge faster (overall) using rapid approximations of gradients rather than slower exact gradients. To conclude the algorithm 1 takes a mini batch of m samples, a loss function, a function f with θ as input. It aims to return the best parameters for θ.

2.5.3 Recurrent Neural Network (RNN)

Recurrent Neural Networks are used to handle data that is sequential in nature. These models have previously been used in speech recognition [18] [19] and information extraction. These networks are experts in training sequences. The network allows cyclic connections. The output of previous time slab is fed as

Algorithm 1 Stochastic Gradient Descent

```
    Input: Parameter $\theta$
    Input: Training Set $D = \{(x_1, y_1), (x_2, y_2)....(x_m, y_m)\}$
    Input: Loss function $\mathcal{L}$
1: while stopping criteria not met do
2:     Sample a mini-batch of $m$ examples $\{(x_1, y_1), ..., (x_m, y_m)\}$ and target labels $y_i$
3:     Compute the gradient estimate $\hat{g} = 1/m \nabla_\theta \mathcal{L}(f(x, \theta), y)$
4:     $\theta \leftarrow \theta - \eta \hat{g}$
5: end while
6: return $\theta$
```

input to the next time slab. This in turn helps Network maintain a memory across different times.

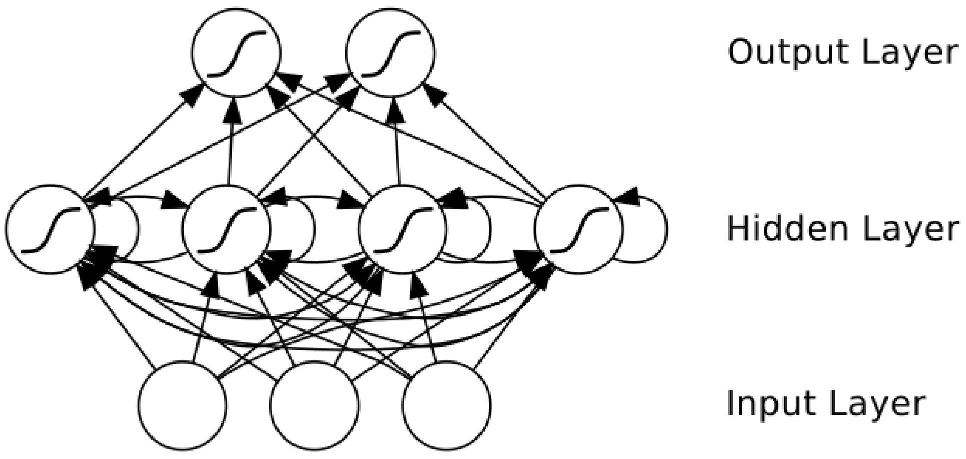


FIGURE 2.8: Recurrent Network from [20]

RNN uses the concept of recursiveness in hidden units.

$$h(t) = f(h(t-1), x(t); \theta) \tag{2.17}$$

The above equation uses folded hidden unit at time t where $x(t)$ is input at time t. $h(t-1)$ is hidden output at time t-1.

$$\begin{gathered} a(t) = Wh(t-1) + Ux(t) + b \\ h(t) = \tanh a(t) \\ o(t) = Vh(t) + c \\ \hat{y}(t) = softmax(o(t)) \end{gathered} \tag{2.18}$$

where, parameters W, U and V are the weight matrices for connections between the two hidden layers over time, input to the hidden layer, and hidden to output layer respectively, and b and c are the bias vectors for the hidden and the output layer respectively. We use hyperbolic tangent function transformation of output in the hidden layer. For output transformation we use softmax function to obtain the output from the model. The final output is $y\hat{(t)}$ for a given timestep t while h(t),h(t-1) are hidden layer outputs from t-1, t time slab. The intermediate results for time t are a(t) and o(t).

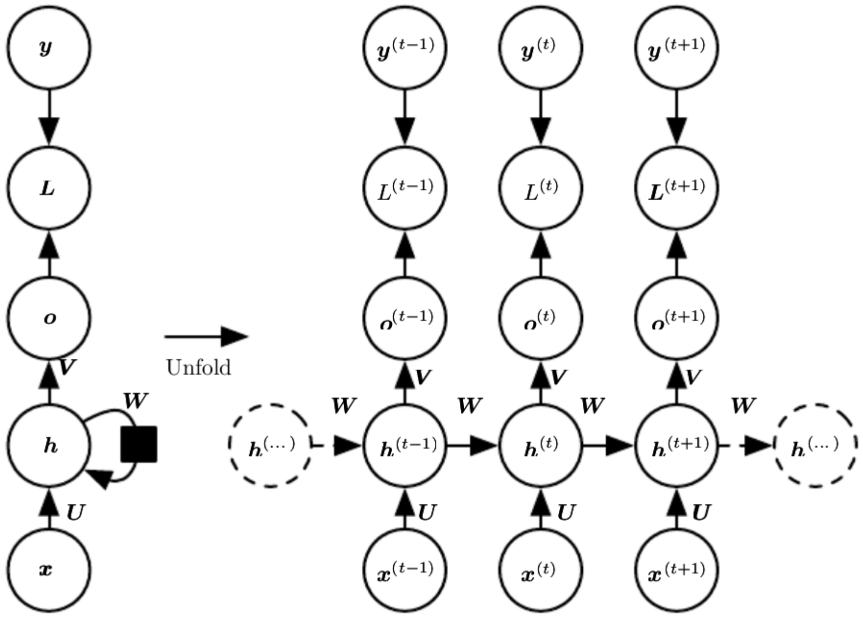


FIGURE 2.9: Unfolded Recurrent Network from [17]

While forward propagation occurs from left to right in case of an unrolled graph. The learning process should occur from right to left using backpropagation but this can computationally expensive. We need to get all past information to use for future transformations. We can use the existing list of training labels and not indulge in parallel training, where gradients for each time step can be computed stand-alone. This is called Teacher Forcing. Teacher forcing is a training technique that is applicable to RNNs that have connections from their output to their hidden states at the next time step. Also we can compute gradient in unrolled RNN using backpropagation like we have done for Feed forward NN.

2.5.4 Attention

Traditional RNN convert all information from a source sentence into a fixed length vector. This is a potential problem for very long sentences. Attention Mechanism takes n arguments $y_1, y_2 ... y_n$ and a context vector c. Combining different parts of y_i it returns z. Attention receive input sentences $a = a_1, a_2 ... a_{l_a}$ and $b = b_1, b_2 ... b_{l_b}$ where a_i,$b_j \in R^d$ is a word embedding vector of dimension d and that each sentence is prepended with a "NULL" token. The training data comes in the form of labeled pairs.

$$\begin{gathered} Trainingdata = \{a^{(n)}, b^{(n)}, y^{(n)}\}_{n=1}^{N} \\ y^N = \{y_1^n, y_2^n ... y_C^n\} \end{gathered} \tag{2.19}$$

From this the model creates input representations $\overline{a} = \overline{a_1}, \overline{a_2} .. \overline{a_{l_a}}$ and $\overline{b} = \overline{b_1}, \overline{b_2} .. \overline{b_{l_b}}$. The vanilla versions of the model defines $a := \overline{a}$ and $b := \overline{b}$. The versions do not consider the word order. From this it creates soft alignment of $\overline{a}$,$\overline{b}$ using variant of neural attention and decompose the problem into the comparison of aligned subphrases. The next step is to compare each of aligned subphrases to produce set of vectors $\{V_{1,i}\}_{i=1}^{l_a}$ for a and $\{V_{2,j}\}_{j=1}^{l_b}$. Each $V_{1,i}$ is a non linear combination of a_i and its softly aligned subphrase in b. In the final step aggregate the sets $\{V_{1,i}\}_{i=1}^{l_a}$ for a and $\{V_{2,j}\}_{j=1}^{l_b}$ for b from the previous step and use the result to predict $\hat{y}$.

2.6 Word and Character Embeddings

Deep Learning algorithms cannot take raw text as input. We need to encode them as numbers. Word Embedding maps words or phrases from vocabulary into vectors or real numbers. The mapping task uses methods like neural networks, dimension reduction in pca. We discuss this section by referring to attribute embedding sections in Deep Matcher [21]. Lets say set of attributes $A \in A_1, A_2, A_3 .. A_j$ have to be converted to word embedding $u_{e_1,j}$, $u_{e_2,j}$. For attribute $A_j \in A$ we denote word embeddings for entity mentions e_1, e_2 as

$$\begin{gathered} u_{e_1,j}, u_{e_2,j} \in R^{d \times m} \\ wordembedding = \{(u_{e_1,j}, u_{e_2,j})\}_{j=1}^{N} \end{gathered} \tag{2.20}$$

Word embedding encodes a word to a fixed d dimensional vector. The technique involves using a lookup table. This table has to be learned or trained using a

network on wikipedia or corpus of the task in hand. This method cannot handle out of vocabulary words. In this aspect character embeddings fare better.
Character embedding takes characters in the word as input and uses neural network to produce a d dimensional representations of the word. Here the output is a trained model and not a lookup table. The idea is that words are made of morphenes or meaningful set of characters of varying lengths. For example, the word “kindness” is made of two morphemes, “kind” and “ness”. This type of embeddings perform better with infrequent words and is more robust to spelling mistakes. Hence character embeddings perform well on entity matching tasks.

Chapter 3

Related Works

In this chapter, we discuss some of the work done related to topic "Entity Matching in Noisy Semistructured Data". This includes some of the link discovery approaches which use embeddings, label match approaches, ontology learning strategies. One thing to note that the literature discussed in this chapter may not be exhaustive but surely covers literature closely related to our topic.

3.1 Introduction

Our problem can be paraphrased in different ways. We can call entity matching in RDF as coreference resolution, record linkage, instance matching in RDFs, link detection, link inference etc under different contexts. You can refer to Figure 3.1. We discuss how these problems have been approached and how they are different from our approaches plus what are the parallels we can draw.

The Linked Data Cloud (LDC) datasets is growing in size and there are many datasets which have to be interlinked. Manual interlinking of such large datasets is nearly not feasible and there is a need to find a way to learn the Interlink resources which are the same via some supervised and unsupervised approaches. These approaches should be optimised, preprocessed and evaluated at a very large scale. One would like to avoid a cartesian product of resource comparison in the LDC because that can be really computationally very expensive. Also an approach which can be run offline would be preferred as these Resource/Entity comparisons can run for a long time. We will discuss some approaches in more detail in the coming sections.

