Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

Documenting Knowledge Graph Embedding and Link Prediction using Knowledge Graphs

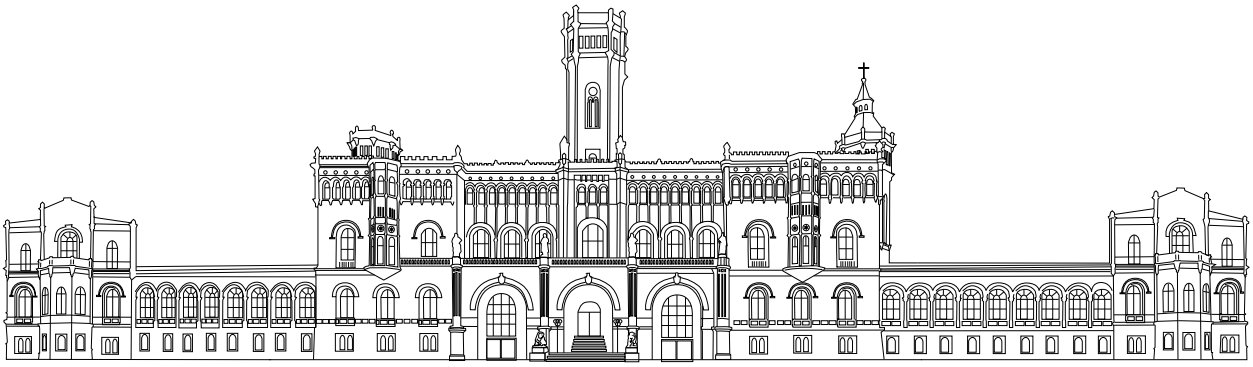
A thesis submitted in fulfillment of the requirements for the degree of
Master of Science (M. Sc.)

BY

Huaxia Zhao
Matriculation number: 10032756
E-mail: strongzhx@gmail.com

First evaluator: Prof. Dr. Maria-Esther Vidal
Second evaluator: Prof. Dr. Sören Auer
Supervisor: M.Sc. Yashrajsinh Chudasama

February 1, 2024



Declaration of Authorship

I, Huaxia Zhao, declare that this thesis titled, 'Documenting Knowledge Graph Embedding and Link Prediction using Knowledge Graphs' 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.
- I have acknowledged all main sources of help.

XXXX

Signature: _____

Date: 01.02.2024

Acknowledgements

Thanks to Prof. Dr. Maria-Esther Vidal for allowing me to complete my master's thesis in the TIB-SDM group. And also thanks to the lecture Prof. Vidal holds in the last two semesters. I learned a lot about the field of knowledge graphs. Her comments and suggestions, as well as all aspects of support, were very helpful to me. Thanks to my supervisor M.Sc. Yashrajsinh Chudasama for providing goals and direction, as well as for providing guidance and help throughout the process. I have benefited a lot from the weekly discussions. Thanks to all members of the SDM team who were very kind to me. Thanks to my parents, I was able to study abroad and complete my studies in a foreign country. Thanks to my girlfriend for the companionship and support. Thanks to the many friends I have met along the way.

Abstract

In recent years, sub-symbolic learning, i.e., Knowledge Graph Embedding (KGE) incorporated with Knowledge Graphs (KGs) has gained significant attention in various downstream tasks (e.g., Link Prediction (LP)). These techniques learn a latent vector representation of KG's semantical structure to infer missing links. Nonetheless, the KGE models remain a black box, and the decision-making process behind them is not clear. Thus, the trustability and reliability of the model's outcomes have been challenged. While many state-of-the-art approaches provide data-driven frameworks to address these issues, they do not always provide a complete understanding, and the interpretations are not machine-readable. That is why, in this work, we extend a hybrid interpretable framework, InterpretME [12], in the field of the KGE models, especially for translation distance models, which include TransE, TransH, TransR, and TransD. The experimental evaluation on various benchmark KGs supports the validity of this approach, which we term Trace KGE. Trace KGE, in particular, contributes to increased interpretability and understanding of the perplexing KGE model's behavior.

Keywords: Knowledge Graph, Knowledge Graph Embeddings, Interpretability

Contents

List of Figures

List of Tables

Acronyms

AutoML Automated Machine Learning

HPO Hyperparameter Optimization

KG Knowledge Graphs

KGE Knowledge Graph Embedding

LIME Local Interpretable Model-agnostic Explanations

ML Machine Learning

MRR Mean Reciprocal Rank

PyKEEN Python KnowlEdge EmbeddiNgs

R2RML RDB to RDF Mapping Language

RDF Resource Description Framework

RML RDF Mapping Language

SHAP SHapley Additive exPlanations

SPARQL SPARQL Protocol And RDF Query Language

TSV Tab-separated values

URI Uniform Resource Identifier

Chapter 1

Introduction

Recently, the sub-symbolic systems, i.e., the Machine Learning (ML) [23] methods have become a widely used technology. From facial recognition to self-driving cars, from personalized recommendations to cancer diagnosis, ML is used in an increasing number of applications [24, 30, 42]. However, these different types of models can provide many statistical results that help us to measure the performance of models, such as accuracy, recall, precision, and F1-score. Table 1.1 demonstrates the traditional ML approach uses dense numeric vectors as input to represent data, and an output with metrics score, such as (true:0.7, false: 0.3) to represent the result is 70% to be true. Nevertheless, the process of decision-making by such models is still opaque, we hardly know how the result comes out, and it is also hard to get a completely correct result every time from the models. Therefore, making the decisions of the model more trustable and interpretable is something that many researchers have been working on [29, 33]. For some critical decisions or medical diagnoses (e.g., cancer), experts cannot simply trust a result that only depends on the high values of a group of metrics, but also on explanations of a model's behavior [33]. Moreover, data-driven frameworks such as SHAP [29] and LIME [33] have shown considerable achievement in various domains (e.g., healthcare). In the biomedical field, cancer is a complex and multifactorial disease, and it can indeed be influenced by various factors. For Oncologists, their decision is always based on different information. For a model, it will analyze and get results based on limited resources, such as some images with cancer characteristics, or some information about patient symptoms as input of the model. But users hardly know which part (or key points) will be treated by a model as critical information and lead to a prediction result in the biomedical area, e.g., cancer. However, such global and local explanations (e.g., relevant features list) of each instance for the prediction task are human-readable. This is the reason why

Table 1.1: The dataset in the format of a hot encoding for traditional ML model, it is different from the Table 1.2, whose input already has features that could be identified by the classifier, the target of traditional ML models is to train the model and make the model able to recognize these features.

Person	hasChild	hasGrandChild	gender	hasSpouse
Afonso	1	1	0	1
Matilda	1	1	1	1
Beatrice	0	0	1	0

even though the model's performance with high accuracy, the experts are still not able to fully trust the ML model's predictions [15]. Therefore, the necessity for research on interpretable methods of such complex ML models is required to facilitate the user with more comprehensive human– and machine–readable interpretations. On the other hand, Knowledge Graphs (KGs) are structured graph data models of real-world knowledge, where nodes are linked by directed edges denoted by labels conveying semantic relations which results in RDF triples called facts. Each fact is represented by ⟨subject, predicate, object⟩. In exemplary cross-domain, several RDF KGs, such as DBpedia [26], Wikidata [38] have been a widespread concept and shown tremendous progress in the assessment of KGs or a recommendation systems [19]. Table 1.2 represents an excerpt from the DBpedia KG about a french royal member. It is a way to represent the data by graphs and also use many methods to enhance the knowledge it carries. It can include processes such as data integration, knowledge representation, and knowledge discovery, and can be used in predictive models. Further, the sub-symbolic learning can be applied to the field of knowledge graphs for not only Knowledge Graph Completion (KGC) task (i.e., Link Prediction (LP)) [7], but also to accomplish some of the downstream tasks of knowledge graphs, such as question answering, chat-bot, and recommendation systems [20, 37, 40].

Knowledge Graph Embedding (KGE) models is a sub-symbolic approach that creates a continuous low-dimensional vector space for entities and relations that can better express the graph, which can then be further used for the prediction of a missing links [4, 19]. The KGE model has gained significant attention [39], relies on supervised learning, and like other ML models, is a black box model. KGE models are very different from how normal ML models usually represent data. Here, the KGE models use a vector space to represent the entities and relations of the KGs in n-embedding dimensions (at least one vector space for entities, and some KGE models have an additional vector space for relations). Table 1.3 depicts the latent vector representation of entities and relations from Table 1.2 in 2-dimensional space.

Table 1.2: The dataset in the format of RDF triples, will not have any features available for the classifier, needs to learn the embeddings based on the model.

Subject	**Predicate**	**Object**
dbr:Adalard_of_Paris	dbo:hasSpouse	No
dbr:Adelaide_of_Paris	dbo:parent	dbr:Adalard_of_Paris
dbr:Adelaide_of_Paris	rdf:type	dbr:Person
dbr:Adelaide_of_Paris	dbo:child	dbr:Charles_the_Simple
dbr:Adelaide_of_Paris	dbo:spouse	dbr:dbr:Louis_the_Stammerer
dbr:Adelaide_of_Paris	dbo:gender	female

Table 1.3: The output of the KGE model in the format of 2-dimensional vector space for the KGE model, embeddings between entities and relations will carry information during the learning process of the model.

2-Dimension	x	y
dbo:hasSpouse	0.9584	-0.6897
dbr:Adalard_of_Paris	0.3279	0.9447
0	0.9122	-0.4097
female	-0.4295	0.9031
10	0.3416	-0.9399
dbr:Marie_de'_Medici	-0.9420	-0.3356

Slightly different from common the ML models, we could use several ranking metrics to evaluate the performance of the KGE models, such as Mean Reciprocal Rank (MRR) and Hits@k (e.g., Hits@10 or Hits@1) described in Chapter 2.

Moreover, in time, the KGE models are capable of performing several tasks, such as Entity Clustering (EC), Node Classification (NC), and LP. Nevertheless, the inner mechanism of such translation models is ambiguous. Consider an example of tail prediction (i.e., $\langle e_h$, r, ?$\rangle$), where given a head entity (Adalard_of_Paris), a relation (hasSpouse) and the goal is to predict the missing tail entity e_t. The translation KGE models can easily learn the representation to predict the link. Nonetheless, this prediction is not self explanatory. Several approaches [11, 34] have been proposed to investigate these research directions. Here, explainable frameworks, such as LIME [33] provides the important features listed as vectors and this representation is hard to understand and also cannot be translated back to original entity characteristics. Closely related to our work, InterpretME [12] is capable of providing the interpretations of ML model decisions in a human-and-machine readable format.

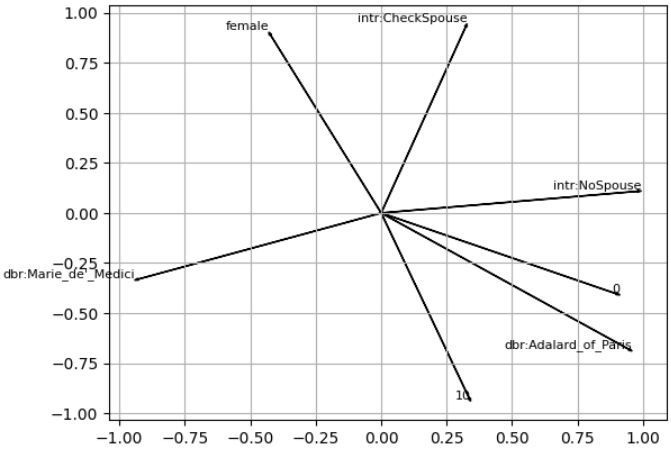


Figure 1.1: Visualization of Table 1.3 in 2D axes format, each entity and relation from the dataset in low dimensional space shown as a vector, further the result of this learning is shown as a change in the vector of the entity shown in Figure 2.4.

Such semantic web technology tools provide a perception of traceability and reliability currently for traditional ML models, thus more research needs to be conducted, in particular, interpreting the KGE models that handle latent vector representation.

1.1 Motivating Example

In the evolving field of data science, information loss due to the passage of time, missing and corrupted data, and other factors is a widespread challenge [9, 10]. Historical databases, created by less advanced technologies and incomplete documentation, are often the most vulnerable to information loss. The critical task of addressing missing data has become an integral part of data assessment. The use of machine learning methods to refine data based on KG characteristics has become a viable solution. Figure 1.2 illustrates our motivation, we take the data refinement problem of the French Royalty KG dataset as an example, aiming to refine whether the target head has a spouse or not using the existing known characteristics of a royal member.
According to the pipeline, we have a structured KG dataset from an endpoint, we could get a subgraph via SPARQL CONSTRUCT clauses. This subgraph is different from the input data required by the common ML model, and the difference can be seen in Table 1.1 and Table 1.2, with this subgraph and some predefined parameter configurations to a KGE model (e.g., TransE), we could use KGE models

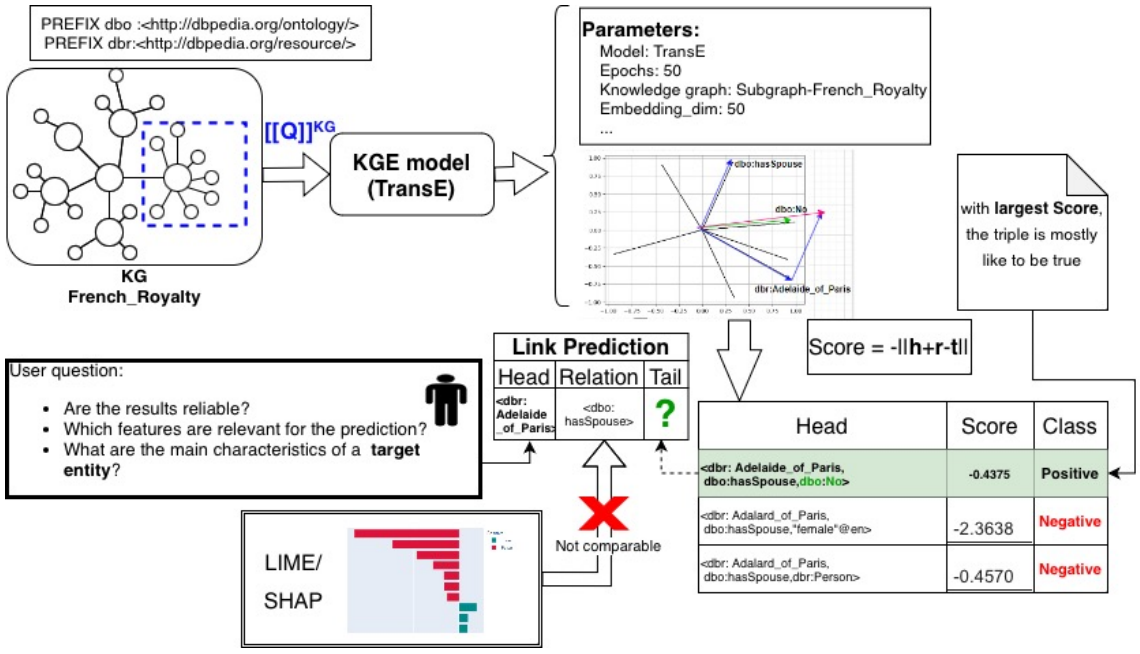


Figure 1.2: **Motivating Example.** Retrieving an RDF sub-graph via SPARQL queries over the French Royalty KG endpoint, a KGE model (e.g., TransE) can be used to predict the missing links. Although tools like LIME and SHAP can be utilized, in general, results generated by these models lack explainability where the learn embeddings is represented in latent vector space.

to get embeddings of entities and relations shown in Table 1.3. A simple example with 2 dimension space depicted in Figure 2.2, with these embeddings, the KGE models could predict missing links, e.g., with a target entity "dbr:Adalard of Paris" as Head, a relation "hasSpouse" and we want to predict the tail entity to refine the triple structure (Head, Relation, Tail). After the training phase, the KGE model will provide feedback with embeddings and allow us to calculate a complete list of scores through a score function, and with a target *head* and a target *relation*, we could find out the most plausible *tail* entity ranking with the highest score.