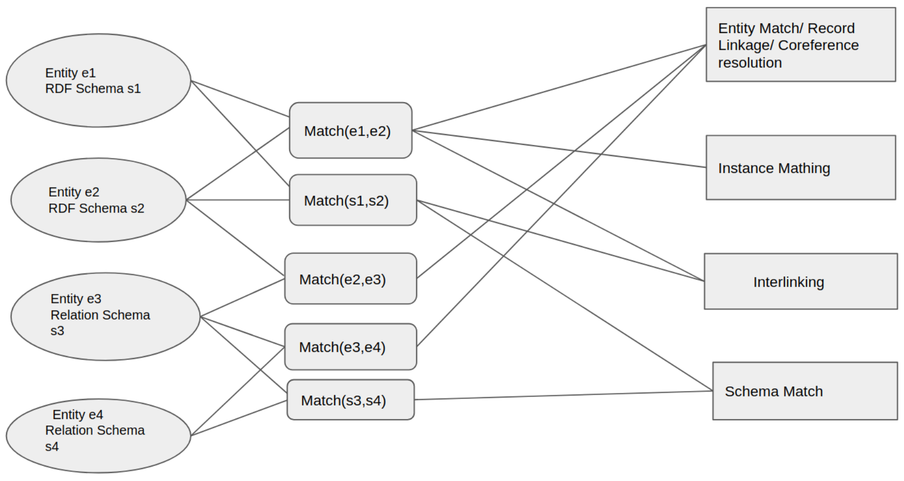


FIGURE 3.1: Entity Matching Terminology

There has been a lot of work done in the field of Link Detection which has been discussed in extensive detail in the paper "A survey of current Link Discovery frameworks" [10]. Some of the content in this section have been taken from that paper and its really recommended that you really go through the content in this reference paper.

3.2 Link Detection Workflow

Link Detection techniques follow an approach which usually involves a standard workflow primarily source and target datasets, a matching algorithm and a set of links. The main phases involved are pre processing, matching stage and post processing.

The target input and output datasets typically should follow RDF/OWL schema because property and schema mappings have to be done seamlessly. The human expert labelling also follow OWL sameAs property to label matches. Link Detection involves computing similarity of resources based on one or more selection criteria. Specifying a linking configuration thus entails the specification of the elements (properties, context) to evaluate as well as the similarity measures to apply (e.g., a 3-gram string similarity, Jaccard similarity for sets or numerical difference) and a way to derive a combined linking decision from the individual similarity values, e.g., based on similarity thresholds to meet [10]. We use value fetch at the literal level as context before applying to a similarity measure using a Deep Learning algorithm. Now coming to the phase of pre processing the

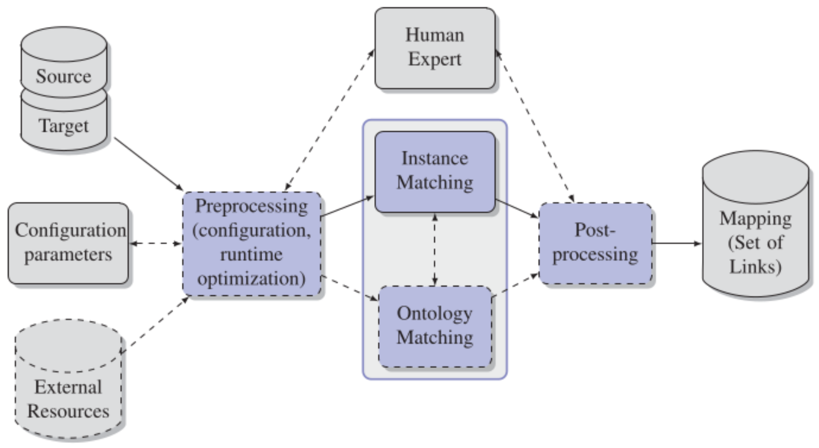
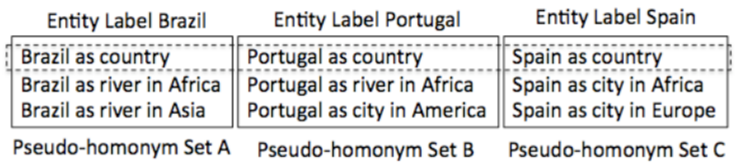


FIGURE 3.2: Link Detection Workflow [10]

methods would strive to reduce the search space for comparisons of resources. We explore only labelled search space(refer section 4.4.1) and do not consider the unlabelled properties. Standard way of reducing search space is blocking and filtering. Blocking partitions the datasets into multiple partitions or blocks such that links are only determined between resources of the same partition. Filtering involves removing pair of records which do not follow similarity conditions. In the stage of Instance matching or Ontology matching the approach can be either structure based or element based where the method studies the literal values. Our approach does not study structure as a whole but uses the structure only to fetch literal values. Structured techniques maybe better because it takes context into account and hence it can consider semantics between resources. The final phase of postprocessing involves tightening loose ends by eliminating inconsistencies mainly using human intervention. Our postprocessing involves manually adding some labels if they have been missed during the transformation phase in the pipeline.

3.3 String Match Approaches

Now we can look at how previously instances from 2 RDF datasets have been matched using string match algorithms. SERIMI [22] deals with interlinking datasets on Linked Data Cloud (LDC). We know that the some of LDC datesets follow RDF schemas and has a large collection of entities bounded by its schema and semantics. SERIMI uses state of the art string matching algorithms at the selection phase and with a function of similarity for approximating the notion of similarity during the disambiguation phase. The approach searches for resources in the target dataset that share the same/similar labels. This will refer to pseudo homonym set. By a naive approach here pseudo-homonym set may have instances of different classes or instances of the same classes sharing same labels. Look at Figure 3.3, In the set A searching by label "Brazil" we get results from class "country" and "river". We get more results from multiple classes for labels "Portugal" and "Spain" too. Hence fetching by just by label match is a bad idea. To solve this issue the paper proposes Resource Description Similarity, or RDS.



Entity Label Brazil	Entity Label Portugal	Entity Label Spain
Brazil as country	Portugal as country	Spain as country
Brazil as river in Africa	Portugal as river in Africa	Spain as city in Africa
Brazil as river in Asia	Portugal as city in America	Spain as city in Europe
Pseudo-homonym Set A	Pseudo-homonym Set B	Pseudo-homonym Set C

FIGURE 3.3: Pseudo Homonym set from [22]

The SERIMI algorithm uses RDF properties and SPARQL to select objects. The selection process involves selecting literals(like Labels) of a fixed size and for comparison uses Jaro Winkler similarity. These similarity measures does not use the knowledge of the RDF schema. The property extraction process is oblivious to the schema and it does not involve tree parsing unlike ours. Jaro Winkler approaches are not robust enough on dirty data or on large text data. The SERMI approach has proved to work on large LDC datasets like DBpedia with good results.
Correference resolution requires efficient use of data sources when there isnt schema mapping involved. If the data sources are big, instance matching has to be made efficient. KnoFuss [23] implements a component-based approach, which allows flexible selection and tuning of methods. It takes the ontological schemata into account to improve the reusability of methods. The task of KnoFuss architecture is to handle data integration process: instance coreferencing, inconsistency detection and inconsistency resolution. It contains method descriptors which are used to perform method selection and assign method parameters.Also there is an application context object which defines the parameters of the method in more specific conditions. Here applicable methods is selected by running the selection criteria queries on the incoming data. Using available selection criteria, context-dependent configuration parameters are defined. Lets say the method used in context has not been used before, a new application context

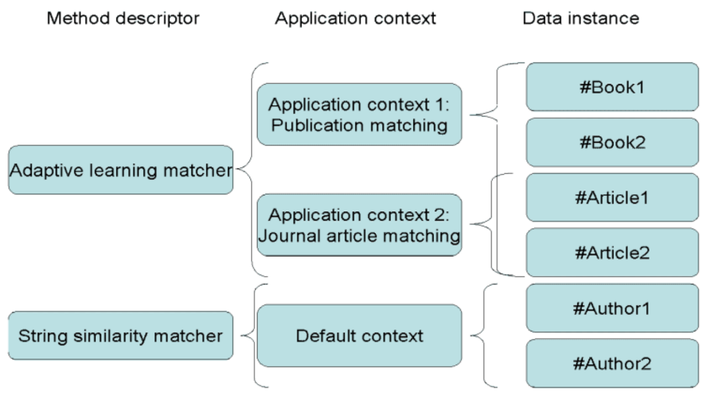

Method descriptor
Application context
Data instance
Adaptive learning matcher
String similarity matcher
Application context 1: Publication matching
Application context 2: Journal article matching
Default context
#Book1
#Book2
#Article1
#Article2
#Author1
#Author2

FIGURE 3.4: KnoFuss architecture via method selection [23]

is defined. In our method ,the class to which entity is mapped is fixed and we do not explore the possibility of matching entities across different classes and hence we do not handle Homonyms. KnoFuss introduces that flexibility. KnoFuss uses traditional string match methods hence may not be robust to dirty data. In Knofuss, the scope for exploring all data properties is less and it can be computationally expensive to do the same. In our method we explore more properties and this can help infer better results. The KnoFuss architecture uses SPARQL wrapper for selection criteria and hence the result retrieval can be really slow as well. In our approach we have minimized the use of the wrapper and hence it the data fetching can be fast. We have additionally tested our approach on LDC datasets which has proved effectiveness on real world datasets and KnoFuss has not been tested on DBpedia and Wikidata datasets.

Our primary dataset is an ITunes music dataset which aims to link two song entities if they are the same. There has also been work done in interlinking Music RDF datasets which is Automatic Interlinking of Music Datasets on the Semantic Web [24]. This method goes for a literal lookup and performs a string match at the query level in SPARQL like SERIMI [22]. At the query level sometimes there is no disambiguation and hence to distinguish further Music Interlinking[24] uses string match of properties to eliminate ambiguity. Our dataset can eliminate ambiguity when there are semantic interoperability conflicts which hasn't been proved in the Music Interlinking [24] and SEREMI [22]. In case we are trying to distinguish "Teriyaki Boys" and "Tokyo Drift" which are basically the same entity. We wont get a match at the query level nor at the string match level which is a visible drawback. This method does not take into account the schema of both datasets and hence is not flexible with varying ontologies. Music Interlinking [24] has to follow a standard schema like OWL or RDF because it uses sparql for literal lookup.

3.4 Schema Based Approaches

Our method relies on pre defined schema mappings for data properties prior to performing Entity Matching in the pipeline. Our approach does not learn these mappings by itself because we manually define it. Now there has been work done to learn these mappings automatically by data integration algorithms [25]. There have been approaches in which the ontologies have been used to match entities. One of the ways to match relational databases has been using DBMS keys. Similar approach has been used in Graph databases too [26].

We need to understand that learning Keys for Graph Databases are much more complex than Relational Databases. Graph patterns are used to identify entities like we use schemas and primary keys in relational databases. The methods like SERIMI we discussed before only rely on label equality but Attribute based Ontological matching [26] take topological patterns into account.

Consider $\varphi 1$ and $\varphi 2$ be two Ontological graph keys(OGK) which represents Pattern 1 and Pattern 2 as represented in Figure 3.4. $\varphi 1$ states that "if two songs share the same name and album, then they refer to the same song". Also for identifying an album we need the name, year of release and artist. Lets look at the other 3 graphs which represent entities v_1, v_2, v_3. Now without considering properties in Ontology O, we can only apply OGK to v_3 as we have all the attributes needed for $\varphi 1$ and $\varphi 2$. However if we study Ontology O we can extend the idea OGK to v_1 and v_2 as we can see Hit is a subclass of song, OST is a subclass of album and band is a subclass of artist. After applying $\varphi 1$ and $\varphi 2$ we can infer now v_1 and v_2 are the same song. Using these OGKs, Ontology 0, Graph G and a matching cost, it defines the Entity Matching Problem using an algorithm chase.

Now there are some similarities with our approach where we consider the Ontology or a subset of it for referring to the properties which is also graph pattern like an OGK. We parse these patterns to get values for properties. The difference however is we do not consider multiple patterns of the graphs for identifying entities.

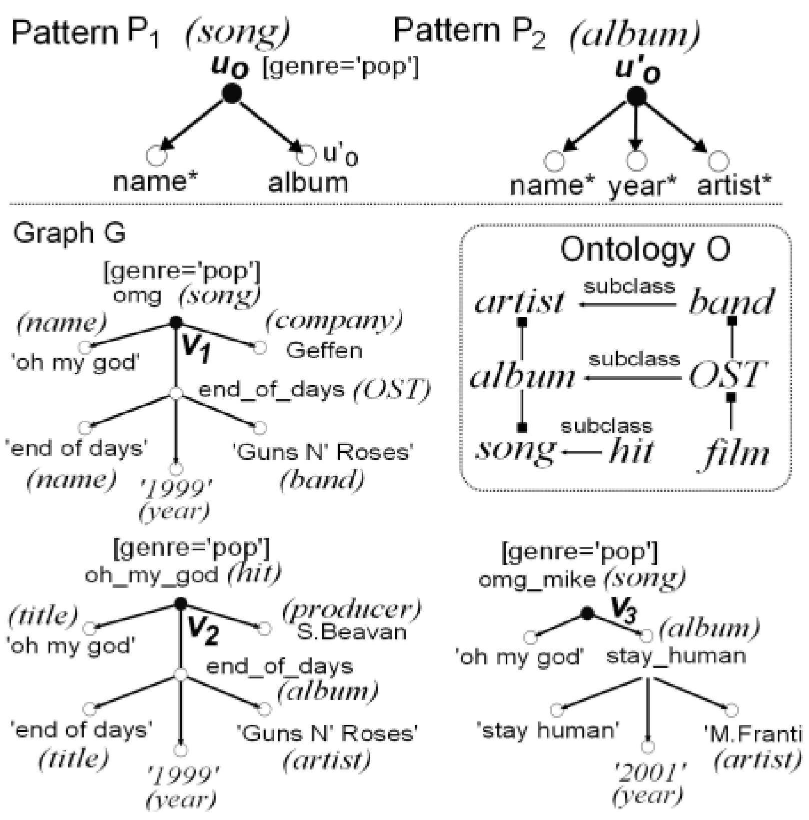


Figure 3.5: Ontology Graph key for Ontological Matching [26]

Computing Class/Node similarity and property similarity help in seeing if individuals refer to the same entity. In order to map similar properties at the schema level, we do it manually in our approach using equivalent property. One of the ways to find node similarity is by using a lowest common ancestor(LCA) for a pair of nodes. Hence computing Taxonomic similarity is one of the methods for Entity Matching. Pathsim [27] is a method which uses Lowest Common Ancestor and Taxonomic relations. According to PathSim, the similarity between two nodes is directly proportional to the amount of paths that meet the meta-path description among them. ”A Framework for Semantic Similarity Measures to enhance Knowledge Graph Quality” [28] presents an ontological similarity measure OnSim based on hierarchy and neighborhood. But plain taxonomic ontology based methods ignores semantics as well as the values for properties. The thesis [28] also presents IC-OnSim which takes ontology, neighborhood plus shared information between the nodes for checking similarity. If the nodes are annotated with an article or description it compares the text corpora while determining similarity score for the nodes. Finally the thesis presents GADES architecture which determines similarity of nodes based on taxonomy, shared information, neighbors and information of attributes. Attribute similarity involves comparing texts in data properties using various distance measures. In our approach attribute comparison, text corporas are all compared using Deep Matcher [21] which has proven to perform really well as compared to conventional string match approaches.

3.5 Embedding based Approaches

In the approaches discussed till now we have not taken into account different representations for Knowledge graphs like KG embeddings. Latent feature methods can be used for knowledge graph analysis. They serve as convenient representations for learning knowledge graphs which can in turn be used for Link prediction. LiteralE [29] incorporates literals into Knowledge Graph embeddings. LiteralE directly enriches these embeddings with information from literals via a learnable parametrized function.

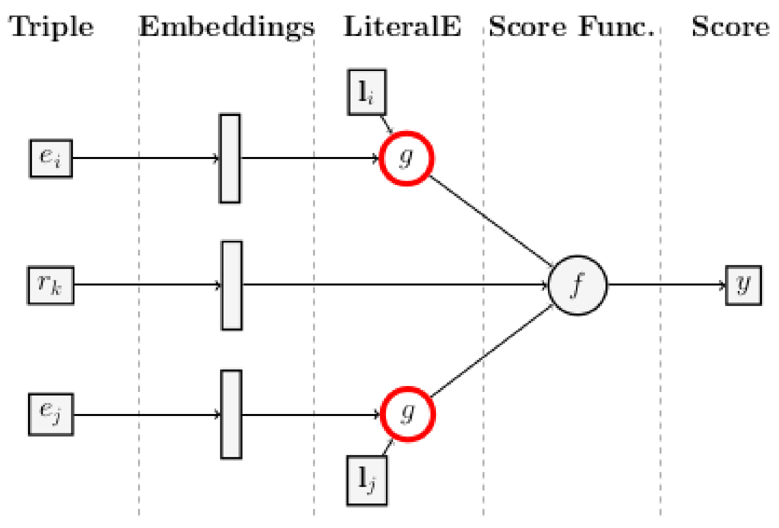


FIGURE 3.6: Overview on how LiteralE is applied to the base scoring function f . LiteralE takes the embedding and the corresponding literals as input, and combines them via a learnable function g. The output is a joint embedding which is further used in the score function f [29].

Latent methods represent entities and relations of a knowledge graph using low dimensional vectors. LiteralE follows a score based approach unlike other latent methods incorporating literals in knowledge graphs. To put a summary LiteralE is a method which takes entity embedding, literal vector as an input and maps them to a vector of entity embedding dimension [29]. There arent many parallels of this method to our approach except for the fact that LiteralE also takes literals into account for link detection. Also the method does not just learn similarity function to compare similarity of entities, it also can infer other links including ”sameAs” which can help in completing the knowledge graph.

3.6 Conclusion

This section has given an overview of the standard ways to approach an entity matching problem in knowledge graphs plus the ways to infer links as well. SEREMI [22], KnoFuss [23], Automatic Interlinking of Music Datasets on the Semantic Web [24] have all used string similarity approach following a literal lookup at the atomic levels. Attribute based ontological matching [26] and literalE [29] have learned the structure of the graph plus literals can detect and infer hidden links. GADES [28] uses attribute based, structure based and taxonomy based approach .These methods can also help detecting same entities and in way performing deduplication. Our approach follows in turn follows a mix of these approaches where we also study literals by parsing through the structure of the graph. Our notable contributions include handling semantic interoperability conflicts, considering long literal attributes which can include dirty data, handling empty attributes.

Chapter 4

Entity Matching Pipeline

We now enter the core section of the thesis, where we describe our Entity Matching approach on RDF datasets which have semantic interoperability conflicts. Some of the concepts used in this section have been discussed in the background chapter, and we may not elaborate on them in detail here. We start with an overview of the pipeline, and next in each section, we will discuss the different parts of the pipeline, which will lead us to solve our Entity Matching Problem.

4.1 Pipeline

Our proposed pipeline would take as input a pair of annotated knowledge graphs and predict match or no match for a pair of entities belonging to its respective knowledge graphs where the class to which the entity belongs is fixed. The annotated knowledge graph after conflict resolution is transformed to a dataframe where each row vector represents values taken for each attribute for a pair of entities and column vector indicates values taken for an attribute by all entities. Using this dataframe for each row vector a similarity representation is created by representing similarity score for a given attribute which is repeated for each attribute for all the entity pairs. The similarity representation matrix merged with label vector is passed to a classifier which performs label prediction. The Figure 4.1 is a pictorial representation of the pipeline.

Lets refer to Section 1.2 where we have discussed the problem statement. Our input graphs $G1$ and $G2$ are left and right RDF graphs in our pipeline in Figure 4.1. The conflict handling phase are done independently for both graphs as we see in the problem statement using resolve function. Following the conflict handling phase we may have new relations which has to be mapped because some of the conflicts like representation conflicts are resolved by adding new properties. Unless we map new properties we add in conflict handling phase, they wont be considered. Next phase of the pipeline is the transformation function where we transform RDF data to a dataframe. The column vector for the dataframe is created using mapped properties. Next Data vector for dataframe is created using graph parsers which fetches values for these properties. The dataframe is later passed onto the Deep Learning model which learns the values for properties and returns the match or no match labels. Going by the KnoFuss workflow discussed in section 3.2 we can categorize input phase plus conflict handling phase as a part of Pre processing, Transformation and Deep Learning Model as part of instance matching. We have not explicitly mentioned post processing phase in the pipeline but that would correspond to handling inconsistencies in our evaluation section.