However, as discussed above, it is hard to convince users that the result is reliable only with a high value of the plausibility score of a triple, when research could show that triples with a high value of the plausibility score are likely to be true [7], and a low value to triples that are likely to be false. By comparing Table 1.1 and Table 1.2, it can be observed that the inputs required by the regular ML model can already be more intuitively observed for some features, and by changing some of the values,

the effect of the feature can be quickly investigated [33], but the inputs of the KGE model can not be adjusted arbitrarily. The two state-of-the-art declarative tools: SHAP [29] and LIME [33]), currently do not consider the KG as input and are not able to provide a proper explanation for the KGE models. This makes the KGE model an imperfect fit at a time when there is a constant quest for higher and better model explainability. However, with the input and output of KGE models, it is able to combine them in an improved context to improve its explainability. In this thesis, a pipeline, Trace KGE, is proposed that can complete the whole process of tracing the KGE model, including data preparation, model building, model training, obtaining results and metrics, and at the same time improve the interpretability of the model through the feature of KG being able to carry rich contexts.

1.2 Contributions

The main contributions of this thesis are summarized as follows:

- This work contributes a pipeline to apply KGE models beginning with extracting training data (subgraph from the endpoint) and using methods for tuning hyperparameters used in the KGE models, then training and evaluating models. During these phases, documenting all relevant contextual metadata, e.g., RDF data from the endpoint or locally stored dataset, hyperparameters, model characteristics and metrics like MRR and Hit@10 from the evaluation phase of the pipeline, which are used to measure the performance of models. In the end, utilizing the RML mapping rules for the traced metadata to generate a new KG named InterpretME KG. Further, using SPARQL queries over the InterpretME KG to retrieve the interpretations of the KGE models.

- This work also optimizes a large number of repetitive operations for the user and reducing the difficulty and risk of errors for the execution of pipeline. Moreover, helping the user to understand the inner characteristics of the model and to analyze the model's performance through the richness of the contexts stored in the InterpretME KG. Extending the IntepretME project in the direction of KGE model interpretability research enables IntepretME to use translation distance models for the link prediction task and demonstrates the results.

- Additionally, an ablation study confirmed that InterpretME pipeline helps to understand the influence of hyperparameters in the evaluation of model's performance in a more efficient way.

1.3 Document Structure

The thesis contains seven chapters. Chapter 1 is the introduction and also shows the reader a motivating example, the contributions, and the problem tackled in this work. Chapter 2 summarizes the basic concepts required to understand the following chapters. In Chapter 3, the reader can find approaches related to this work, and summarize the state-of-the-art approach. The proposed solution and extensions and improvements to it are discussed formally in Chapter 4, it also shows a running example, which is the same as the motivating example and from the same dataset. Chapter 5 describes how to implement the solution proposed in the previous chapter, and introduces the packages that help in the implementation process. The next chapter, Chapter 6 starts with two research questions, evaluating the proposed pipeline by conducting multiple sets of experiments and analyzing it, in order to answer the research questions posed. The final chapter, Chapter 7 contains a conclusion, the known and possible limitations of the pipeline, and suggests ideas that might improve or refine this work in future work.

1.4 Summary of the Chapter

This chapter briefly describes the applications of ML techniques, the need for model interpretability, and the implications of these developments for the field of Knowledge Graphs, shows the reader the contributions of this work and the problems to be solved by means of a motivating example, and describes the structure and relevance of the entire thesis.

Chapter 2

Background

This chapter introduces some basic concepts used in this thesis. The background part can be divided into two main parts. The first part is about the basic concepts and the topic of KG, and the second part introduces approaches related to inductive knowledge, focusing on TransFamily.

2.1 Knowledge Graphs

2.1.1 Resource Description Framework

The RDF is a standardized data model recommended by W3C (WWW Consortium). RDF is a loosely typed (or semi-structured) data model. It uses RDF triples structure to store data, which statement as subject, predicate, and object(head, relation, and tail) [8]. Given the infinitive sets R (set of Uniform Resource Identifier (URI)), B (set of Blank Nodes), and L (set of literals). An RDF graph G is defined as a labeled directed graph G= (V, E), where: V $\subseteq$ (R $\cup$ B $\cup$ L), E is a set of RDF triples t= (V1,V2,V3), V1 is the subject of t and V1 $\in$ (R $\cup$ B), V2 is the property of t and V2 $\in$ R, V3 is the object of t and V3 $\in$ (R $\cup$ B $\cup$ L).

Typically hierarchical structure of URI:

[scheme]://[authority][path][?query][#fragment]

scheme: Indicates the protocol to access the resource (e.g., http, https, ftp).

authority: the domain or IP address of the server hosting the resource. It may also include port information(e.g., dbpedia.org, user:password@example.com:8080).

path: the path where we could find the resource on the server, like ontology.

query: start with question mark **?**, use query to retrieve the resource, has additional parameters, in terms of ?P1=value1&P2=value2.

fragment: start with slash mark # identifier of the fragment to be accessed.

Examples of URI

`https://dbpedia.org/ontology/spouse`
`http://dbpedia.org/resource/Edward_I_of_England`
`https://www.w3.org/1999/02/22-rdf-syntax-ns#type`

RDF represents factual statements as triples: subject, property, and object. Figure 2.1 shows the RDF triples linked with the prefixes in the form of URLs. *prefix* is a type of URI that enhances readability and maintainability. Examples of prefixes are provided below Table 2.1. In Figure 2.1, `dbr:Edward_I_of_England` is the subject represents a resource *Edward I of England* that was extracted from DBpedia with the assigned prefix `dbr` utilized for resources.

Structured data models:
A data model is strictly typed, also known as a structured data model, if each data point must fall into one of several types or categories. Types are predefined and cannot change dynamically. A relational data model Table 1.1 is one of the examples of structured data models.

A relational database has a unique key for each row in the table. The columns of a table describe the type or category of data, and each record typically has a value for each type. Relational tables accept NULL values in some cases, however, in the ML field, this might affect training and evaluation, leading to incorrect conclusions and unreliable analysis [10].

Semi-structured data models:
Loosely typed (or semi-structured) data models allow items to exist independently and be connected without adhering to a category or being of a specific type.

RDF translates factual statements into three small components as resources: the subject (a URI or a blank node), the predicate (a URI), and the object (a URI, literal, or blank node). It establishes a foundation for combining identifying, and distributing structured and semi-structured data across various applications or protocols[1].

Blank nodes: Blank nodes correspond to existential variables. They denote the existence of an unknown resource that makes the triple true.

[1] `https://www.w3.org/RDF/#`

@prefix dbo:<http://dbpedia.org/ontology/>
@prefix dbr:<http://dbpedia.org/resource/>
@prefix dbp:<https://dbpedia.org/property/>

Subject	Predicate	Object
dbr:Edward_I_of_England	dbp:father	dbr:Henry_(son_of_Edward_I)
dbr:Eleanor_of_Castile	dbp:mother	dbr:Henry_(son_of_Edward_I)
dbr:Eleanor_of_Castile	dbo:spouse	dbr:Edward_I_of_England
dbr:Edward_I_of_England	dbo:spouse	dbr:Edward_I_of_England
dbr:Edward_I_of_England	dbo:spouse	dbr:Margaret_of_France,_Queer

Figure 2.1: **Example of RDF Triples** . The RDF model interprets the natural language representation of sentences. "Edward I was Henry's father, and he married Eleanor of Castile and Margaret of France. Eleanor is Henry's mother and had a spouse named Edward I" into RDF triples of the form subject, predicate, and object.

Prefix	Namespace
rr	http://www.w3.org/ns/r2rml#
rml	http://semweb.mmlab.be/ns/rml#
ql	http://semweb.mmlab.be/ns/ql#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
schema	http://schema.org/
xsd	http://www.w3.org/2001/XMLSchema#
owl	http://www.w3.org/2002/07/owl#
prov	http://www.w3.org/ns/prov#
intr	http://interpretme.org/vocab/
dbo	http://dbpedia.org/ontology/
dbr	http://dbpedia.org/resource/
dbp	https://dbpedia.org/property/
foaf	http://xmlns.com/foaf/0.1/

Table 2.1: Prefixes and Their Associated Namespaces

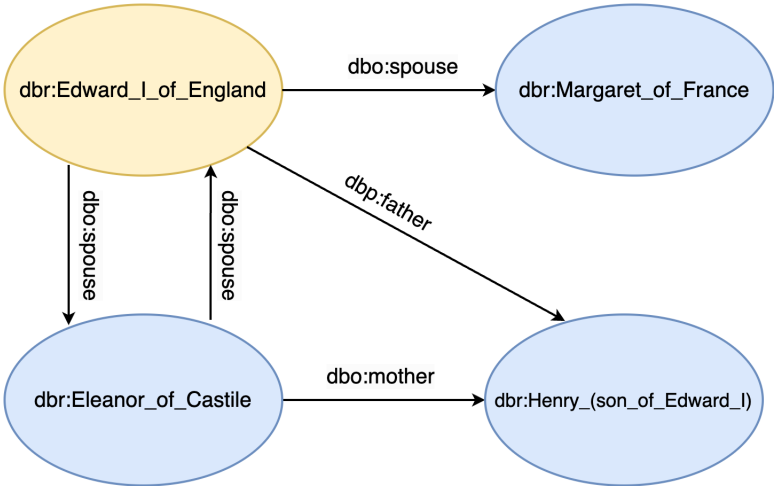


Figure 2.2: **Data Graph**. The example in Figure 2.1 depicts a subgraph concerning the entity `dbr:Edward_I_of_England`. The graph depicts an entity's connection to its spouse, mother, and father.

> **Literals:** Literals that are of a certain type can be specified using XML schema datatypes (e.g., string, integer, float or date).

Examples of Literals

"Docetaxel" http://www.w3.org/2001/XMLSchema#string
"4500.60" http://www.w3.org/2001/XMLSchema#float
"2022-10-21" http://www.w3.org/2001/XMLSchema#date

2.1.2 Graphs

> **Graphs:** graph is a structure that consists of a set of nodes and connected by a set of edges. Graphs are widely used to model relationships and connections between different entities. [17]

Data Graphs

> **Data Graphs:** DGs are structures that store data in the format of graphs, which have nodes connected by edges [19].

From Figure 2.2, could see nodes represent entities and edges could show relations, and it could be generated from triples.

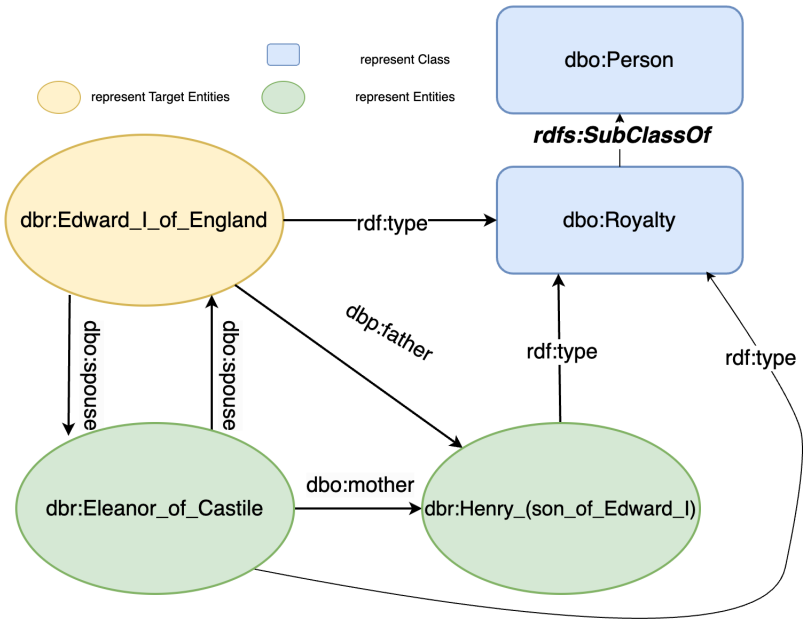


Figure 2.3: The illustration depicts an example of a KG, which contains not only data but also metadata that describes the entities, allowing the reader or machine to better understand the content in a limited context. The example illustrates that there are three entities, and that there is some relationship between the three entities, it also tells us through the metadata that the three entities are all human beings, but they are also members of the French royal family.