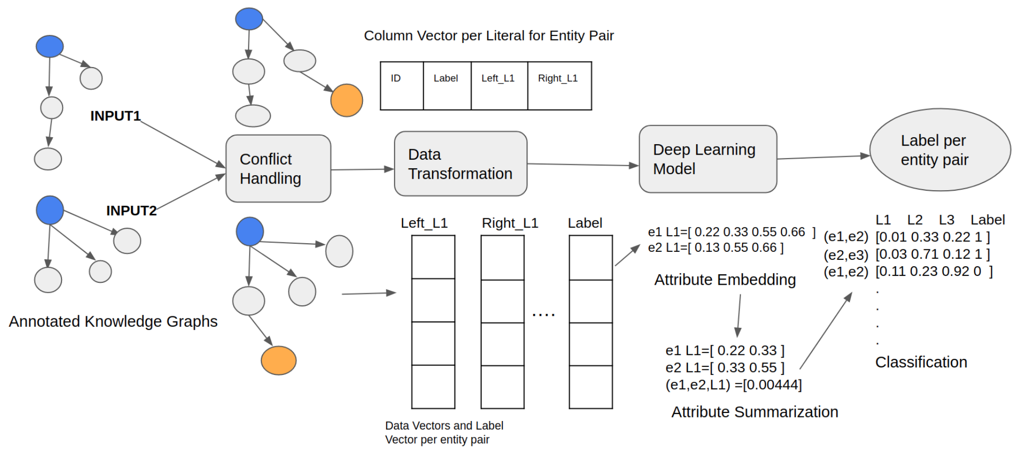


FIGURE 4.1: Entity Matching Pipeline: We propose a three-step pipeline where the first step fixes semantic conflicts present in the inputs. Second, we transform the input data into a vector representation to finally apply a deep learning model that summarizes the attributes to predict entity matches.

4.2 Preprocessing

We discuss both input Knowledge Graphs in this section which follow the RDF/OWL schema. Our Input (Right and Left RDF graphs) is a set of triples consisting of subject predicate objects. The graph consists of a schema which acts like a skeleton of the whole graph and is called an Ontology. The most important aspect in this section is pre processing where we need to have a schema in place for our transformation step in the pipeline.

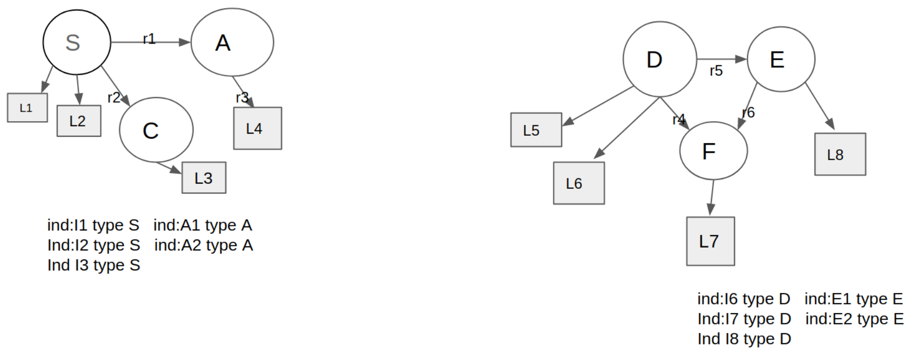

S
r1
A
L1
L2
r2
C
r3
L4
L3
ind:I1 type S ind:A1 type A
Ind:I2 type S ind:A2 type A
Ind I3 type S
D
r5
E
L5
r4
r6
F
L8
L6
L7
ind:I6 type D ind:E1 type E
Ind:I7 type D ind:E2 type E
Ind I8 type D

FIGURE 4.2: Sample Left Right Ontology

Lets consider Figure 4.2 which are 2 RDF graphs(R and L) which have classes $L_{class} = \{S, A, C\}$, $R_{class} = \{D, E, F\}$. We have object properties $L_{object} = \{r1, r2, r3\}$, $R_{object} = \{r4, r5, r6\}$ and data properties $L_{literal} = \{L1, L2, L3, L4\}$, $R_{literal} = \{L5, L6, L7, L8\}$. From Figure 4.2 we have individuals $L_{individual} = \{I1, I2, I3, A1, A2\}$ and $R_{individual} = \{I6, I7, I8, E1, E2\}$. We need to map the individuals to sameAs or differentFrom which can also be called $labels = \{1, 0\}$ where 1 indicates sameAs, 0 indicates differentFrom and this forms our classification problem. For any classification problem we need data i.e more information on individuals. Below we describe a sample Input to our pipeline without semantic conflicts. We have some information on individuals which are label and literal values which we discuss next.

$I1_{triples}$ represents triples set of individual I1, $I2_{triples}$ represents triples set of individual I2 and so on. Label set represents individuals represents same and different entities. Using the below data can we infer I3 sameAs I8? Are I3 and I8 representing same entity? We can see I3 has literal value “escape” for L1, “Jan 11, 1999” for L2. For I8 we see the literal L5 has value as “escape” too, “Jan 11, 1999” for L6 which is same for I3. We see the same pattern for the other pairs of individuals. The more data we have, the more we can infer. How do we know which properties should we consider? We should define these properties before hand which we call property mappings. We need to define property equivalence, L2 is equivalent to L5, L3 equivalent L6 etc. If we do not define equivalent property and classes, We may have to force a cartesian product of comparisons across entities of all classes and check all properties. This maybe really inefficient as discussed in KnoFuss architecture [23].

Example 1:

$$\begin{aligned}
L_{class} &= \{S, A, C\}\\
R_{class} &= \{D, E, F\}\\
L_{object} &= \{r1, r2, r3\}\\
R_{object} &= \{r4, r5, r6\}\\
L_{literal} &= \{L1, L2, L3, L4\}\\
R_{literal} &= \{L5, L6, L7, L8\}\\
L_{individual} &= \{I1, I2, I3, A1, A2\}\\
R_{individual} &= \{I6, I7, I8, E1, E2\}\\
I1_{triples} &= \{(I1, L1, “beat\ it''), (I1, L2, “Jan16, 2016''), (I1, r1, A1), (A1, L4, “Thriller'')\}\\
I6_{triples} &= \{(I6, L5, “Beat\ it''), (I6, L6, “Jan16, 2016''), (I6, r5, E1), (E1, L8, “Thriller'')\}\\
I2_{triples} &= \{(I2, L1, “Hero''), (I2, L2, “Jan11, 1999''), (I1, r1, A2), (A1, L4, “Escape'')\}\\
I7_{triples} &= \{(I7, L5, “Heroo''), (I7, L6, “Jan11, 1999''), (I7, r5, E2), (E2, L8, “escape'')\}\\
I3_{triples} &= \{(I3, L1, “escape''), (I3, L2, “Jan11, 1999''), (I3, r1, A2)\}\\
I8_{triples} &= \{(I8, L5, “escape''), (I8, L6, “Jan11, 1999''), (I8, r5, E2)\}\\
Label &= \{(I1, sameAs, I6), (I2, sameAs, I7),\\
&\quad (I1, differentFrom, I8), (I3, differentFrom, I6)\}
\end{aligned}$$

4.3 Conflict Handling

When we talk of Linked Data Cloud we have to understand the inconsistencies in formats, notations and structures across all open RDF datasets. When there are conflicts in these notations, it would affect our Entity Matching tasks. It is important that we identify what semantic interoperability conflicts (2.4) we have at the schema level or at the data level. We discuss how we have handled some conflicts we have mentioned in 2.4 to solve our task.

4.3.1 Identifying Conflicts

In the figure 4.3 we see 3 different conflicts. The first kind is of the type GRANULARITY. We see that L2 is of type string with value 20 kgs and also with value 20,000 grams. The base algorithm we use for comparing literals or even a raw string match algorithm cannot infer if they belong to the same individual using that property. On the other hand if one of the value was string and other was integer, it would be a SCHEMATIC conflict. A SCHEMATIC conflict would make comparisons difficult. Now to further extend if one individual would have been 20kg while the other 44 pounds, it would be a REPRESENTATION conflict. In the second example in figure 4.3 there are 2 different date formats which can be

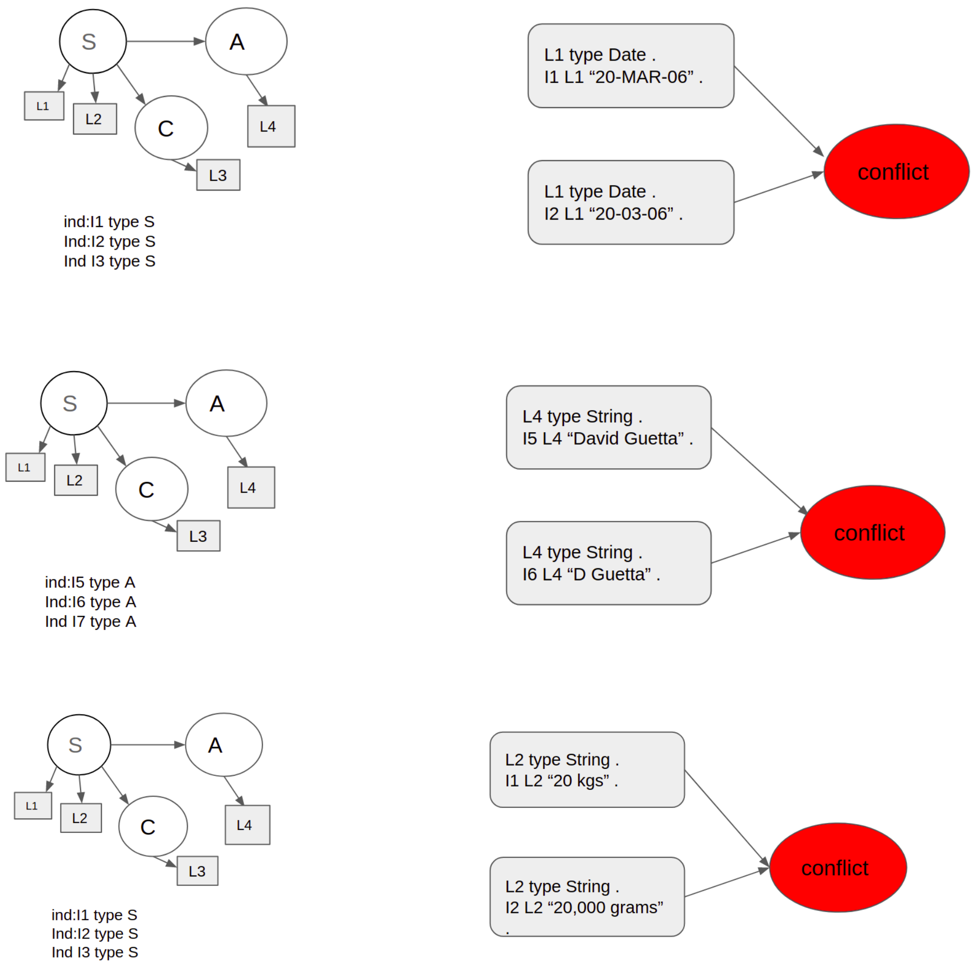


FIGURE 4.3: Semantic Interoperability Conflicts

categorised as REPRESENTATION conflict. Finally the third conflict in figure 4.3 would be identified as a DOMAIN conflict. Conflict identification is on the whole a manual task in our approach.