Knowledge Graphs

> **Knowledge Graphs:** Given a set Con of countable infinite constants. *Knowledge Graph* [19] is defined as a directed edge-labeled graph G = (V,E,L) where V $\subseteq$ Con is a set of nodes, L $\subseteq$ Con is a set of edge labels, and E $\subseteq$ V$\times$L$\times$V is a set of edges

Labeled edges provide significance to the knowledge graph's entities by encapsulating structured information. That is, a labeled edge can be regarded as a sentence consisting of a subject (the entity), the predicate (the edge's label), and the object (the literal or another entity). Multiple sentences (or triples in the RDF graph) about an entity sets it in a semantic context, giving it meaning. Unlike data graphs, knowledge graphs contain both data and metadata.Figure 2.3. Metadata and data can be combined with inference methods to deduce new facts.

Metadata: *Metadata* refers to organized information that describes, explains, locates, and represents data. It simplifies the complexity of an object. Metadata facilitates the retrieval, use, and management of scientific data.

2.1.3 SPARQL

SPARQL: SPARQL is a query language, that defines the protocol for using clauses to make SPARQL queries and receiving the query results from RDF. It can retrieve the expected result sets or result graphs from a KG or any data source that can be mapped to RDF [35]. It is specified, structured, and maintained by W3C.
A SPARQL query consists of two main sections and several optional sections, which can be formed into
[clause section][WHERE section{optional1}]{optional2}

SPARQL commonly supports four types of query clauses: SELECT, CONSTRUCT, ASK, and DESCRIBE. It also supports the use of namespaces with prefixes to simplify and improve the readability of queries. Several prefixes could be read in Table 2.1. In the clause section, we need to define the required result, which will be columns of the result set or the structure of the graph. *WHERE* section is used to limit the result or help us get more specific results. There are also many useful operators in the WHERE section, like FILTER, LIMIT, and so on, to help restrict the result, and there is an optional2 section that can use GROUP BY, ORDER BY, etc. More descriptions and examples can be found in W3C website[2].

SELECT: The syntax of a SELECT query is as follows:

- SELECT nominates which components of the matches made against the data should be returned.
- FROM (optional) indicates the sources for the data against which to find matches.
- WHERE defines graph patterns to match against the data.

[2] https://www.w3.org/TR/rdf-sparql-query/

Examples of SELECT

```
PREFIX dbr:<http://dbpedia.org/resource/>
PREFIX dbo:<http://dbpedia.org/ontology/>

SELECT DISTINCT ?name ?numChilds
WHERE{
?name dbo:numChilds ?numChilds
}
```

Listing 2.1: A formal SELECT clause has SELECT section, that can define the column of the result set, and a WHERE section, that can specify the result, in this query, it will get the name of the entities that has triples where relation is dbo:numChilds, and also return the obejct of these triples. DISTINCT is used to remove the duplicate of the result.

name	**numChilds**
dbr:Maria_Josepha_of Austria	11
dbr:Adelaide_of_Italy	0
dbr:Maria_Kunigunde_of_Saxony	0
...	...

Table 2.2: Result set from Listing 2.1

CONSTRUCT: The CONSTRUCT query returns a single RDF graph as a result, which is specified by a graph template, and can be parts or the whole of graphs from the target RDF dataset. It uses a set union to combine the result of the query solution from the clause section, and it defines the result template in triple format.

Examples of CONSTRUCT

```
PREFIX dbr:<http://dbpedia.org/resource/>
PREFIX dbo:<http://dbpedia.org/ontology/>
CONSTRUCT {
(?name dbo:numChilds ?numChilds).
(?name dbo:gender ?gender) }
WHERE{
?name dbo:numChilds ?numChilds.
OPTIONAL {?name dbo:gender ?gender}
```

```
}
```

Listing 2.2: A formal CONSTRUCT clause has CONSTRUCT section, that can define the structure of graph, cause the result is usually is the subgraph of the original graph, and a WHERE section with same purpose as above, the result will be like 2.1.3.

subject	predicate	object
dbr:Adela_of_Champagne	dbo:gender	”female”
dbr:Adela_of_Champagne	dbo:numChilds	2
dbr:Bivin_of_Gorze	dbo:gender	”male”
dbr:Bivin_of_Gorze	dbo:numChilds	0
...	...	...

Table 2.3: Result graph from Listing 2.2, it uses PREFIXes from the CONSTRUCT clause, and it is stored and demonstrated in TSV format.

ASK: A question pattern has only two possible answers: yes or no. Applications can utilize the ASK form to determine whether a question pattern has a solution. There is no information returned regarding the possible query solutions; only whether or not a solution exists.

Examples of ASK

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

Listing 2.3: A formal ASK clause has ASK section, that can define a query pattern and test whether has a solution, result with yes or no.

The input of the KGE models is RDF triples, i.e., ⟨subject, predicate, object⟩. The CONSTRUCT query is used to extract a RDF sub-graph from the full KG. As a result, the pipeline for training the KGE model and predicting missing links between entities becomes more efficient.

2.1.4 RDF Mapping Languages

R2RML

R2RML: RDB to RDF Mapping Language (R2RML) is a language used to customize mapping rules from relational databases to RDF data [3]. Based on W3C[4] recommendation language, the R2RML is expressed in RDF and composed of following syntax:

- *logicalTable* used to define the source, which could be a table or a valid SQL query.
- *subjectMap* defines the subject of the generated RDF triples; for one subjectMap. It may have multiple PredicateObjectMaps.
- *PredicateObjectMap* contains predicate and objectMap, aiming to define the predicate and the object of the triples.

This language is not designed to generate one single triple, but based on the TirplesMaps generates bunches of triples according to the table and also fits the format defined by the mapping rules. One example of the result could be found in Table 2.5, which is based on Listing 2.4 and Table 2.4.

Examples of R2RML

```
@prefix ex: <http://example.com/ns#>.
<Employee>
    rr:logicalTable [ rr:tableName "EMP" ];
    rr:subjectMap [
        rr:template "http://www.example.com/employee/{
           EMPID}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "NAME" ];
    ].
    rr:predicateObjectMap [
        rr:predicate ex:Department;
        rr:objectMap [ rr:column "Department" ];
    ].
```

Listing 2.4: R2RML-Example

EMPID	NAME	Gender	Department
0	Tom	"male"	Tech
1	Jhon	"male"	HR
2	Sam	"male"	Tech
3	Jack	"male"	Sale
...	...	...	...

Table 2.4: Relational Table *EMP*: works for Listing 2.4, RDF creator will base on the mapping rule, get data from this table to build a RDF graph, entire columns can be left out, depending only on the requirements of the mapping rules.

subject	predicate	object
ex:0	ex:name	Tom
ex:0	ex:Department	Tech
ex:1	ex:name	Jhon
ex:1	ex:Department	HR
...	...	...

Table 2.5: Part of created RDF graph base on Listing 2.4 and Table 2.4, It is stored and displayed in TSV format.

RML

RML: RDF Mapping Language (RML) [14] is an extension of R2RML, which is standard from W3C to express mappings from relational databases to RDF, and RML enable us to express mapping from heterogeneous data structures, i.e., JSON, CSV, XML, and relational databases.

Examples of RML mappling

```
<ModelDetails>
    rml:logicalSource [ rml:source "InterpretME/files/
       model_config.csv";
                        rml:referenceFormulation ql:CSV;
                        ];
```

```
    rr:subjectMap [
        rr:template "http://interpretme.org/entity/{
           run_id}_{model}";
        rr:class intr:Model
    ];
    rr:predicateObjectMap [
        rr:predicate intr:ModelName;
        rr:objectMap [
            rml:reference "model";
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate intr:hasEpochs;
        rr:objectMap [
            rml:reference "num_epochs";
        ]
    ];
```

Listing 2.5: RML mappling -Example The biggest difference between RML mapping R2RML mapping is it can be defined for data in any other source format, not only for Relational database. It is specified for this purpose.

In our work, we use RML to extract expect RDF triples from different data sources, like orignal data sources, hype-parameters in our pipeline, link predict result from KGE models, evaluation result provided by PyKEEN.

2.1.5 SDM-RDFizer

SDM-RDFizer [21] is a mapping rule interpreter that can turn unstructured data into RDF knowledge graphs. It allows the generation of RDF knowledge graphs from heterogeneous data sources. The SDM-RDFizer uses optimized data structures and relational algebra operations to enable the efficient execution of RML triple maps even in the presence of large amounts of data. SDM-RDFizer can handle data from a variety of sources (CSV, JSON, RDB, and XML), processing each set of RML rules (TriplesMap) in a multi-thread safe approach. The SDM-RDFizer demonstrates the advantages of establishing solutions for the problem of knowledge graph construction in well-established fields such as data integration systems and query processing. SDM-RDFizer was made accessible as a resource due to the requirement for efficient

knowledge graph development in commercial and research applications.

SDM-RDFizer combines entities acquired during ML model training, as well as all traceable metadata from the InterpretME pipeline, into the InterpretME KG as factual assertions. The traced information from InterpretME contains SPARQL endpoints, feature definitions in the input KG, SHACL validation reports for entities in the input KG, predictive model features and outcomes, LIME entity interpretations, and so on. The InterpretME KG documents the learned ML behavior through both machine and human statements. Traceability is ensured by aligning entities in the InterpretME and input KGs. The GitHub [5] repository has a full overview of the usage and actions to take when using SDM-RDFizer.

2.2 Translation Family of Knowledge Graph Embedding Models

Given a Knowledge Graph (KG), defined as a directed edge-labeled graph KG=(V,E,L). A KGE models [4] transform nodes and edges in G into a low-dimensional continuous vector space that preserves the structure of G. There has been widespread adoption of KGs, which store knowledge in machine-driven formats. KGs are typically noisy and incomplete, several methods have been developed to predict new relationships. Specifically, it is described as predicting the head/tail entities for $(h, r, ?)/(?, r, t)$ predicting missing relationships between entities in the KG.

> **Plausibility scores:** Given a directed edge-labeled graph G = (V,E,L), a set of vectors T, and a knowledge graph embedding (ϵ, p) of G. A plausibility score function ϕ, is a partial function $\mathbb{T}$ x $\mathbb{T}$ x $\mathbb{T} \Rightarrow \mathbb{R}$, such that for a triple t=(s p o) in V x L x V, $\phi(\epsilon(s), \rho(p), \epsilon(o))$ computes the plausibility of t.

KGE models, given the embedding vectors for the entities and relations of the KG, calculate a score expressing the plausibility of a triple. Assigning high values of scores to triples that are probably true and low scores to triples that are probably false is the aim of a plausibility score function. The objective of a plausibility score function is to assign high values of ϕ to triples that are likely to be true and low scores to triples that are likely to be false.

KGE models learn latent vector representations of the entities $e \in \varepsilon$ and relations $r \in \mathcal{R}$ in a KG that best preserve its structural properties. The objective of a knowledge graph embedding model is to learn the embeddings in (ϵ, p) that maximize the plausibility of positive edges in E^+ and minimize to the edges the plausibility of

[5] https://github.com/SDM-TIB/SDM-RDFizer

negative edges in E^-.

- The set of positive edges in G, E^+, correspond to the edges in E.
- The set of negative edges in G, E^-, correspond to the edges in $\mathcal{V}$ x $\mathcal{L}$ x $\mathcal{V}$ that are not in E+.

TransE, TransH, TransR, and TransD are among the KGE models that are part of the translation family. The euclidean lengths of the subject and object are used by the translation-based embedding models to convert the KG entities into low-dimensional vector space.

TransE

TransE is a representative model of translational distance models [7, 39] and one of the most significant models of knowledge graph embedding. TransE converts a knowledge graph's entities and relations into a low-dimensional vector space. A 2-dimensional vector space example of a head entity and relation translation to a tail entity is shown in Figure 2.4. Nevertheless, such learnt vector embeddings capture and maintain the semantic relationships of entities in a KG. Entities and relations are represented as vectors in the same space, TransE can model relations as a translation of head-to-tail embeddings, for a correct ternary (h, r, t), the embedding vector of the head entity plus the embedding vector of the relation should be approximately equal to the embedding vector of the tail entity, i.e., $\mathbf{h}+\mathbf{r} \approx \mathbf{t}$. Thus, the scoring function is defined as:

$$f(h,r,t) = -\|\boldsymbol{h}+\boldsymbol{r}-\boldsymbol{t}\|_p \tag{1}$$

with p $\in$ 1,2, and the score is expected to be large, when the triple is true.

TransE can be used for large-scale KGs. However, it inherently cannot model 1-N, N-1 and N-M relations: assume(h,r,t_1), $(h,r,t_2)\in$ K, then the model adapts the embeddings to ensure $\boldsymbol{h}+\boldsymbol{r} \approx \boldsymbol{t_1}$ and $\boldsymbol{h}+\boldsymbol{r} \approx \boldsymbol{t_2}$, cause h, r, and t_1 t_2 in the same vector space, if h and r are determined, then $h+r=\mathbf{t}$ t is determined, with $\mathbf{t} \approx t_1$ and $\mathbf{t} \approx t_2$ lead result to $\boldsymbol{t}_1 \approx \boldsymbol{t}_2$, in many scenarios result in $\boldsymbol{t}_1 \approx \boldsymbol{t}_2$ is not expected, so while TransE performs well in some scenarios, it still has its limitations [39]. For exmaple, one person may have two children, this two children could not be counted as same or close, while they have a lot of similarities, they will theoretically have a lot of differences, such as gender, and for royalty, some have succession rights and some don't, which makes the gap between two children of the same parents sometimes wide, but these are blurred by the characteristic of the TransE.

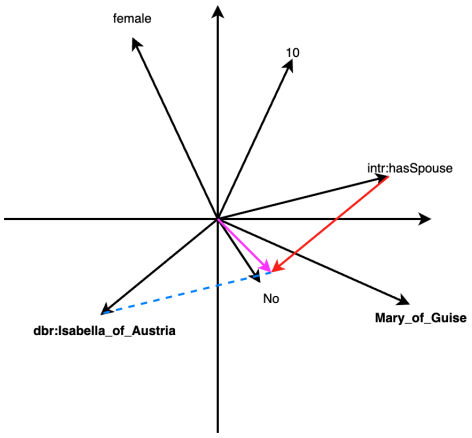


Figure 2.4: The figure demonstrates that the embeddings generated by the TransE model, and presented as vectors in 2-dimensional space, and with this data, it is possible to work on a link prediction task. In which, for LP task (dbr:Isabella_of_Austria, intr:hasSpouse, ?)the vector of target entity dbr:Isabella_of_Austria can add vector of relation intr:hasSpouse, and get a tail, which is approximately equal to literal "No".

TransH

TransH [41] Figure 2.5 is an extension of TransE that specifically addresses the limitations of TransE in modeling 1-N, N-1, N-M relations. In TransH, each relation is represented by a hyperplane, or more specifically a normal vector of this hyperplane $\boldsymbol{W}_r \in \mathrm{R}$, and a vector $\boldsymbol{d}_r$. To compute the plausibility of a triple, the head embedding and the tail embedding are first projected onto the relation-specific hyperplane, by using $\boldsymbol{h}_r = \boldsymbol{h} - \boldsymbol{W}_r^\intercal \boldsymbol{h} \boldsymbol{W}_r$ and $\boldsymbol{t}_r = \boldsymbol{t} - \boldsymbol{W}_r^\intercal \boldsymbol{t} \boldsymbol{W}_r$.

After the projection, we get $\boldsymbol{h}_r$ and $\boldsymbol{t}_r$, and the relation itself exists on the hyperplane corresponding to it, i.e., $\boldsymbol{d}_r$, which has similar operations with TransE. The projection of the head entity $\boldsymbol{h}_r$ add the relation $\boldsymbol{d}_r$ should be approximately equal to projection of the tail entity $\boldsymbol{h}_r$.

$$f(h, r, t) = -\|\boldsymbol{h}_r + \boldsymbol{d}_r + \boldsymbol{t}_r\|_p \tag{2}$$

In this case, $\boldsymbol{t}_{r1}$ and $\boldsymbol{t}_{r1}$ will be similar, but $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ could have many difference before projected to the hyperplane.

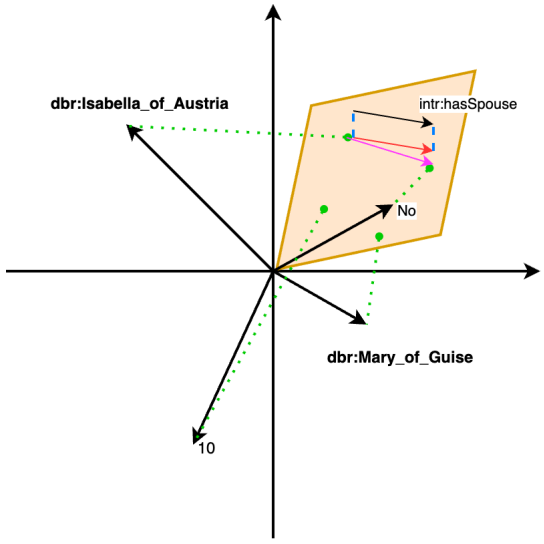


Figure 2.5: The figure demonstrates the embeddings generated by the TransH model, relations are all presented in a hyperplane, and all entities will be projected into this hyperplane, and find out which entity is close to the target entity add relation.

TransR

TransR [28] is a further extension of TransH, that no more put all the entities and relations from the KG within the same semantic space, it explicitly considers entities and relations as different objects and therefore represents them in different vector spaces. For a triple(h,r,t) the vector embeddings of the entities, h,t $\in$ R , are first projected into the relation space by means of a relation-specific projection matrix $\boldsymbol{M}_r$: $\boldsymbol{h}_r = \boldsymbol{M}_r\boldsymbol{h}$ and $\boldsymbol{t}_r = \boldsymbol{M}_r\boldsymbol{t}$. r is on the relation space and with $\boldsymbol{h}_r$ and $\boldsymbol{t}_r$, we could repeat the approximately equation, i.e., $\boldsymbol{h}_r + r \approx \boldsymbol{t}_r$. corresponding scoring function is as follow:

$$f(h, r, t) = -\|\boldsymbol{h}_r + \boldsymbol{r} + \boldsymbol{t}_r\|_p \tag{3}$$

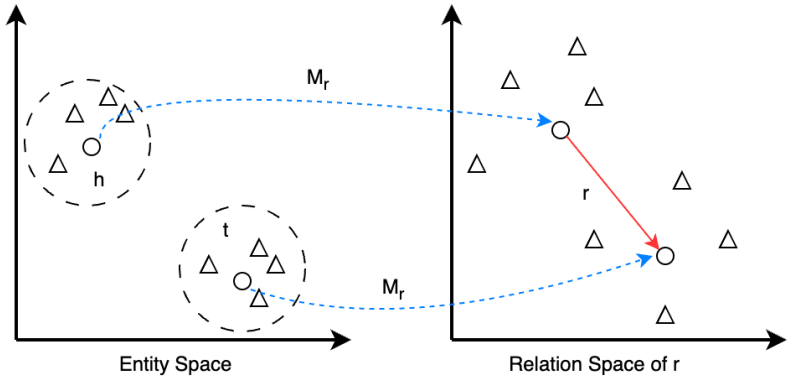


Figure 2.6: The figure demonstrates that the embeddings generated by the TransR model, will generate relation space for each relation, and the entity will be projected to the relation space to explore the relation between two entities.

TransD

TransD [22] is improved from TransR. It also considers entities and relations as objects existing in different vector spaces. The example could be found in Section 2.2. However, instead of performing the same relation-specific projection for all entity embeddings, entities are separated into head entities and tail entities. and use different matrics Equation 4 and Equation 5 to project the entity to the relation space. After the projection, TransD takes the same action as TransE.

$$\boldsymbol{M}_{rh} = \boldsymbol{r}_p \boldsymbol{h}_p^\top + \boldsymbol{I}^{m \times n} \tag{4}$$

$$\boldsymbol{M}_{rt} = \boldsymbol{r}_p \boldsymbol{t}_p^\top + \boldsymbol{I}^{m \times n} \tag{5}$$

$$\boldsymbol{h}_\perp = \boldsymbol{M}_{rh} \boldsymbol{h} \tag{6}$$

$$\boldsymbol{t}_\perp = \boldsymbol{M}_{rt} \boldsymbol{t} \tag{7}$$

$$f(h, r, t) = -\|\boldsymbol{h}_\perp + \boldsymbol{r} + \boldsymbol{t}_\perp\|_2^2 \tag{8}$$

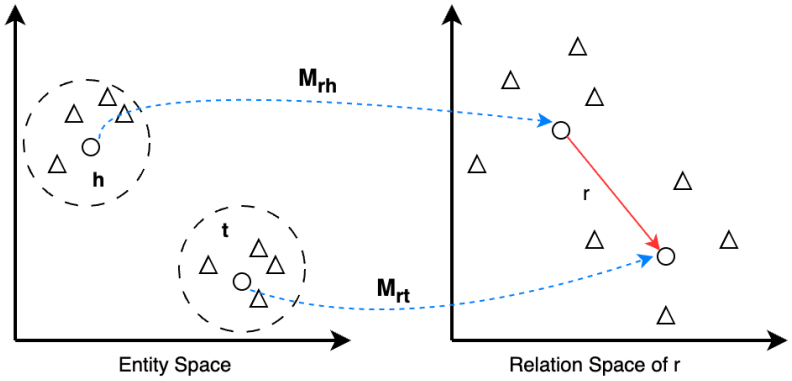


Figure 2.7: **Demonstration of TransD** The biggest difference between TransR and TransD is that TransD considers head and tail as separated entities, and proposes two mapping matrix $\boldsymbol{M}_{rh}$ and $\boldsymbol{M}_{rt}$ to help project the entities into relation space.

2.2.1 Evaluation Metrics

Metrics are a set of standards for evaluating the performance of things or a model in this book. It enables the model to be evaluated in the same way and under the same criteria. For example, experiments are conducted in the same environment, the same dataset is trained and evaluated, or the results are computed using the same set of formulas to understand their performance or to compare them with relevant other models, i.e., MRR, Hits@10 and so on.

MRR: MRR is a metric that measures the performance of the ranking task. It will show the speed of retrieving the expected answers from queries. Reciprocal rank (RR) is used to retrieve the ranking of the first relevant or correct result from the result-ranked list and get its reciprocal. MRR will repeat several times of RR query and calculate their averages. It tends to be more affected by low-ranking values [32].

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i} \tag{9}$$

where N is the number of queries, and rank_i is the rank of the correct answer in the i-th query. The higher the MRR, the better the system is at placing relevant results at the top of the list, and queries can also faster to run and get the expect result.

Example of MRR:

- Query 1: Correct answer ranked 4.
- Query 2: Correct answer ranked 3.
- Query 3: Correct answer ranked 2.

The MRR for these queries is calculated as follows:

$$\text{MRR} = \frac{1}{3}\left(\frac{1}{4} + \frac{1}{3} + \frac{1}{2}\right) = \frac{1}{6} \approx 0.17$$

Hits@k: Hits@k is a metric that focuses more on the performance of the system in accuracy, which k could be 1, 3, 10, and so on, to measure the correct answer hit in Top-K.

Example of Hits@10:

- Query 1: Found expected to result in the Top 10 of the result list.
- Query 2: Found expected to result in Top 10 of the result list.
- Query 3: Did not find an expected result in Top 10 of the result list.

The calculation of HIT@10 is:

This means that 2 out of 3 users found correct result in the Top-10.

2.3 Symbolic Learning

Symbolic Learning is a form of inductive learning technique commonly used over KGs. In contrast to many other techniques, symbolic learning focuses on reasoning and logic-based methods to acquire and represent knowledge. Knowledge graph embeddings are numerical models of knowledge graphs. However, these models are often difficult to explain or comprehend. These numerical models are used to predict edges between nodes in KGs, but they cannot provide explanations or justifications for the predictions made. Rule mining is a self-supervised symbolic learning tech-

nique that involves identifying significant patterns in the form of rules from massive collections of background knowledge. The purpose of rule mining is to discover new rules with a high ratio of positive edges and a low ratio of negative edges.

$$B_1 \wedge B_2 \wedge ..... \wedge B_n \implies r(x, y) \tag{10}$$

In the above equation, $r(x, y)$ represents the head of the rule and $B_1 \wedge B_2 \wedge ... \wedge B_n$ represents the body of the rule. The horn rule claims that if all of the body atoms occur in the KG, the head atom can be determined.

Mined Rules:
Body**:-** Head
Body is a conjunction of predicate facts; Head is a predicate fact:
dbo:gender(X,"male"@en),dbo:parent(Y,X)**:-**dbo:father(Y,X)

Example of Mined Rules:

dbo:gender(dbr:Sancho_I_of_Portugal,"male"@en),
dbo:parent(dbr:Sancho_I_of_Portugal, dbr:Afonso_II_of_Portugal)
:-
dbo:father(dbr:Sancho_I_of_Portugal, dbr:Afonso_II_of_Portugal)

The body of the above mentioned example rule states that the gender of `dbr:Sancho_I_of_Portugal` is male and his parent is `dbr:Afonso_II_of_Portugal` which implies the head of the rule `dbr:Sancho_I_of_Portugal` has father `dbr:Afonso_II_of_Portugal`

This thesis will use the self-supervised approach known as Rule Mining, which falls under the category of Symbolic Learning. Rule Mining can be defined as follows: Given a directed edge-labeled graph $G = (V, E, L)$, E^+ set of positive edges which correspond to the edges in E. E^- set of negative edges corresponding to the triples $(s^{'}, p^{'}, o^{'}) \in (V, E, L) - E$. Several metrics like *Support*, *Confidence* and *PCA Confidence* scores are computed in the rule mining process.

2.4 Summary of the Chapter

Some fundamental ideas utilized in this master's thesis are introduced in this chapter. The first section covers the fundamental ideas of Scientific Data Management and KG, which aids in understanding the pipeline's workflow. RDF and KG are the foundational elements that allow entities to carry metadata and be enhanced with semantic context; these elements form the basis of the KGE models. SPARQL is the standard query language that permits users to query data from any data source that can be mapped to RDF, including databases. In order to search the KG and extract appropriate subgraphs for KGE models, SELECT and CONSTRUCT queries are employed. InterpretME allows tracing the metadata at different stages of the predictive model pipeline. The RML mappings are executed over the RDFizer and are used to create InterpretME KG that contains all the traced metadata. The second part will introduce approaches related to Inductive Knowledge, which focuses on TransFamily and symbolic learning, The TransFamily model is part of KGE models, which contain TransE, TransH, TransR, TransD, and several metrics to measure the performance of the model, symbolic learning is based on rule mining to enrich the incomplete KGs. This enriched KG will have an impact on TransFamily models.

Chapter 3

Related Work

To improve the interpretability of machine learning (ML) models, researchers have proposed a variety of methods and tools in recent years. Among them, LIME, SHAP, and *InterpretME* are three approaches that have attracted much attention [12, 29, 33]. LIME focuses on model-independent local explanations by generating interpretable graphical representations of individual samples by generating interpretable graphs. In contrast, SHAP is based on Shapley values and provides a globally consistent interpretation, allowing the user to understand the characteristic contributions of the entire model. However, these two methods have limitations; their outputs are not machine-readable, and they also remain difficult for humans to interpret in the absence of context. InterpretME bridges the gap between data value and predictive modeling by making it easier to describe predictive model conclusions using factual statements and by establishing relationships to the application domain KG.

3.1 LIME

Predictions of black-box models are often difficult to interpret, which makes it difficult for users to trust the model's decisions. Users are interested in learning more about the model's methodology, particularly when it comes to application scenarios that demand a high degree of interpretability. The core idea of LIME is to approximate the decision boundaries of the model by generating local neighborhood samples in the input space and observing the predictions of these neighborhood samples in the black-box model. LIME emphasizes its model-independence, i.e., it can be used to interpret any classifier without being constrained by the type and structure of the model itself. This is achieved by generating locally interpretable models that do not necessarily have to be the same as the original model. The idea of LIME is

to construct an interpretable local model that explains the model's predictions on a particular instance by generating local perturbations in the input space and observing the model's behavior around these perturbations. Figure Figure 3.1shows the interpreted graphical representations generated by LIME, which visualize which features in the input space play a key role in the model's predictions. The paper points out that LIME can be applied in a variety of domains, including healthcare, finance, and other domains that require high levels of model interpretability. It improves user confidence in the model and facilitates practical application of the model.