4.3.2 Resolving Conflicts

Now in order to resolve the conflicts we use sparql to fetch, modify,alter the literal values. The task is again to minimize comparison across individuals (avoid cartesian product) to resolve conflicts.

Lets look at the example below with individuals I8,I9. Lets first define mappings where we say L1 is equivalent to L5, L2 is equivalent to L6, L3 is equivalent to L7 and finally L4 is equivalent to L8. Now L2 and L6 follow different date formats. We standardize the format using a function $\varphi I*_{REPRESENTATION}$

because its a representative conflict. Now the entity matching algorithm can compare L2 and L6 better because the date format is the same. Lets consider L4 and L8. Both represent price of the album. They are using different currency formats. This representative conflict is again resolved using the function $\varphi I*_{REPRESENTATION}$. As you can see in $I8_{resolved}$ and $I9_{resolved}$ we add a new resource a2 in first dataset and b2 in second dataset. We split the price(a2,b2) into currency and amount as literals for both datasets. Now we can use the currency conversion function to standardise the currency to Euro. After this we can compare the prices better in our approach after replacing the price literal with currency and amount. Coming to the L1 and L5 which has values "Pigs" and "Pigman". We are not sure if it refers to the same resource. But if we observe its name attribute and both belong to the same album and has same release date. We define a predicate that if a set of resources have same album name and release date it maybe the same song and we standardise value for song name. We need to minimise comparisons here and hence we only consider strings belonging to same album and those which have same release date. On this subset we perform string match to check for partial string match and see if its an acronym or an abbreviation or just another synonym. If yes the values are replaced and standardised. All of these are defined using the function $\varphi I*_{DOMAIN}$. Now we have a modified set of triples as we see in $I8_{resolved}$ and $I9_{resolved}$.

Example 2:

$$I8_{conflicts} = \{(I8, L1, ''Pigs''), (I8, L2, ''21-01-76''), (I8, r1, a1),$$
$$(a1, L3, ``Animals''), (a1, L4, ``EUR1'')\}$$
$$I9_{conflicts} = \{(I9, L5, ``Pig\ man''), (I9, L6, ``01-21-76''), (I9, r5, b1),$$
$$(b1, L7, ``animals''), (b1, L8, ``Rs.88'')\}$$
$$I_{resolve} = \{\varphi I*_{DOMAIN}, \varphi I*_{REPRESENTATION}\}$$
$$\varphi I*_{REPRESENTATION}((I8, L2, ``21-01-76''), (I9, L6, ``01-21-76''))$$
$$\varphi I*_{REPRESENTATION}((a1, L4, ``EUR1''), (b1, L8, ``Rs.88''))$$
$$\varphi I*_{DOMAIN}((I8, L1, ``Pigs``), (I9, L5, ``Pigman''))$$
$$I8_{resolved} = \{(I8, L1, ``Pigs''), (I8, L2, ``21-Jan-76''), (I8, r1, a1), (I8, r2, 12)$$
$$(a1, L3, ``Animals''), (a2, L41, ``EUR''), (a2, L42, ``1'')\}$$
$$I9_{resolved} = \{(I9, L5, ``Pigs''), (I9, L6, ``21-Jan-76''), (I9, r5, b1), (I9, r6, b2)$$
$$(b1, L7, ``animals''), (b2, L81, ``EUR''), (b2, L82, ``1'')\}$$

4.4 Data Transformation

This section mainly discusses how we fetch the preprocessed data from the previous phase in the pipeline. The aim in this phase is to have the data in the form desired for the Deep Learning algorithm. We recommend you to go through the documentation of Deep matcher [30] to learn about the input format desired by our deep learning model. The rdf data in turtle format returned by the previous phase is far from what Deep Matcher wants. Figure 4.4 represents an instance which is an input pair we are sending to the transformation phase. Left graph is for left entity and right graph is for right entity.

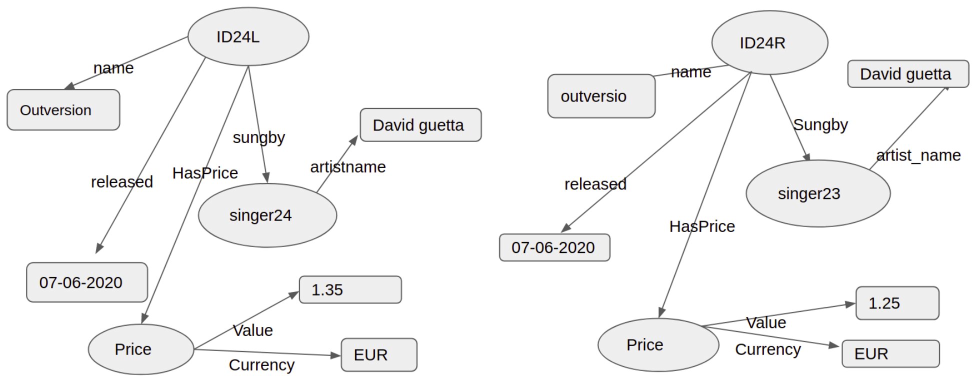


FIGURE 4.4: Pair of input to transformation

4.4.1 Column Vector

We need to first get the column vector in the form our deep matcher algorithm wants. We need an Id vector which can be auto generated per pair of entities/individuals. There is also a label attribute which considers labels i.e 1 for a match or 0 for no match. Coming to the rest we consider a pair of equivalent properties. The left schema properties need to have a prefix left_ and right schema properties need to have a prefix right_ .The name of the property can be any one of the labels of left or right schema. i.e if we have $(L1, equiprop, L5)$, the column vector will be $(left\_L1, right\_L1)$. Lets look at the example below we have set of literals in $L_{literal}$, set of mappings Map and a function $\varphi Map\_Prop(L_{literal}, Map, Column)$ which returns column vector col.

Example 3:

$Column = \{id, label\}$
$L_{literal} = \{L1, L2, L3, L4\}, R_{literal} = \{L5, L6, L7, L8\},$
$Map = \{(L1, equiprop, L5), (L2, equiprop, L6), (L3, equiprop, L7), (L4, equiprop, L8)\}$
$Column = \varphi Map\_Prop(L_{literal}, Map, Column)$
$Column = \{id, label, left\_L1, left\_L2, left\_L3, left\_L4, right\_L1, right\_L2, right\_L3, right\_L4\}$

4.4.2 Data Vectors

Now we have the column vector by the method discussed in the previous subsection. Next step would be to fetch values from literals for the resources. The transformation should be in desired format for Deep Matcher [30]. The obvious way to fetch data is using a SPARQL wrapper. But a sparql fetch is not an efficient method as each fetch consumes a lot of time. An alternative we use is Graph parse method. We will discuss how we parse to the literal using graph parse methods like Depth first search. Our input is a column vector and RDF dataset with resolved conflicts. Prerequisite is also a mapping to pair of equivalent properties which we consider for our deep matcher algorithm. The output will be a data vector for all the columns we consider.

Below we present a sample fetch function of data for individuals. We take data from example 1 and column vector from example 2.We discuss the transformation step with data vector creation in detail. Primarily we have the fetch function which takes the individual and literal to be fetched as parameters. It parses the graph and fetches the value for the literal. We do this we for all individuals. If literal belongs to left schema we append the fetched value in the left_literal vector. If literal belongs to right schema, we find the equivalent literal from left schema and populate it in the right_literal.In case of a label vector, for pair of individuals if the relation is sameAs, its mapped to 1 and 0 otherwise. By this we have literal vectors for each literals and label vectors. By merging all vectors into dataframe our data is ready for Deep Matcher algorithm. To illustrate further,lets say we want to fetch the literal L1 value for I1 as we have done in Example 4 we get “Beat it”. Similarly if we have to fetch literal L5 value for I6 we also get “Beat it”. We have equivalent property vector from example 3 where we know L1 and L5 are equivalent. L1 is mapped to left_L1 vector and L5 to right_L1 vector. We repeat this for each pair in labels. After each vector is populated we merge it to a dataframe which is later passed to deep matcher.

Example 4:

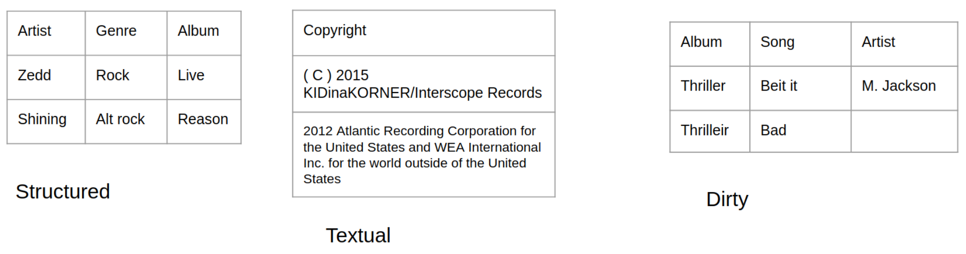
$fetch(I1, L1) = \text{“}Beat\ it''\ fetch(I1, L2) = \text{“}Jan\ 16, 2016''\ fetch(I1, L4) = \text{“}Thriller''$
$fetch(I6, L5) = \text{“}Beat\ it''\ fetch(I6, L6) = \text{“}Jan\ 16, 2016''\ fetch(I6, L8) = \text{“}Thriller''$
$fetch(I2, L1) = \text{“}Hero''\ fetch(I2, L2) = \text{“}Jan\ 11, 1999''\ fetch(I2, L4) = \text{“}Escape''$
$fetch(I7, L5) = \text{“}Heroo''\ fetch(I7, L6) = \text{“}Jan\ 11, 1999''\ fetch(I7, L8) = \text{“}escape''$
$fetch(I3, L1) = \text{“}escape''\ fetch(I3, L2) = \text{“}Jan\ 11, 1999''\ fetch(I3, L4) = \text{“}Escape''$
$fetch(I8, L5) = \text{“}escape''\ fetch(I8, L6) = \text{“}Jan\ 11, 1999''\ fetch(I8, L8) = \text{“}escape''$
$Label(I1, I6) = 1\ Label(I2, I7) = 1\ Label(I1, I8) = 0\ Label(I3, I6) = 0$
$left\_L1 = \text{“}Beat\ it'', \text{“}Hero'', \text{“}Beat\ it'', \text{“}escape''$
$right\_L1 = \text{“}Beat\ it'', \text{“}Heroo'', \text{“}escape'', \text{“}Beat\ it''$
$left\_L2 = \text{“}Jan\ 16, 2016'', \text{“}Jan\ 11, 1999'', \text{“}Jan\ 16, 2016'', \text{“}Jan\ 11, 1999''$
$right\_L2 = \text{“}Jan\ 16, 2016'', \text{“}Jan\ 11, 1999'', \text{“}Jan\ 11, 1999'', \text{“}Jan\ 16, 1999''$
$left\_L4 = \text{“}Thriller'', \text{“}Escape'', \text{“}Thriller'', \text{“}Escape''$
$right\_L4 = \text{“}Thriller'', \text{“}Escape'', \text{“}escape'', \text{“}Thriller''$
$label = 1, 1, 0, 0$
$T = merge(id, label, left\_L1, right\_L1, left\_L2, right\_L2, left\_L4, right\_L4)$

4.5 Deep Learning Model

In this phase we enter the last step of the pipeline. In this document we have often referred to this Deep Learning Model phase as Deep Matcher. However we can use different Deep Learning Models like Magellan [31]. The Deep Learning model is expected to be robust to dirty, noisy, long textual data. Hence choosing a good model is critical to our Entity Matching Task. This section discusses Deep Matcher [21] in some detail.

4.5.1 General Architecture

The Deep Learning Model is expected to take data from properties of literals for a pair of individuals and predict if they refer to the same individuals. Even a well structured text input alone cannot be read by a Deep Learning Model. First we need to learn character or word embeddings for text representations in a vector format. These embeddings can be trained or be pre trained on an external text corpora . The embeddings should be able to represent large sequences of text. The word embeddings vectors can be of varying dimensions and hence it should have a good function for dimensional reduction and summarization. The summarized representation helps deduce a similarity representation indicating similarity score for an attribute per entity pair. Lastly the Deep Learning Model needs a good classifier to determine for a set of similarity representations if the entity pair is similar or not.



Artist	Genre	Album
Zedd	Rock	Live
Shining	Alt rock	Reason

Structured

Copyright
(C) 2015 KIDinaKORNER/Interscope Records
2012 Atlantic Recording Corporation for the United States and WEA International Inc. for the world outside of the United States

Textual

Album	Song	Artist
Thriller	Beit it	M. Jackson
Thrilleir	Bad	

Dirty

FIGURE 4.5: Entity Matching problem types

4.5.2 Deep Matcher Motivation

Traditional string match methods give priority to characters at the beginning of the string. If there are two strings by name “Michael M Jackson” and “Jackson Michael”, conventional string match algorithms will probably flag a mismatch in strings. Deep Matcher(Hybrid or Attention) would take one of the strings as context and do a character comparison from all direction using an RNN. Deep matcher has proved to work really well on large textual data, dirty data and structured data. Figure 4.5 shows a few cases relevant to our experiment. After the transformation phase we get a copyright vector with a lot of text data. We can also get other vectors with empty or noisy values. In the ideal case each vector is complete and data is structured. However incase of a structured data, because deep matcher needs really high training time and use of such complex model may seem unnecessary. Our ITunes dataset has a copyright field which contains large text and some noisy data. Also wikidata contains fields with noisy data and empty data. While we have handled most conflicts, our conflict handling has its limitations especially when there is a Domain conflict. Keeping these conflicts in mind we opt for Deep Matcher because of its performance when there are such anomalies.

4.5.3 Deep Matcher Architecture

Prior to this we have already discussed some aspects of Deep Matcher in the background chapter and it is recommended you revisit the Neural Network, other

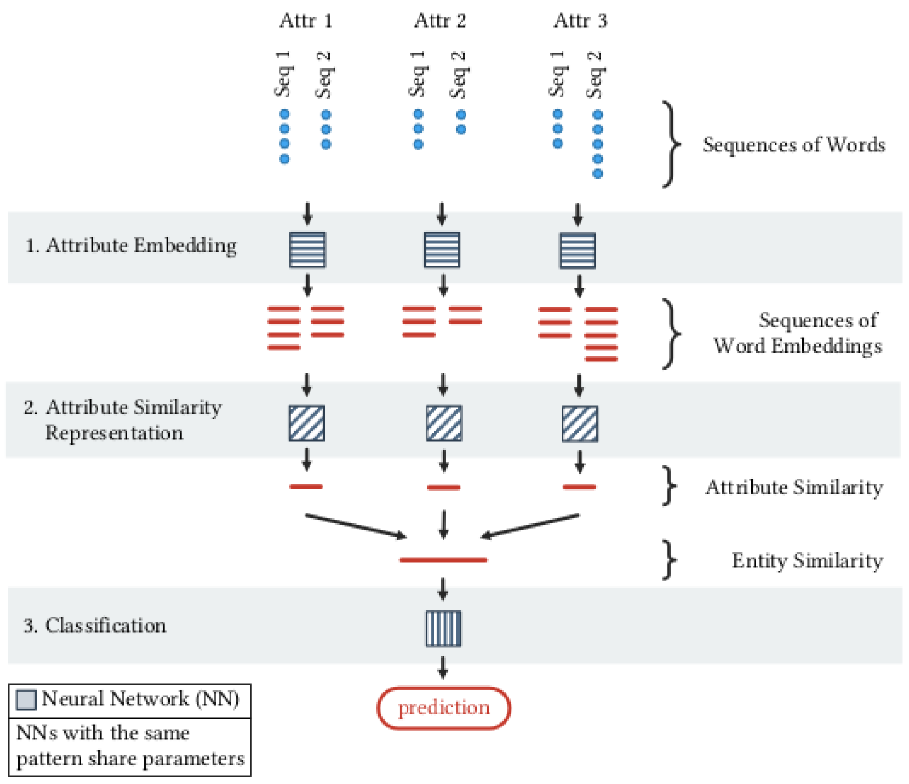


FIGURE 4.6: Deep Matcher Architecture [21]

deep learning sections prior to going through this subsection. We have also discussed the general architecture expected from the Deep Learning model in section 4.5.1. In the Figure 4.6 we show the architecture of Deep Matcher. Because we cannot promise an exhaustive coverage of Deep Matcher in this section, we recommend you to refer the paper "Deep learning for entity matching: A design space exploration" [21] and its implementation documentation [30]. In the previous step we have data vectors all consolidated into a dataframe. We will in this section discuss the architecture in detail and how it is used in our task. Each row of the dataframe represents pair of entities with its attributes along with label and id. For each attribute we have two sequences of text , one for each entity as we can see in the figure 4.6 . Here each attribute with respect to our approach refers to properties we have fetched in the transformation phase.

We know raw sequences of text cannot be understood by the Deep Learning algorithms. The Attribute embedding module takes a sequence of words and returns word embeddings. This embedding represents word vector in a high dimensional vector space. For attributes $A_j \in A_1, A_2, A_3...A_N$, given word vectors for A_j , $w_{e_1,j}$ and $w_{e_2,j}$. It converts them to d dimensional word embeddings $u_{e_1,j}$ and $u_{e_2,j}$. If word sequence for e1 for A_j contains m elements, then we have $u_{e_1,j} \in R^{d \times m}$.Important to note both word sequence can have varying number of elements. The choice of word embeddings include word or character embeddings with option of pre trained or learned embeddings. Deep Matcher uses character embeddings which are pre trained(Fasttext).

Now we have two embedding vectors for each attribute, we send it to the attribute similarity phase which returns attribute similarity representations. The representation should be able to capture the attribute similarities. We can divide this phase into two parts. The first part would be attribute summarization and the second part would be attribute comparison. The attribute summarization phase takes pair of word embedding vectors and would summarize the information of both sequences with summarization vector(both vectors being of the same dimension). For $u_{e_1,j}$ and $u_{e_2,j}$ we get $u_{e_1,j} \in R^{d \times m}$ and $u_{e_2,j} \in R^{d \times k}$ i.e word embedding vectors for an attribute with sequences of length m and k. From here we have summarization vectors $s_{e_1,j} \in R^h$ and $s_{e_2,j} \in R^h$. Next we pass these vectors to comparison phase where it takes the pair of summary representation vectors , passes it to a similarity function and returns one similarity representation for a pair of entities per attribute. i.e for $s_{e_1,j} \in R^h$ and $s_{e_2,j} \in R^h$ we get $s_j \in R^l$.

There are four options for attribute summarizations. The first one is an aggregate function which uses a simple weighted average or an average function to produce a d dimensional summarization. The second method is sequence aware summarization which can help learn complex interaction among tokens. This method uses an RNN that takes order and semantics of the given sequence. The input sequence of embedding is passed into the summarization function to obtain a sequence $h_{e,j}$ hidden states. The hidden states are then aggregated into h dimensional vector to get $s_{e,j}$. Now we move onto the next type of method called sequence alignment which takes two word sequences together while using one as context. The previous methods did not take word sequences together while creating a summary vector. Sequence alignment computes a soft alignment between two given sequences of words and then performs word by word comparison. Sequence alignment cannot take position of tokens into account and only weighs on the context provided as input. To handle the drawbacks of the three methods, the fourth and final method hybrid is proposed. This takes both sequence aware and sequence alignment together which helps accuracy on dirty textual data. Hybrid training is expensive and can lead to high training times.

Finally similarity representations for each attribute given pair of entities we pass it to a classifier module. The classifier using the representations checks if the two entities are the same or not.

4.5.4 Representative solutions for Deep Matcher

Deep matcher offers four types of solutions for Entity Matching which we discuss in the 4 subsections below. All these solutions are completely referred from Deep Matcher paper [21] .