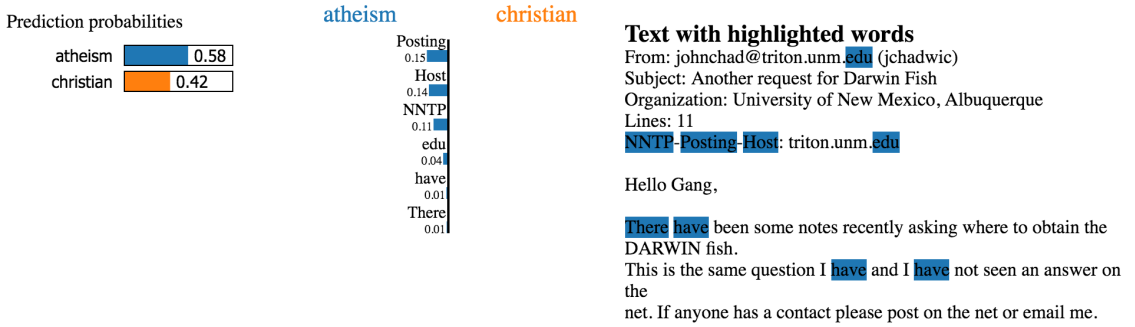


Figure 3.1: An example of LIME, where colorful marks and weights highlight the features, that make the classifier predict the writer as atheism or Christian.

3.2 SHAP

SHAP is a framework for analyzing the outputs of machine learning models Figure 3.2. It is based on game theory's Shapley values and is used to determine how much each characteristic adds to the model output. The Shapley values represent the contribution of each feature to the model output considering all possible combinations of features. SHAP not only provides an interpretation of individual samples but also can compute an average Shapley value over the entire set of features, thus providing a global model interpretation. This helps to understand the impact of different features on the overall model behavior. Horizontal bar plots (bar plots) or Hammington circle plots (force plots) are used to visualize the impact of each feature on the model output. Similar to LIME, SHAP is model-agnostic and is suitable for a variety of machine learning models, including deep and integrated learning.

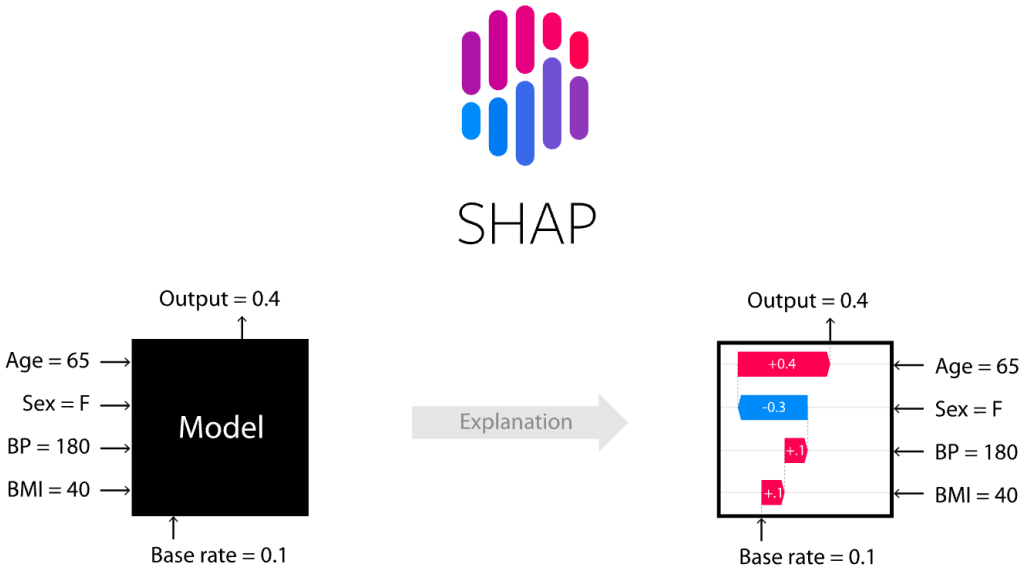


Figure 3.2: An example of SHAP, shows weights of input features, which makes a black box model have more explanation.

3.3 InterpretME

InterpretME[12] Figure 3.3 is an analytical tool for tracing and explaining predictive models created using data from KGs, CSV, and JSON files. InterpretME is a standalone framework that works on independent systems. Multiple runs can be integrated into a single InterpretME KG for comparing interpretations. InterpretME combines machine learning algorithms with features from KG to provide rich and relevant insights. It improves the model's interpretability, assists the user, and increases efficiency by integrating various forms of information from the trained model into the KG. InterpretME examines SHACL constraints across input KG nodes and generates validation reports for each constraint and target entity. These results demonstrate if an entity validates or invalidates the restrictions set by the user. Validation reports state the validity of data used by predictive models. Query processing allows for comparing the predic-

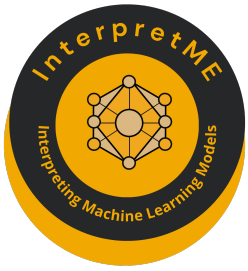


Figure 3.3: InterpretME

tion model's attributes in the InterpretME KG to those of individual entities in the input KGs. Using query federation, users can interpret results for a specific entity based on input properties that match SHACL validation results.

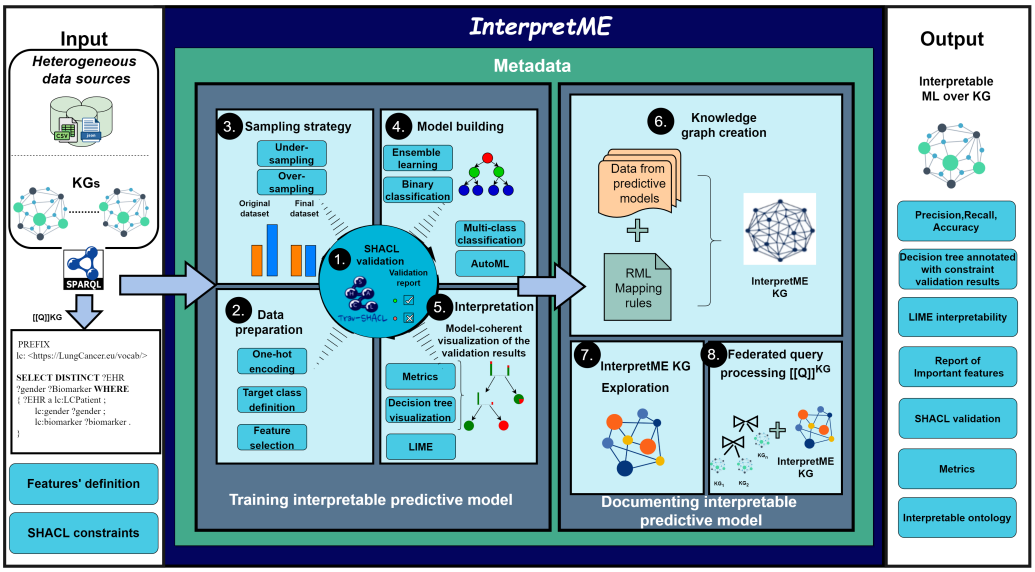


Figure 3.4: The InterpretME architecture displays input as either KGs or datasets in formats such as CSV and JSON. Training interpretable predictive model components, including SHACL validation, data preparation, and sampling, are used to prepare data for predictive models. The data is subsequently supplied to the model-building component, which performs predictive modeling tasks. To document interpretable predictive models, generate and explore the *InterpretME* KG to acquire insights.

InterpretME has two primary quadrants Figure 3.4[6]. *Training interpretable predictive model* includes evaluating SHACL constraints over nodes in the input KG and creates output in the form of validation report indicating valid or invalid entities. Furthermore, the input data is preprocessed into the form required for predictive model training. The data preparation phase includes sampling strategy decisions, one-hot encoding, and target class determination; the model creation and AutoML-based model optimization in the model selection phase; and the predictive model is then executed over the preprocessed data. The interpretation component generates visualizations in the form of decision trees connected with the SHACL restrictions, as well as visualizations of interpretable tools such as LIME for local explanations to show the contribution of each feature from the input KG to the prediction.

[6]https://github.com/SDMTIB/InterpretME/blob/main/images/architecture.png

The second quadrant, *Documenting interpretable predictive model*, uses all tracked metadata throughout the process, from identifying important features from input data to training predictive models over it. RDF mapping language RML is utilized to transform the traced metadata into triples of the form ⟨`subject, predicate, object`⟩. RML mapping rules are used to generate *InterpretME* KG. The *InterpretME* KG is used for statistical analysis of a target item or to understand why it was categorized into a specific class by the prediction model. *InterpretME* offers to explore multiple KGs by executing a SPARQL query over the federation of KGs to gain more clear human and machine understandable insights. Furthermore, several metrics such as precision, recall, and accuracy, as well as well-presented LIME interpretations, provide users with a more intuitive understanding of the model's performance. Experiments with InterpretME illustrate that tracing metadata and data utilized within the pipeline can improve the interpretability of predictive models.

3.4 AutoML Tools

The purpose of Automated Machine Learning (AutoML) tools is to eliminate the barriers so that non-experts may utilize machine learning techniques without understanding the underlying technical complexities. It can also increase effectiveness by automating model construction and eliminating human involvement, as well as improving model performance by systematically searching and optimizing all aspects of model building to find superior model configurations. This involves things like data preparation, feature engineering, selecting models, and hyperparameter tuning. The fast growth of complexity observed in machine-learning applications has created a demand for automated machine-learning algorithms that are easy to use without the requirement of expert knowledge. Auto-WEKA [25] and auto-sklearn [16] are two state-of-the-art AutoML tools that assist non-expert users with data processing, feature selection, model construction, and other tasks. These programs can process these works automatically, providing users with a good ML model and results.

AutoML provides Hyperparameter Optimization (HPO) tools for non-experts to optimize a model's hyperparameters during training. *InterpretME* uses *Optuna* [1], a hyperparameter optimization framework intended specifically for adjusting hyperparameters for all ML models. Optuna is a next-generation hyperparameter optimization application, which only focuses on part of AutoML principles [1]. It allows users to search the parameter space dynamically, and use several searching and pruning strategies to get a higher efficiency in running a predictive model, and also designed with the idea of easy-to-setup, versatile architecture for several non-expert users.

PyKEEN has a hyperparameter optimization function that uses Optuna on the

backend for tuning hyperparameter optimization. PyKEENs hpo_pipeline() performs a grid, random, or similar search across the supplied model parameters and produces a tuned hyperparameter list. There are many other types of hyperparameters offered by hpo_pipeline(), which can be used to optimize the model fit. Hyper-parameter optimization is most commonly used to specify the dataset, model, and number of runs. The hyper-parameter optimization pipeline can optimize hyper-parameters for their respective arguments provided in the `pykeen.pipeline.pipeline()`. The following example demonstrates how to use the n_trials option to optimize the TransE model on the YAGO3 dataset many times.

Example of PyKEENs Hyper-parameter Optimization:

```
from pykeen.hpo import hpo_pipeline
hpo_pipeline_result = hpo_pipeline (
                                    n_trials = 30,
                                    dataset = "YAGO3",
                                    model = "TransE",
                                    )
```

Currently, this thesis uses PyKEENs HPO, which can be converted in the future to use Optuna to be more efficient with the implementation of *InterpretME*.

3.5 Summary of the Chapter

This chapter discussed the state-of-the-art approaches or tools for supporting interpretability. *SHAP* includes feature contributions for each instance, global explanations, and feature importance, whereas The *LIME* technique uses local surrogate models to explain particular pipeline predictions. LIME provides explanations for target entities that are easy to understand. However, the explanations are not machine-readable and cannot be translated for the domain application. *InterpretME* addresses these limitations by providing fine-grained representations of local interpretations related to target entities in the domain application KGs enhancing the interpretability. *AutoML* simplifies working with ML models, and the hyperparameter optimization method can help users achieve an efficient model.

Chapter 4

Tracing KGE Models over InterpretME

This chapter shows the problems faced by existing KGE model applications and the needs of ordinary users and introduces technologies and proposed approaches used to solve these problems, including PyKEEN, and SDM-RDFizer [5, 21]. Finally, shows how the problem is tackled.

4.1 Problem Statement

Consider a directed edge labeled graph G, and the directed edge is represented as triple (subject, predicate, object) and statement of fact, in which, each node in the graph G represents a subject or object (in a directed edge, node entity in the head is considered as subject, node entity in the tail is considered as object), each edge in the graph G represents a relationship.
Given a link prediction $(s, p, ?)$ where s is a head entity in G, p is the relationship between entities, and $(s, p, o) \notin G$. The goal is to find a set of most plausible triples by inferring new triples based on the existing triples in G, and provides an interpretability-improvable set of triples I that lead the black-box model to predict tail Figure 4.1. Unlike the previous method, it uses a result-oriented approach to provide explanations.

In comparison to baseline ML methods, our goal is to construct a robust pipeline for both utilizing the model and systematically evaluating the performance of KGE models. The emphasis is on enhancing interpretability to empower users with a deeper understanding of model decisions.

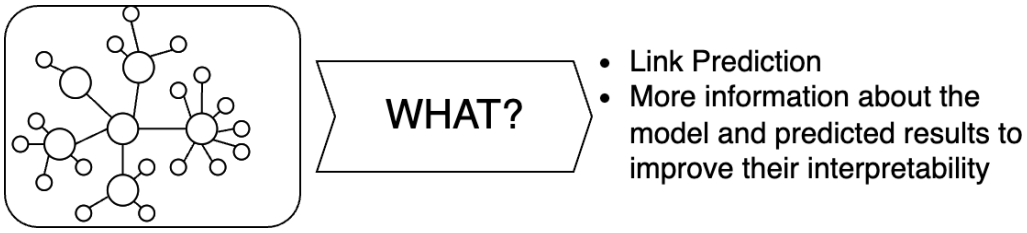


Figure 4.1: An RDF knowledge graph as input, and as output requires finish the link prediction task and also a better understanding of the process and result, what approaches could achieve this purpose.

4.2 Proposed Solution

The data and metadata involved in multiple steps can be traced and documented, later represented in a specific way to help users better understand how the model works and the performance of the model, and the interpretability of the model could be enhanced, which can be tracked to eventually generate an RDF that correlates this data with the model and experiments.

4.2.1 Tracing Inputs

Input information plays an important role in model decision-making, and the rich metadata stored in the RDF graph, as well as the full structure of the information (compared to the subgraph obtained through the CONSTRUCT clause, KG from the endpoint has 132 relations, where the subgraph only only has 10), which preserves the data source information for subsequent repeated access, can be very effective in explaining the experimental decision-making process.
Due to the variety of input data, such as data sources accessed through endpoints or locally stored TSV file formats. This information must be tracked in an appropriate manner to ensure that the data is captured correctly when the data source is accessed again.

4.2.2 Tracing Data Preparation

In the data preparation phase, the CONSTRUCT clause is utilized to extract the portion of the KG that includes entities and properties to be included in the KG. Additionally, metadata is retrieved, which helps explain the model decision. Storing CONSTRUCT clauses in a decomposed way like in the Listing 4.1 allows humans

and machines to have a better understanding of the content, and quick refactoring to reproduce the clauses and generate SPARQL query to access the endpoint.

Data Preparation Example

```
"Independent_variable": {
      "person": "?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia
          .org/ontology/Person> ."

    },
"Independent_Optional_variable": {
      "father": "?person <http://dbpedia.org/ontology/father> ?father.",
      "mother": "?person <http://dbpedia.org/ontology/mother> ?mother.",
      "parent": "?person <http://dbpedia.org/ontology/parent> ?parent.",
      "name": "?person <http://xmlns.com/foaf/0.1/name> ?name.",
      "successor": "?person <http://dbpedia.org/ontology/successor> ?successor.",
      "child": "?person <http://dbpedia.org/ontology/child> ?child .",
      "spouse": "?person <http://dbpedia.org/ontology/spouse> ?spouse .",
      "gender":"?person <http://dbpedia.org/ontology/gender> ?gender ."
  },
"Dependent_variable_subject": "person",
"Dependent_variable": {
      "hasSpouse": " ?person <http://dbpedia.org/ontology/numSpouses> ?numspouses.
      BIND(IF(?numspouses>0 ,\"Yes\",\"No\" )as ?hasSpouse )"
    },
```

Listing 4.1: Elements from CONSTRUCT clauses are stored in a decomposed way, and the independent variable aims to set the main structure of the subgraph, it enables to query of all human beings from the endpoint, and the optional variable is used to provide more information but is not necessary. The dependent variable is used to set the prediction target. This allows users to a better understand the CONSTRUCT clause and provides a basis for quickly generating SPARQL queries.

4.2.3 Tracing HPO and Model Building

Appropriate hyperparameters can make the model perform better in prediction tasks. The *optuna* framework provides an open source automatic hyperparameter optimization software framework [2], this framework uses the study to keep track of the best set of hyperparameters while exploring different trials with different hyperparameter combinations and uses pruning to enable early stop and make the optimization process more efficient. PyKEEN integrates this framework into its library, enabling users to acquire suitable hyperparameters before model training. The model-building process predominantly relies on hyperparameters from the preceding step. However, because multiple models are used concurrently in the pipeline, and the hyperparameters vary for each training iteration due to diverse data sources, it becomes essential to precisely monitor and record each set of hyperparameters corresponding to every model during training. By recording all relevant hyperparameters, Users could gain a

better understanding of the model built when reviewing this data in some scenarios.

4.2.4 Tracing Model Evaluation

Metrics serve as indicators of model performance during the evaluation phase. By calculating and ranking the results, the evaluation framework computes MRR and Hits@10 for performance assessment. Since the outcomes vary with each model training, it is necessary to systematically track and correlate the evaluation results from the current training with those of the model under current training. This ensures accurate and efficient analysis of the evolving performance metrics.

4.2.5 Building a Comprehensive Pipeline

We propose a pipeline for documenting KGE models. This pipeline encompasses the phases of data preparation, model building, model training, model evaluation, and output RDFization. It facilitates the execution and tracking of the entire model process while efficiently utilizing memory resources. Additionally, it allows for the preservation of selected data along with the ongoing processes. The pipeline generates local files for tracking key information. In the final stage of the pipeline, all pertinent information is categorized and saved in designated CSV files for RDFization, contributing to the construction of the InterpretME Knowledge Graph (KG). The information tracked includes facts represented as triples $T_1(s_1, p_1, o_1)$. Here, T_1 is not in the existing graph G but is part of the desired information set I. This information aids the black-box model in predicting the tail, enhancing the interpretability of each round of the model.

Figure 4.2 shows the sequential process described in the motivating example. The process starts with an Endpoint or a local dataset, which uses the CONSTRUCT clause from SPARQL to extract a target subgraph as a dataset (which is small and content is more specific, better for model training). The dataset is used to train and evaluate the KGE model. After that, we could get embedding vectors as a result. With the Score function, it will enable us to Calculate the triple tables to determine which triple is most likely to be true. During all these processes, we collect their relevant information, e.g., the URL of Endpoint, the SPARQL clause used to query the result from Endpoint, predefined hype parameters used in the KGE model, which model will be used, and so on. After we get embedding vectors, we use RDFizer[7] and defined mappings to create a new Knowledge Graph and enable it in an Endpoint

[7] https://github.com/SDM-TIB/SDM-RDFizer

to make specific clauses to help us better understand the result, or use the result to compare the performance of the datasets.

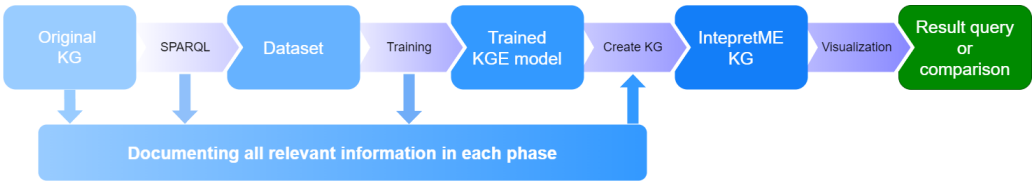


Figure 4.2: This diagram illustrates the pipeline workflow, which involves not only model-related work, but more importantly, tracing and documenting the data during the pipeline. To get a rich context to enhance the interpretability of the model.

4.3 Summary of the Chapter

This chapter addresses the challenges of improving the interpretability of KGE models. It presents a pipeline to document these models by tracing the transformations conducted by the models. The pipeline stores relevant information for further analysis and it is part of the InterpretME framework.

Chapter 5

Implementation

This chapter focuses on the implementation of the translation distance models (e.g., TransD) and hyperparameters optimization in the InterpretME framework for tracing the inner mechanism of the KGE models. The integration of the KGE models into the existing code is of particular interest. Figure 5.2 describes in detailed the implemented functions for documenting and tracing the KGE models. These semantic-driven interpretations provide a description of a target entity with the model characteristics. Also, encompasses the challenges involve in integration of the KGE models with InterpretME framework. In general, Figure 5.1 shows a simplest application of the KGE model where a RDF KG is required as input, and as output, the model can result in a embedding learning that can be used to predict links or other tasks such as node classification, and entity clustering. But the simplest application seems simple, certain requirements hinder the usage of the existing KGE models. As aforementioned in Chapter 4, the pipeline includes the components (i.e., data preparation, hyperparameter optimization, model buidling, training and evaluation, KG creation, statistical analysis over the InterpretME KG) defined in terms of the inputs and outputs. Figure 5.2 illustrates our trace KGE pipeline over InterpretME framework.

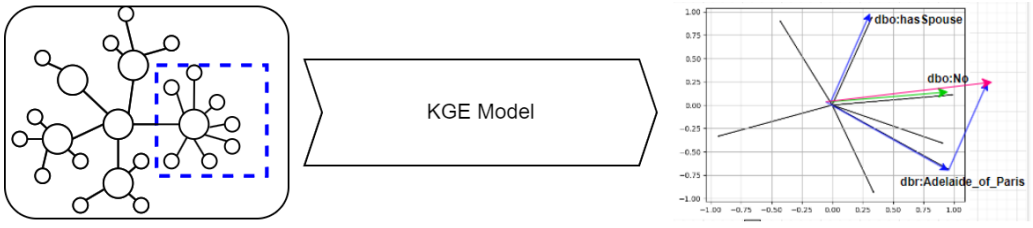


Figure 5.1: Shows the simplest application of the KGE model.

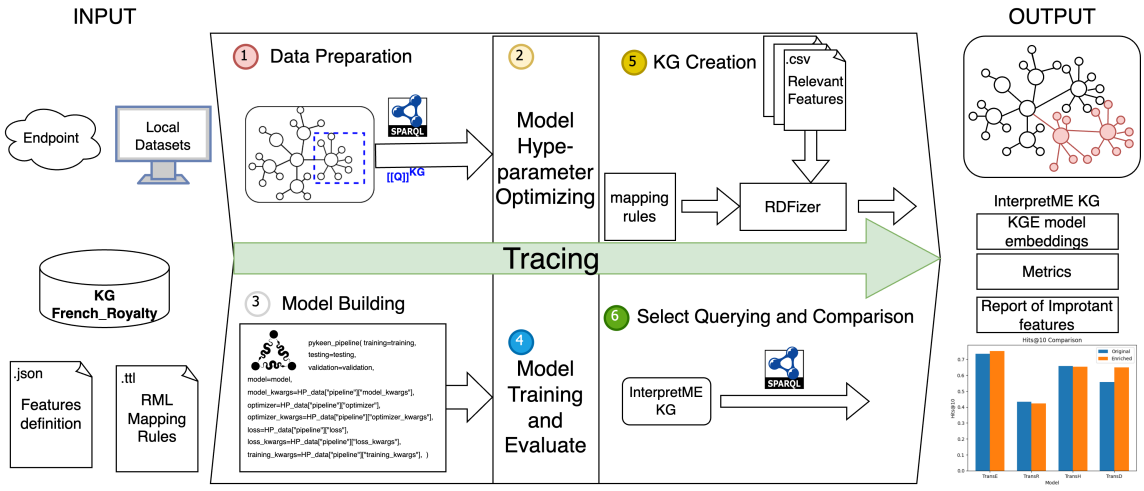


Figure 5.2: **InterpretME pipeline** for tracing the KGE Models. Firstly, the pipeline retrieves the required data, then hyperparameters optimizing is deployed. Further, model building is obtain to train and evaluate the model. During execution of the pipeline, each step is traced and stored as metadata. Lastly, these stored metadata with RML mapping rules is utilized to generate the InterpretME KG.

① **Data Preparation.** The target to deal with inputs of heterogeneous formats and storage environments. For example, when the data source for the Endpoint, or data source to TSV, CSV, or JSON format is stored locally, the simple input as a uniform format in this case is no longer applicable to the need to use the project to be able to flexibly deal with different input.

② **Model Hyperparameter Optimizing** is obtained to build the KGE model with the best parameters (e.g., batch size, embedding dimensions, etc.). Here, the process of hyperparameter optimization is independent of the process of training and evaluation of the model. Moreover, ③ **Model Building** collects the knowledge regarding the KGE model with their best hyperparameters. Currently, InterpretME is integrated with the translation distance models, such as TransE, TransH, TransD, and TransR. Based on the user's preference the type of KGE model is obtained for training in ④ **Model Training and Evaluate**. For ordinary users, this is a very complex process and some operations are difficult to understand and implement from the state of art approaches. Therefore, the implementation of our pipeline overcomes these limitations and simplifies the whole process of training the KGE model.

⑤ **Knowledge Graph creation.** InterpretME traces the KGE models' characteristics at each step of the pipeline and store it as metadata. Here, the RML

mapping rules are obtained as described in Listing 5.5. Using an efficient RML engine, RDFizer, to generate the semantic representation of the metadata. These semantic representation, we term as InterpretME KG, which contain the contextual knowledge about each target entity's behavior in the KGE model training and evaluation.

⑥ **InterpretME KG**. For the performance of the model results and the decision-making process, it is difficult for ordinary users to understand and interpret the model outcomes with evaluation metrics, such as MRR and Hits@10. While, InterpretME pipeline facilitates federated query processing as a module to extract the input and model characteristics of a target entity with better contextual interpretations. Additionally, these statistical analysis of different KGE model's comparisons is visualized with bar, line and radar plot using `InterpretME.plot()` function.

5.1 Data Preparation

This section describes how to handle the main data inputs. There are two main schemas: locally stored data in TSV, N-triples format and RDF sub-graphs retrieved from the SPARQL endpoint. The user defines the required features in the JSON format document for extracting data either locally or endpoint, and the operation will redirect the pipeline to a different data processing class. The pseudo-code to control this process can be found in Listing 5.1. The code shows that this process is controlled by the contents of the input JSON, i.e., by the user features' definition

Data Preparation

```
if "Endpoint" in JSON and "path_to_data" not in JSON:
    # input from endpoint
    knowledge_graph_embedding_endpoint(input_data, st)
elif "Endpoint" not in JSON and "path_to_data" in JSON:
    # input from dataset
    knowledge_graph_embedding_dataset(input_data, st)
else:
    print("There are something wrong 'Endpoint' and '
       path_to_data' in the JSON file can only have one
       of these two data at same time.")
```

Listing 5.1: Pseudo code for processing different data sources

To retrieve data from the SPARQL endpoint, the IntepretME pipeline provides a function that currently supports only SELECT clause and does not support other three clauses, In this thesis, this part of the function is extended, which is able to support CONSTRUCT clauses to extract a RDF subgraph. Here, the user does not need to have full knowledge of SPARQL for writing a query, but only needs to understand which are the graph patterns whose answers correspond to triples. Further, these RDF triples is included in the training of the KGE model based on the input data source. Graph patterns can include the OPTIONAL operator to ensure that all the key entities are not missing, if part of the entities are not satisfied multiple triple patterns, the key entities in this situation can still be recognized and processed by query.