- **SIF: An Aggregate Function Model** SIF uses weighted sum of word embeddings for summarization and element-wise absolute difference for comparison. The weights used to compute average over word embeddings is weighted by a weight $f(w) = \frac{a}{(a+p(w))}$ where a is a hyperparameter and $p(w)$ the normalized unigram frequency of word w in the input corpus.

- **RNN: A Sequence-aware Model** This type of method uses a bi directional RNN for for attribute summarization and an element-wise absolute difference comparison operation to form the input to the classifier module. Precisely bidirectional GRU-based RNN model is used. The forward RNN processes the input word embedding sequence in its regular order and the backwards network that processes the input sequence in reverse. The final attribute summarization representation corresponds to the concatenation of the last two outputs of the bidirectional RNN. This method helps find what sequences are present in the input sequence

- **Attention: A Sequence Alignment Model** Here Decomposable attention is used for attribute summarization and vector concatenation to perform attribute comparison. This method analyses input sequences jointly to learn similarity representation. Intuitively it performs soft alignment and pairwise token comparison across the two input sequences. Attention tries to understand sequences present in both entities together.

- **Hybrid** This model uses a bidirectional RNN with decomposable attention to implement attribute summarization and a vector concatenation augmented with element-wise absolute difference during attribute comparison to form the input to the classifier module. This is the model with the highest representational power we consider in deep matcher [21].

Algorithm 2 Entity Matching algorithm for RDF

```
Input: Graph G1, G2, Labels, Mappings
Output: ${1, 0}$
1: $Column\_Vector \leftarrow get\_col\_vector(G1, G2, Mappings)$
2: $T \leftarrow Column\_Vector$
3: for all $i1, i2 \in Labels$ do
4:   if i1 sameAs i2 then
5:     $T[, label] = 1$
6:   else if i1 differentFrom i2 then
7:     $T[, label] = 0$
8:   end if
9:   for all $property\ p \in G1$ do
10:    if $p \in Mappings$ then
11:      $T[, left\_p] = DFS(i1, p)$
12:      $T[, Right\_p] = DFS(i2, equivalent\_property(p))$
13:    end if
14:  end for
15: end for
16: $Output = DEEP\_MATCHER(T)$
```

To apply these solutions to our transformed data, we can choose the appropriate solution based on the type of data we get from literals. If we have too many attributes with large text, hybrid is worth the training time expense. If there are many semantic interoperability conflicts left unhandled , especially domain or representative its recommended to go for the attention or hybrid models as it best handles reversed string, acronym. In case of dirty/noisy attributes data SIF is not recommended. Lastly if our phase 2 of our pipeline has worked perfectly and transformation has returned well structured data, we can go for SIF as it saves training time and is efficient.

Chapter 5

Evaluation

This chapter we will discuss our input datasets, the parameters considered in deep learning models, the semantic conflicts identified, resolved and finally our results, accuracy with respect to all mentioned factors. We divide the sections into two for each datasets we used and we will also mention the challenges faced during the pre processing phase. After the conflict handling and transformations on our datasets, we pass the data to our deep learning models which in our case is the deep matcher. The deep matcher architecture include attribute embeddings which are in our case pre-trained character-level embedding (Fasttext). We test our data on all attribute summarizers like SIF, RNN, Attention and Hybrid. The classifier we use is a multilayered Neural Network which is a popular choice for DL classifier solutions. We use F1 measure for measuring accuracy of our deep learning model which we define below

$$\begin{gathered}
Precision = \frac{TP}{(TP + FP)} \\
Recall = \frac{TP}{(TP + FN)} \\
F1 = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \\
where, TP = True\ Positives \\
FP = False\ Positives \\
FN = False\ Negatives
\end{gathered} \tag{5.1}$$

5.1 Itunes Dataset

In Itunes dataset experiments we interlink individuals of two different subsets of Itunes dataset which follow different structure and schema. Itunes dataset is a RDF dataset comprising of details about songs on Itunes platform. The dataset is divided into two different schemas with two sets of song entities abiding by their respective schemas. We need to identify if song abiding by its schema is same as a song from the other schema i.e if they are the same entity. We already have some labelled data in hand for Itunes songs where some pair of song resources are mapped using sameAs or differentFrom. These individuals have many data properties like name, albumname, price etc for us to consider in our approach. These properties have semantic interoperability conflicts. We define and discuss these conflicts that came our away in our approach. Also to add how we resolved these conflicts using conflict resolution functions.

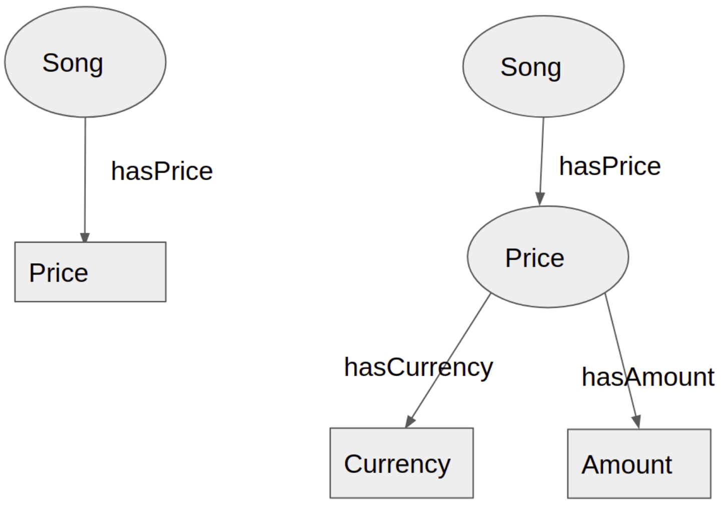


FIGURE 5.1: ITunes Dataset representative conflict resolution

- **Representative Conflicts:** The data property Released date for both schemas follow different date formats. These varying representations can really affect accuracy of our base deep learning model. We standardize date formats for Released date attribute in both schemas and change values for literals accordingly. Now in the Song Price data property, we have song prices represented in different currencies. We split the price field into currency and amount as shown in Figure 5.1. After the splitting we can standardize the currency in the desired format which deep learning model can distinguish.

- **Domain Conflicts:** The artist name property in both schemas has different names/alias for the same artist. Some of the names are hard to distinguish because it can be their alias which is completely different from their original name(complete string mismatch). We resolve these conflicts using sparql where we create a blocking predicate to identify an artist. If the name is different for the same artist, we standardize the name and remove alias. For example Flo Rida is also called "Tramar Lacel Dillard'. No string match algorithm can flag them as same. If we know more information on the artist like songs he sang, albums we can distinguish the strings.

F1 measure			
No.	Summarizer	Epoch=15	Epoch=30
1	SIF	81.82	83.07
2	**RNN**	**89.28**	**87.71**
3	Attention	83.87	86.66
4	**Hybrid**	**93.1**	**88.88**

TABLE 5.1: Itunes Dataset

In the Deep learning model phase we test our dataset using deep matcher. As you can see from the table 5.1 above we can observe the Hybrid summarizer gives us the best results on Itunes data. SIF summarizer provides us lesser accuracy because the data is not completely well structured plus it has many empty literals, few dirty data, long texts like copyright and not all conflicts are resolved. However SIF needs far less training time. When we increase the iterations for SIF, the accuracy increases. RNN gives good accuracy because it performs well on long text attributes, dirty data. Both RNN and attention also require high training times but accuracy of attention increases with higher iterations. The attribute embedding is fasttext and the batch size is set to 16. It is recommended to use Hybrid unless our pre processing phase has gone really well and data completeness is near perfect.

5.2 DBpedia and Wikidata Dataset

In the second part we wanted to cover the open linked datasets. We have tried to apply our approach to linked open datasets. One big challenge is handling the large knowledge base and its schema as we can see in Figure 5.2. In comparison to Itunes ontology, DBpedia ontology is very huge with hundreds of classes. Also our approach wont be straightforward because most of these open datasets are not complete, contain dirty and redundant data. DBpedia, Yago and Wikidata has very few property mappings with limited labels and may not necessarily abide by OWL/RDF schema. The solution is to hence apply our approach to a subset of the knowledge base. We try to match film class individuals in wikidata and DBpedia. We take some existing mappings and labels but we had to create some mappings on our own as a part of preprocessing. We had to filter out incomplete properties and data. After this we identify conflicts and try to resolve them in the conflict handling phase.

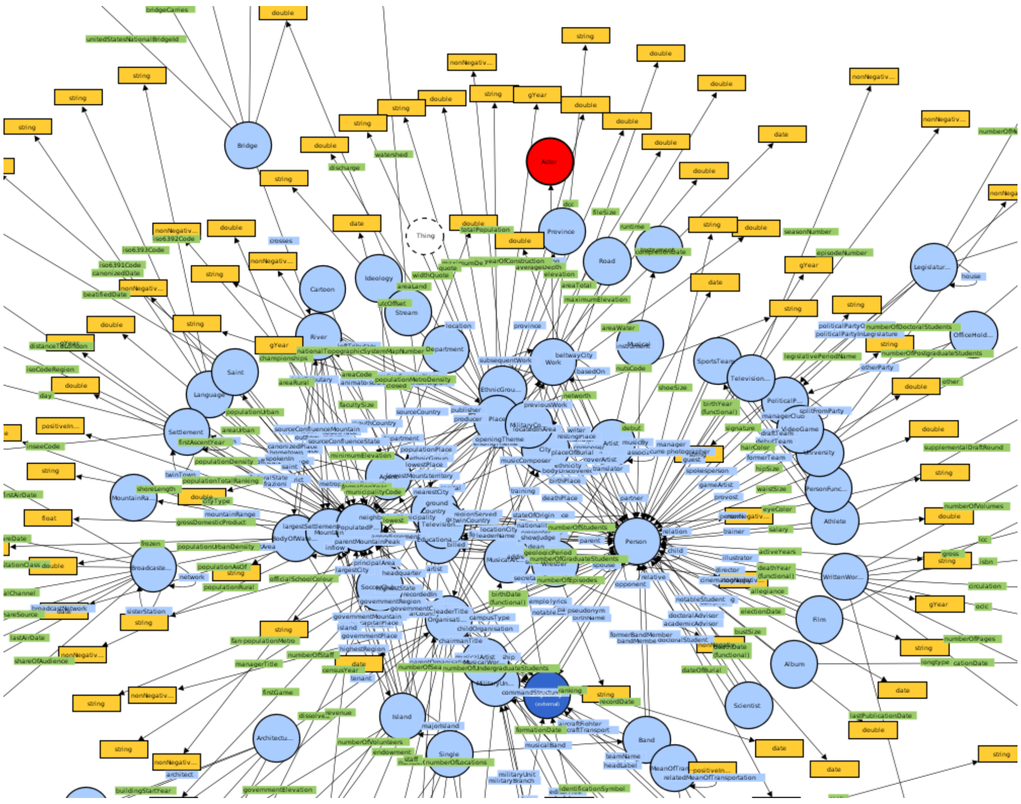


FIGURE 5.2: Above we see a glimpse of DBpedia schema. Because of the scale we match individuals from film class (in red) and take subgraph surrounding it for learning properties

- **Representative Conflicts:** . Gross Amount literal was represented in different formats. We had to standardize the representation by using a single currency to help deepmatcher distinguish the individuals. First we split gross amount to gross currency and gross amount. After a currency is decided upon we can standardize it.