CONSTRUCT Query

```
CONSTRUCT {
?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://dbpedia.org/ontology/Person> .
?person <http://dbpedia.org/ontology/parent> ?parent.
?person <http://dbpedia.org/ontology/gender> ?gender .
?person <http://dbpedia.org/ontology/hasSpouse> ?
   hasSpouse .
}
WHERE {
?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://dbpedia.org/ontology/Person> .

OPTIONAL {?person <http://dbpedia.org/ontology/parent> ?
   parent .}
OPTIONAL {?person <http://dbpedia.org/ontology/gender> ?
   gender .}
?person <http://dbpedia.org/ontology/numSpouses> ?
   numspouses.
BIND(IF(?numspouses>0 ,"Yes","No" )as ?hasSpouse ) }
```

Listing 5.2: Example of extracting a RDF subgraph from DBpedia. More rich and comprehensive subgraphs can be obtained by adding more triple patterns, in this example, we obtained the entities that are of type person and get the corresponding triples with the relationships: dbo:parent, dbo:successor, dbo:spouse, dbo:gender, etc. to perform the LP task.

5.2 KGE Models Hyperparameter Optimization

This section illustrate the functionality of the pipeline using PyKEEN `hpo()` for selecting the best optimize parameters to train a KGE model. The optimization of model hyperparameters is distinct from the processes of training and evaluating the model. This thesis integrates these steps into a unified pipeline, streamlining the user experience. Users now only need to set two parameter sets instead of manually initiating model training after conducting HPO (Hyperparameter Optimization) Listing 5.4. Following the retrieval of hyperparameters, specific parameters tailored for different KGE models come into play. After obtaining the hyperparameters, utilize the PyKEEN common interface to input these parameters and generate the corresponding Knowledge Graph Embedding (KGE) model, as depicted in Listing 5.3. Next, the training and evaluation processes are seamlessly executed within this integrated pipeline. Despite all models utilizing the same interface, certain models require specific hyperparameters. For instance, TransR introduces the notion of a relation space, necessitating the inclusion of the relation_dimension parameter.

Model build

```
result = pykeen_pipeline(
    training=training,
    testing=testing,
    validation=validation,
    model=model,
    model_kwargs=HP_data["pipeline"]["
       model_kwargs"],
    optimizer=HP_data["pipeline"]["optimizer"],
    optimizer_kwargs=HP_data["pipeline"]["
       optimizer_kwargs"],
    loss=HP_data["pipeline"]["loss"],
    loss_kwargs=HP_data["pipeline"]["loss_kwargs"
       ],
    training_kwargs=HP_data["pipeline"]["
       training_kwargs"],
)
```

Listing 5.3: Use pykeen_pipeline to build training and evaluate model.

5.3 KGE Model Training and Evaluation

The current version of InterpretME is capable of performing ensemble learning models (e.g., random forest), this thesis aims to extend the pipeline functionality with *kge_model* as a mode. Based on user preference, InterpretME is utilized in various prediction tasks. Henceforth, InterpretME resorts to PyKEEN [5] which is a general-purpose python library designed for training and evaluating KGE models[8]. It provides comprehensive support for building pipelines using a variety of KGE models, training strategies, and loss functions. PyKEEN's modular structure allows for different components to be implemented as independent submodules, thus, facilitating the creation of various combinations that meet specific requirements. Although the library contains more than 20 models, our study focuses only on the translation distance KGE family, which include TransE, TransH, TransR, and TransD. As mentioned in Listing 5.4, the best parameters of each KGE model are obtained with a certain number of trials (e.g., TransE trained with 100 trials). Best embeddings learning is traced and provided to the training module. Moreover, in the evaluation phase, KGE models generate trained embeddings, and the scoring process involves calculating scores for all possible entities E based on the given tail (h, r) or head (r, t) prediction. The entities are then ranked (h, r, E) (or (E, r, t)) using the plausibility score function derived from the respective KGE model. Additionally, the PyKEEN module computes aggregation metrics according to the definition of a realistic rank. InterpretME implements this definition and reports metrics, specifically MRR and Hits@k.

Model selection

```
    "model": ["TransE","TransR","TransH","TransD"],
    "HPO_trials": {"TransE": 100,"TransR": 20,"TransH": 30,"TransD": 20},
```

Listing 5.4: The user could select which model should be trained and evaluated during this run by setting the input JSON file.

5.4 Documenting KGE Models

The essence of this book necessitates the inclusion of specific data and metadata of the KGE model characteristics. This includes tasks such as assigning a unique identifier (ID) to each run, and documenting input data (i.e., detailing the extracted

[8] https://github.com/pykeen/pykeen

features and graph patterns involved. Moreover, enumerating the hyperparameters (i.e., embedding dimension, batch size, epochs, etc.) obtained from the HPO for model building along with their best values. Additionally tracing the evaluation metrics such as MRR and Hits@10. To organize and store these contextual metadata, several designated CSV files are utilized. These files are later transformed into a knowledge graph format by an efficient RML engine, RDFizer, employing predefined RML mapping rules. The mappings for the endpoint are illustrated in Listing 5.6.

Mapping rules

```
@prefix rr:<http://www.w3.org/ns/r2rml#>.
@prefix rml:<http://semweb.mmlab.be/ns/rml#>.
@prefix ql:<http://semweb.mmlab.be/ns/ql#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdfsyntaxns#>.
@prefix schema:<http://schema.org/>.
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>.
@prefix owl:<http://www.w3.org/2002/07/owl#>.
@prefix prov:<http://www.w3.org/ns/prov#>.
@prefix intr:<http://interpretme.org/vocab/>.
```

Listing 5.5: Commen prefix in Mapping rules

```
<InputEndpoint>
rml:logicalSource [
    rml:source "InterpretME/files/endpoint.csv";
    rml:referenceFormulation ql:CSV;
    ];
    rr:subjectMap [
        rr:template "http://interpretme.org/entity/{
           run_id}";
        rr:class intr:RunID
    ];
    rr:predicateObjectMap [
        rr:predicate intr:hasEndpoint;
        rr:objectMap [
        rml:reference "endpoint"
        ]
    ].
```

Listing 5.6: Mapping rules for endpoint

The resulting knowledge graph is named InterpretME KG. The InterpretME KG is a heterogeneous graph which represents each target entity with the KGE model characteristics shown in Figure 6.6. The pre-requisite vocab required to describe the KGE models is available in GitHub[9]. InterpretME as a framework offers a function, that enables users to access this generated KG locally through SPARQL queries. A better understanding of the model and the influence of parameters on the performance of model, requires a well-designed query to get specific interpretations. Additionally, the federated query processing module helps to query both the input KGs and the InterpretME KG for a target entity to have more rationale about the prediction task. For instance, a SPARQL query over the federated KGs that retrieves the relevant known facts about the head entity and the model parameters utilized in the prediction helps in interpreting a tail prediction task. The proposed implementation is model-agnostic and the final task is to extend InterpretME in the domain of KGE modeling. Also, incorporates the approach mentioned in [11] to extract necessary and sufficient interpretations for a particular prediction task.

5.5 Summary of the Chapter

This chapter details the implementation of a comprehensive pipeline within InterpretME, designed to trace KGE models. The pipeline incorporates six distinct stages, encompassing data preparation, hyperparameter optimization, model training and evaluation, KGE model data tracing, and querying over the InterpretME KG. The objective is to streamline the overall process, accommodate diverse data sources, and increase KGE models' interpretability with contextual insights.

[9] https://github.com/STdove/Neo_masterthesis/tree/main/InterpretME/KGE_model/mappings

Chapter 6

Experimental Evaluation

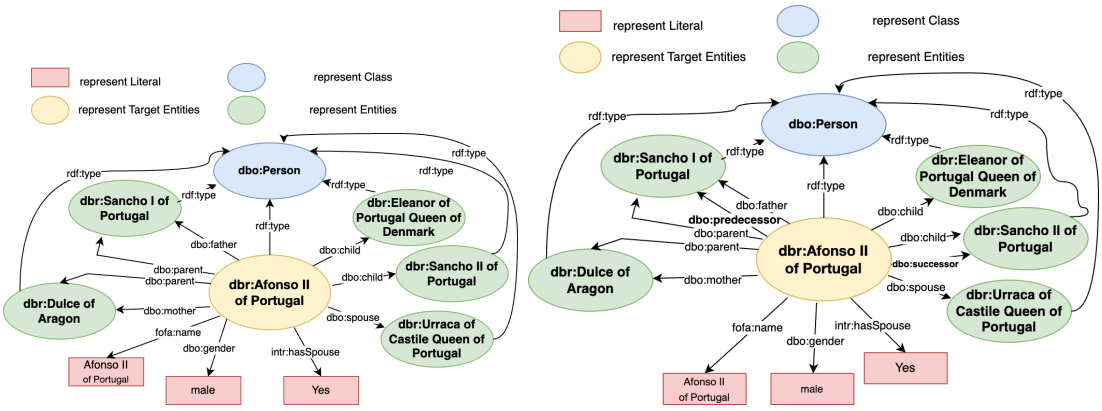
This chapter reports the results of the experimental evaluation of the proposed pipeline in Chapter 5. We empirically evaluate with four KGE models: TransE [7], TransD [22], TransR [28], and TransH [41]. The purpose of this evaluation aims to address the following research questions: **RQ1)** What is the impact of traced hyperparameters in the KGE models decision-making process? **RQ2)** To what extent does InterpretME enhance the interpretability of the KGE models?

We first extract the required RDF data from the benchmarks and then utilize InterpretME [12] for training and tracing the KGE models. Further, the fine-tuned KGE models are ultimately assessed in the Link Prediction (LP) task. For each evaluation of the translation model, we compare the evaluation metrics (e.g., Hits@10) for the benchmark KGs reported in Table 6.2. Moreover, an ablation study for the influence of hyperparameters is illustrated in Section 6.5.2. The benchmark KGs, scripts for training and optimizing hyperparameters, and a SPARQL query template for the statistical analysis over the InterpretME KG are publicly available in GitHub[10] repository.

Table 6.1: Benchmark KGs statistics

Knowledge Graph	#triples	#entities	#relations
French Royalty KG	4424	2212	10
Enriched French Royalty KG	12544	2212	12

[10] https://github.com/STdove/Neo_masterthesis



(a) French Royalty KG (b) Enriched French Royalty KG

Figure 6.1: **Exemplar Sub-Graph**. Figure 6.1a shows an instance `dbr:AfonsoII_of_Portugal` with relations, such as `dbo:parent, dbo:mother, dbo:sposue, dbo:child`. While, Figure 6.1b demonstrates the neighborhood sub-graph of `dbr:AfonsoII_of_Portugal` with enriched heuristics edges (e.g., `dbo:predecessor, dbo:successor`) generated by SPaRKLE approach.

6.1 Benchmarks

The following two benchmark datasets are considered: *French Royalty KG* [18] and *Enriched French Royalty KG* [31]. Table 6.1 provides statistics for these datasets and their respective entities and relations counts. Figure 6.1a and Figure 6.1b shows the exemplar instances from this two benchmark KGs. The *French Royalty KG* is fully curated from DBpedia [27], by retrieving information about the french royal families. For each person in the KG, we require the class `dbo:Person`, the name of children (if has one or more), the name of the parent (father and mother in this context), the gender of french person, and additional triple related counts like the number of spouse, successor, and predecessor. Furthermore, we added the ground truth for the relationship `dbo:hasSpouse` which states the marital status of a french royal member. On the Contrary, the *Enriched French Royalty KG* is obtained from SPaRKLE [31] method–hybrid AI–inductive learning that incorporates the fusion of sub-symbolic (i.e., KGE) and symbolic learning (i.e., Rule Mining) for the assessment of knowledge graph completion (KGC). Thus, the second dataset is materialized using the defined concept of PCA heuristics edges in [31]. Here, the proposed approach relies on SPaRKLE to provide the refined semantic description of an entity. Moreover, the trained embeddings for both benchmarks are obtained to perform the LP task, i.e.,

"Whether a french royal person has a spouse?". Lastly, the proposed approach is compatible with different data sources:

- one is handling the data from the SPARQL endpoint.
- another is accessing the file stored locally in CSV, N-triples, TSV format.

Henceforth, to show this implementation, the pipeline is evaluated with two data sources: the *French Royalty KG* will be provided in the SPARQL endpoint, and the dataset *Enriched French Royalty* can be accessed locally in TSV format.

6.2 Metrics

This section describes the metrics utilized for the evaluation of testbeds for LP. The efficiency is assessed through Mean Reciprocal Rank (MRR), Hits@10, and Execution Time.

- *MRR* and *Hits@10*: MRR is the arithmetic mean of reciprocal ranks, Hits@10 describes the percentage of true entities that appear in the top 10 entities of the sorted rank list. The values for MRR and Hits@10 is range in [0,1].
- *Execution Time*: This metric is measured as the elapsed time spent by the pipeline to complete the extraction of the training dataset, HPO configuration, model training, evaluation, and generating an InterpretME KG; it is measured in seconds reported by the time.time() command which is supported by Python.

6.3 Implementation Details

The following configuration is set up to answer our research questions. The proposed approach–Trace KGE over InterpretME–an interpretable framework is implemented based on a hybrid design pattern mentioned in 13. Here, the traditional ML models are replaced with the translation distance KGE models (e.g., TransE) to perform the prediction task. The experiments are executed in a Windows 10 OS equipped with a CPU Intel (R) Core (TM) i9-9900KF@3.60 GHz, 32 GB memory with Python 3.9 version. For optimizing the hyperparameters of the KGE models (e.g., embedding dimensions), the proposed pipeline relies on PyKEEN `hpo()` function. Executing HPO configuration required a GPU NVIDIA Tesla T4, with CUDA version 12.0 and PyTorch version 2.0. HPO configuration provides a set of predefined settings that enable hyperparameters to be computed and generate optimal solutions under

the same conditions. Moreover, the time out is set to 3 hours since the HPO part, which is supported by PyKEEN to avoid memory issues. During the dataset splitting phase, the random state was set to 1234 to ensure that the training, testing, and validation datasets were consistent across each experimental testbed.