- **Structured Conflicts:** ImdbId was represented as integer in wikidata and as result some of the alphabets were truncated out. We had to convert it to string and agree on a common datatype for wikidata and DBpedia attribute ImdbId. Hence the datatype was standardized to string for both set of literals.

F1 measure			
No.	Summarizer	Epoch=15	Epoch=30
1	SIF	84.12	83.87
2	**Attention**	**97.44**	**97.74**
3	**RNN**	**94.8**	**94.89**
4	**Hybrid**	**95.22**	**96.96**

Table 5.2: Dbpedia Wikidata Dataset

Like the Itunes dataset in wiki-DBpedia dataset we can also observe in the table 5.2 that the accuracy is high for Hybrid due to incompleteness, empty and dirty data for literals. Similarly the accuracy is also high for RNN and attention but is lower for SIF due to incomplete properties. For attention and Hybrid accuracy increases with iterations. The attribute embedding is fasttext and batch size is 16. Due to incompleteness of data, dirty data and empty literals hybrid is highly recommended on Linked Open Datasets.

Chapter 6

Conclusion and Future work

In this chapter we look back at our work done in this thesis, and discuss the distinctness, drawbacks of our approach, how we can improve it and how we have managed to contribute to the Knowledge Graph community.

Our primary task in hand was to match entities of two RDF datasets. The approach is supervised, i.e., it relies on labels (e.g, we can take advantage of the sameAs links already annotated by the Semantic Web community) and it needs property mappings. Based on our experiments, our approach is robust to dirty/unstructured/semi structured data and promises good accuracy. However it expects the datasets to stay compliant with OWL/RDF schema. We need to understand that Wikidata and DBpedia do not necessarily abide by OWL/RDF schema and it requires a more pre processing if we were to use such datasets in our approach. It also is heavily vulnerable if the graph paths are broken because of the tree traversal technique it follows.

While comparing our method to SEREMI [22], music interlink [24] approaches, our method is more sophisticated because it merely does not work on raw string match at the atomic level. It can explore far more number of properties and is a more supervised approach which is immune to conflicts and incompleteness of data. Our approach also avoids using SPARQL wrappers as query fetches can be slow and can give unexpected number of results on filters. We have proposed a way to use deep matcher for entity matching on RDF graphs and improved its accuracy when there are semantic interoperabiltiy conflicts. In the background work to the thesis we inferred about poor performance of Deep Matcher when there are Semantic interoperability conflicts.

Our pipeline heavily relies on existing mappings and labels. These mappings have to be created manually which is a overhead. Data integration and schema based approaches help learn mappings [25, 26]. There has been work done in automating property mappings and these are not incorporated in our approach. Our approach also does not take learn the schema or object properties during the approach. This is another drawback. LiteralE[29] uses the schema, object properties and literals in its approach. It uses knowledge graph embeddings instead of embeddings provided by deep matcher. Knowledge graph embeddings can represent graph data and its semantics better.

To summarize we have presented an approach which performs well on literals with large textual data, dirty data and it matches entities based on its data properties. However the drawbacks include that there is no schema learning or knowledge embeddings being used.

6.1 Future Works

There can be a lot of extended contributions done with respect to reducing the overhead. The manual dependency on property mappings can be removed by using Ontology Graph keys [26] for schema learning and also we can deduce methods that use graph embeddings to learn semantics prior to sending to the deep learning model.

Apart from that the approach should be extended to non OWL/RDF schemas, and has to be made more robust to learning different types of schemas with less manual interventions. The deep learning model can be changed from deep matcher, and other models can be tested. New techniques can be developed using deep matcher for Knowledge Graphs with dirty data, noisy data where we can try new types of link detection apart from sameAs and different From.

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Data never sleeps 2.0, 2020. URL `https://www.domo.com/assets/downloads/18_domo_data-never-sleeps-6+verticals.pdf`.

[2] Rachel Howard Bryce Merkl Sasaki, Joy Chao. *Graph Databases for Beginners*. MIT Press, 2016. `ttps://neo4j.com`.

[3] Ehud Reiter and Robert Dale. Building natural language generation systems, 2000.

[4] Er.Ramanjot Kaur Er. Karmjeet Singh. A review: Various hashing algorithms in data deduplication for storage enhancement. *INTERNATIONAL JOURNAL OF RESEARCH IN ELECTRONICS AND COMPUTER ENGINEERING*, 6(1):193–196, 2018.

[5] Cengiz Kahraman Sezi Çevik Onar, Başar Öztayşi. Record linkage using fuzzy sets for detecting suspicious financial transactions. pages 241–246, 2015. doi: 10.2991/ifsa-eusflat-15.2015.36.

[6] Antoine Buemi Erik A Sauleau, Jean-Philippe Paumier. Medical record linkage in health information systems by approximate string matching and clustering. *BMC Medical Informatics and Decision Making*, (1):1–13, 2005. doi: 10.1186/1472-6947-5-32.

[7] Jim Hendler Ian Horrocks Deborah L. McGuinness Peter F. Patel-Schneider Lynn Andrea Stein Sean Bechhofer, Frank van Harmelen. Owl web ontology language reference, 2004. URL `https://www.w3.org/TR/owl-ref/`.

[8] Nektarios Gioldasis Ioannis Stavrakantonakis Stavros Christodoulakis Nikos Bikakis, Chrisa Tsinaraki. The xml and semantic web worlds: Technologies, interoperability and integration. a survey of the state of the art. pages 1–42, 2013. doi: 10.1007/978-3-642-28977-4_12.

[9] Dan Brickley and R.V. Guha. Rdf schema 1.1, 2009. URL `https://www.w3.org/TR/rdf-schema/`.

[10] Axel-Cyrille Ngonga Ngomo Erhard Rahm Markus Nentwig, Michael Hartung. A survey of current link discovery frameworks. pages 419–435, 2017. doi: 10.3233/SW-150210.

[11] Yunlong Zhang Xing Niu, Shu Rong and Haofen Wang. Zhishi.links results for oaei 2011. volume 814, pages 220–227. OM'11: Proceedings of the 6th International Conference on Ontology Matching, 2011.

[12] Bernardo Cuenca Grau Ernesto Jiménez-Ruiz. Logmap: Logic-based and scalable ontology matching. The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, 2011.

[13] Manzil Zaheer Luke Vilnis Ishan Durugkar Akshay Krishnamurthy Alex Smola Andrew McCallum Rajarshi Das, Shehzaad Dhuliawala. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. Proceedings of the 2018 International Conference on Management of Data, 2018.

[14] Prof. Dr. Sören Auer Trlan Grangel-González, Prof. Dr. Maria-Esther Vidal. A knowledge graph based integration approach for industry 4.0, 2019.

[15] Mark Hudson Beale Orlando De Jesús Martin T. Hagan, Howard B. Demuth. *Neural Network Design*. 2014.

[16] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006. ISBN 978-0-387-31073-2.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Neo4j, 2018. http://www.deeplearningbook.org.

[18] Dmitriy Serdyuk Philemon Brakel Yoshua Bengio Dzmitry Bahdanau, Jan Chorowski. End-to-end attention-based large vocabulary speech recognition. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, 2016. doi: 10.1109/ICASSP.2016.7472618.

[19] Navdeep Jaitly Alex Graves. Towards end-to-end speech recognition with recurrent neural networks. pages 1764–1772. ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, 2014.

[20] Alex Graves. *Supervised Sequence Labelling*. 2012.

[21] Theodoros Rekatsinas AnHai Doan Youngchoon Ganesh Krishnan Rohit Deep Esteban Arcaute Vijay Raghavendra Sidharth Mudgal, Han Li. Deep

learning for entity matching: A design space exploration. ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, 2016. doi: 10.1145/3183713.3196926.

[22] Daniel Schwabe Arjen P. de Vries Samur Araujo, Jan Hidders. Serimi - resource description similarity, rdf instance matching and interlinking. 2011.

[23] Enrico Motta Andriy Nikolov, Victoria Uren and Anne de Roeck. Handling instance coreferencing in the knofuss architecture. 2008.

[24] Christopher Sutton Yves Raimond and Mark Sandler. Automatic interlinking of music datasets on the semantic web. 2008.

[25] Alon Halevy Anhai Doan, Pedro Domingos. Learning to match schemas of data sources: A multistrategy approach. 2003. doi: 10.1023/A:1021765902788.

[26] Yinghui Wu Hanchao Ma, Morteza Alipourlangouri. Ontology-based entity matching in attributed graphs. 2003. doi: 10.14778/3339490.3339501.

[27] Xifeng Yan Philip S. Yu Tianyi Wu Yizhou Sun, Jiawei Han. Pathsim: Meta path-based top-k similarity search. 2011.

[28] Dipl.-Ing. Ignacio Traverso Ribón. A framework for semantic similarity measures to enhance knowledge graph quality, 2017.

[29] Denis Lukovnikov Jens Lehmann Asja Fischer Agustinus Kristiadi, Mohammad Asif Khan. Incorporating literals into knowledge graph embeddings. 2018.

[30] Prof. AnHai Doan Prof. Theodoros Rekatsinas Sidharth Mudgal, Han Li. Deepmatcher documentation, 2016. URL `https://anhaidgroup.github.io/deepmatcher/html/#`.

[31] Shishir Prasad Ganesh Krishnan Pradap Konda, Jeff Naughton. Magellan: Toward building entity matching management systems. 2016. doi: 10.14778/2994509.2994535.