6.4 Baseline

The experiments used source codes from the PyKEEN [5] pipeline to run several KGE models, which include TransE, TransH, TransD, and TransR. We use the same hyperparameter optimization tool provided by PyKEEN to tune the hyperparameter settings of the KGE model based on the benchmark KGs. The translation distance KGE models, including TransE, TransH, TransD, and TransR, translate the embedding of the target head entity and the given relation to make them together closer to the target tail entity. The transformed embedding vector is then calculated using the scoring function provided by the KGE models. Here, the tail entities is ranked based on their difference from the predicted entity. This ranking helps identify the true tail entity for a given head entity and relation. Executing several testbeds, the evaluation reveals that the experimental results are slightly different. However, during the model building phase, lots of hyperparameters is defined by pipeline for each testbed, there are still unknown parts of the computation in PyKEEN that will generate new random numbers to perturb the results, and cannot be traced by interacting with the PyKEEN module.

6.5 Results

The following multiple sets of data comparison, are able to answer **RQ1**: tracing a wide variety of data in the whole pipeline, convenient for users to quickly access the results of the experiment, with the test setup, users can compare the data in time to data visualization of the test is carried out under what conditions, to facilitate the user to complete the following operations: 1) Compare the performance of the different models under the same conditions; 2) To a single parameter adjustment of a single parameter to analyze the effect of the parameter on the model performance. 3) The HPO function provided by PyKEEN now has a certain degree of randomness, which makes it possible to analyze the effect of this randomness on the model performance with the presentation of a large amount of experimental data. 4) The data from multiple experiments will be retained in InterpretME KG, which enables us to quickly review and reproduce the experiments according to the stored settings.

6.5.1 Baseline Experiment

Table 6.2: **Evaluation Results**. KGE models evaluation is represented in terms of Hits@10 and MRR over benchmark KGs. Bold represents better values.

Knowledge Graph	Metrics	TransE	TransH	TransR	TransD
French Royalty KG	**Hits@10**	0.436	0.755	0.390	0.530
	MRR	0.151	**0.528**	0.301	0.182
Enriched French Royalty KG	**Hits@10**	0.488	**0.761**	0.452	0.578
	MRR	0.176	0.507	0.381	0.201

Table 6.2 show the performance of the *French Royalty KG* and the *Enriched French Royalty KG* respectively, in the absence of HPO considering it as a baseline experiments for the four models of translation family. Figure 6.2 based on aforementioned Table 6.2 reveals that the TransH model performs the best on both KGs, where Hits@10 could reach more than 70% and MRR has more than 50%. TransE has a poor performance compared to other models, which is in the expectation because there are one-to-many and many-to-many relationships in the KG, e.g., a person will have more than one child, and a person will usually have two parents. Thus, TransE because of its modeling properties, is not very good at handling $1-N$, $N-1$ and $N-M$ relations resulting in worst performance than the other translation models. In *French Royalty KG*, TransE is reported with Hits@10 0.436, while TransD is 0.530 and TransH is 0.755. Simultaneously, we are able to observe that the *Enriched French Royalty KG* is able to slightly improve the performance of the model on Hits@10 as well as on MRR, which is as expected because from Table 6.1 we know that the database itself has more triples, while the number of entities remains unchanged, indicating that there is more contextual information to describe the entities. Resulting Hits@10 for TransH be 0.761, while TransE is 0.488. Moreover, TransH is slightly up in Hits@10 and the MRR performance is down 2%, suggesting that the correctly predicted triple is more often found in the first 10 predictions, but this correct triple is not ranked so highly in top-10. Intriguingly, without optimize parameters the TransR model is performing worst in both benchmark KGs.

6.5.2 Influence of HPO

The HPO function provided by PyKEEN, based on the target model, and the data source finds the best optimal hyperparameters. For the statistical analysis and visualization of experimental data, Figure 6.3 and Figure 6.4 are provided based on the

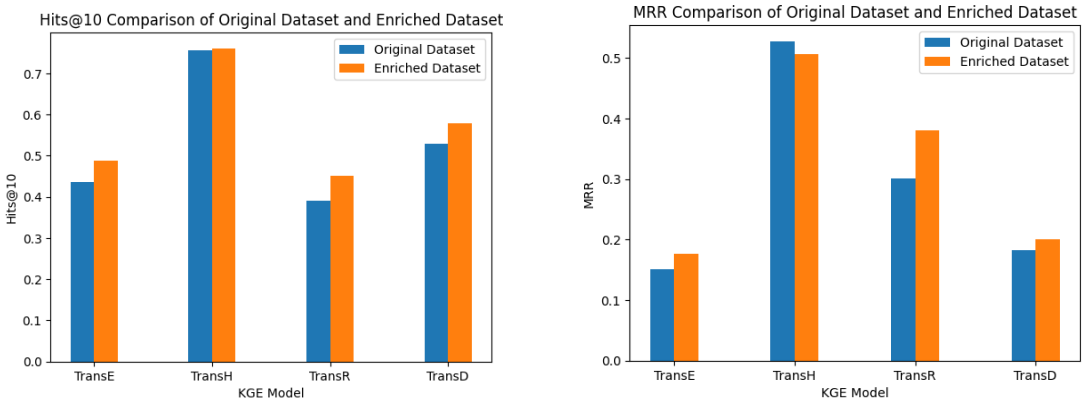


Figure 6.2: Hit@10 and MRR Comparison Plot based on Table 6.2

comparison of the KGE models evaluation with, without and custom hyperparameters settings. Here, we can analyze the impact of the HPO module on experimental results, it can be concluded that the HPO function provided by PyKEEN is currently beneficial in improving the performance of the TransE, TransH and TransD models. However, this assistance is exceedingly limited and even falls short of being comparable to manually customizing parameter settings, a possible reason for this is that the way HPO now looks for optimal hyperparameters can only find relatively good hyperparameters, but it is difficult to find an optimal solution. Simultaneously, we observed that adjusting hyperparameters has a significant positive impact on the performance of the TransE model in Hits@10, but it also has a negative effect on the performance of the TransR model, which is shown in Table 6.4. In the meanwhile, under the MRR metric, observed from Figure 6.4, the radar chart shows that it has a limited effect on the TransH and TransD. And customizing hyperparameters that negatively affect the most models while making TransE perform better, probably because customized hyperparameters hardly take into account different parameters of other translation models, and may only improve TransE model performance with this setting. Through the comparison of model training results using without, with and custom hyperparameters presented in Table 6.3, we were able to find that custom hyperparameters can have a positive impact on the model, and the hyperparameters provided by HPO can also optimize model performance, but difficult to reach optimal solution. In *French Royalty KG*, TransH is peforming better with Hits@10 0.709 in custom hyperparameters settings, whereas with HPO fucntion from PyKEEN is reported 0.609. Thus, the experimental findings of HPO influence addresses **RQ1)**

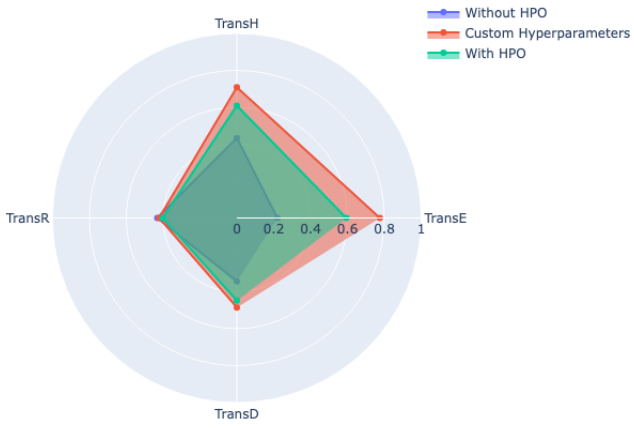


Figure 6.3: Figure based on Table 6.3, a better visualization of the impact, that HPO has made on the models. In metrics Hits@10. As shown in the figure, HPO has no impact on TransR, while the use of custom Hyperparameters for these three models has a better performance than the use of the hyperparameters provided by HPO.

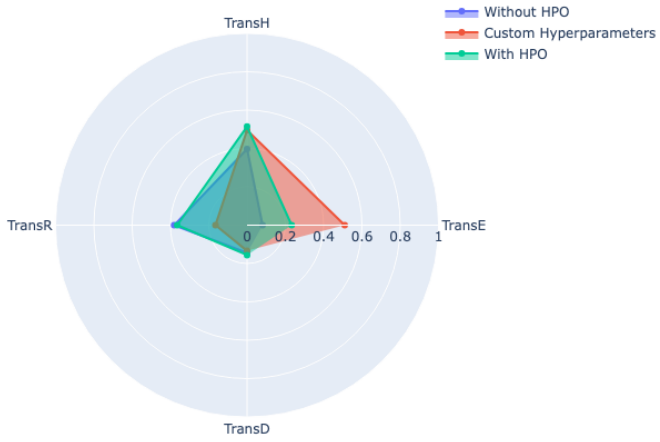


Figure 6.4: Figure based on Table 6.3, a better visualization of the impact, that HPO has made on the models. In metrics MRR. As one can see from the figure, it has a limited impact on the MRR performance of TransH and TransD. And the customized hyperparameters can still bring better performance for TransE.

Table 6.3: **Evaluation results of setting hyperparameters on TransFamily**. The first group is without setting hyperparameters, the second group is when the model was built with embedding dimension set to 200 and epochs set to 100, and the third group is when the model uses the PyKEEN HPO function to calculate the hyperparameters. And the bold represents the best-performing data for each type of model in this run.

HPO		TransE	TransH	TransR	TransD
Without HPO	**Hits@10**	0.220	0.434	**0.432**	0.343
	MRR	0.081	0.397	**0.381**	0.144
Custom HPO	**Hits@10**	**0.776**	**0.709**	0.422	**0.485**
	MRR	**0.510**	0.495	0.165	0.134
With HPO	**Hits@10**	0.594	0.609	0.402	0.448
	MRR	0.233	**0.514**	0.365	**0.156**

which have a great impact in the evaluation of the model outcomes.

6.5.3 Influence of hyperparameters in TransR

Table 6.4: Evaluation of the TransR model in different hyperparameters settings reported with Hits@10 and MRR.

Run_ID	Model	Hits@10	MRR	Dimension	Batch Size	Epochs
1706103208087	TransR	0.429	0.381	168	3712	15
1706103692128	TransR	0.432	0.406	**200**	3712	15
1706104410104	TransR	0.420	0.388	**136**	3712	15
1706104450287	TransR	0.324	0.236	168	3712	**100**
...	TransR	...	...	...	...	...

Using a customized SPARQL query, we are able to obtain the formulated results from the InterpretME KG demonstrated in Table 6.4. This query retrieves following relevant information about the trained TransR model in multiple runs, which contains a specific Run_ID, model, Hits@10, MRR, embedding dimensions, batch size and epochs, and allow user to explore the effect from various hyperparameters. The data is sorted based on Run_ID, so often the positions of the data that the user wishes to analyse are not aligned. Thus, there is a need to access specific data from the IntepertME KG through the methods proposed in Subsection 6.5.5 to help

the user to perform better analyses. An ablation study of comparing different hyperparameters such as embedding dimensions in Table 6.5, batch size in Table 6.6, and epochs in Table 6.7 individually affecting the evaluation of the TransR model. We separately study the impact of each hyperparameter on the models performance. Based on the above three tables, three line charts in Figure 6.5 are used to provide a more intuitive comparison. In Table 6.5, the experiments on the hyperparameter embedding dimension states that the model performed optimally with dimension at 180, with Hits@10 and MRR higher than the other settings. Meanwhile independent trials of epoch with a setting of 115 resulted in a model that performed much better than the other models. In the meanwhile, batch size has a tendency to grow with fluctuations, and although the performance of the model improves as the batch size increases, there is no guarantee that this growth trend will be maintained. From the three line charts, we can observe that as the hyperparameters change, the two metrcis maintain the similar trend, which can explicitly confirm that the hyperparameters have an impact on the performance of the model, not the performance of a single metric. At the same time, this trend shows that different parameters produce different impacts impact on the model. And we find that parameters that are too large or too small can affect the results, it is not possible to find the optimal hyperparameters directly through a linear associationand, this increases the difficulty in optimizing the performance of the model. Based on the above experimental results and analyses, we additionally set up one experiment by setting the hyperparameters Embedding dimension to 176, Epochs to 115, and batch size to 4736. We could build and train a model TransR and get performance in *Hits@10* at **0.441** and *MRR* at **0.407**, a significant improvement compared to the testbeds of French Royalty KG input into TransR model in Table 6.2. However, the differences compared to adjusting them individually are very slight, but, due to the range of experimental parameters and step size settings, we cannot guarantee that the experiments conducted are not local optimal hyperparameters but global optimal hyperparameters. This my caused by the hyperparameters still not optimal. Meanwhile, according to the above analysis and compared to the testbeds in Table 6.3 we can confirm that HPO has some limitations in finding the optimal hyperparameters.

6.5.4 Operations required to initialize the pipeline

The use of the integrated pipeline can greatly simplify the various stages of the operation and is also required with limited input from users. Listing 6.1 depicts the required inputs are simplified compared to setting up separate tests, "Endpoint" can also be set to "path_to_data" to allow Pipeline to access the local RDF graph,

Table 6.5: This table is used to analyze the effect of embedding dimensions, step for adjustment is 32, the hyperparameters not shown here will remain the same.

Run_ID	Model	Hits@10	MRR	Dimensions
1706275630133	TransR	0.421	0.385	208
1706275630133	TransR	**0.439**	**0.400**	176
1706274900713	TransR	0.417	0.369	144
1706275110524	TransR	0.368	0.301	112
1706275187105	TransR	0.412	0.381	80
1706278942768	TransR	0.412	0.356	48

Table 6.6: This table is used to analyze the effect of batch size, step for adjustment is 1024, the hyperparameters not shown here will remain the same.

Run_ID	Model	Hits@10	MRR	Batch size
1706103208087	TransR	**0.430**	**0.403**	4736
1706276551129	TransR	0.390	0.323	3712
1706274900713	TransR	0.417	0.369	2688
1706276428746	TransR	0.390	0.323	1664
1706276487034	TransR	0.343	0.249	640

Table 6.7: This table is used to analyze the effect of epochs, step for adjustment is 20, the hyperparameters not shown here will remain the same.

Run_ID	Model	Hits@10	MRR	Epochs
1706277109460	TransR	0.356	0.303	155
1706277036531	TransR	0.423	0.365	135
1706277036531	TransR	**0.430**	**0.403**	115
1706276806125	TransR	0.392	0.347	95
1706276896928	TransR	0.406	0.359	75

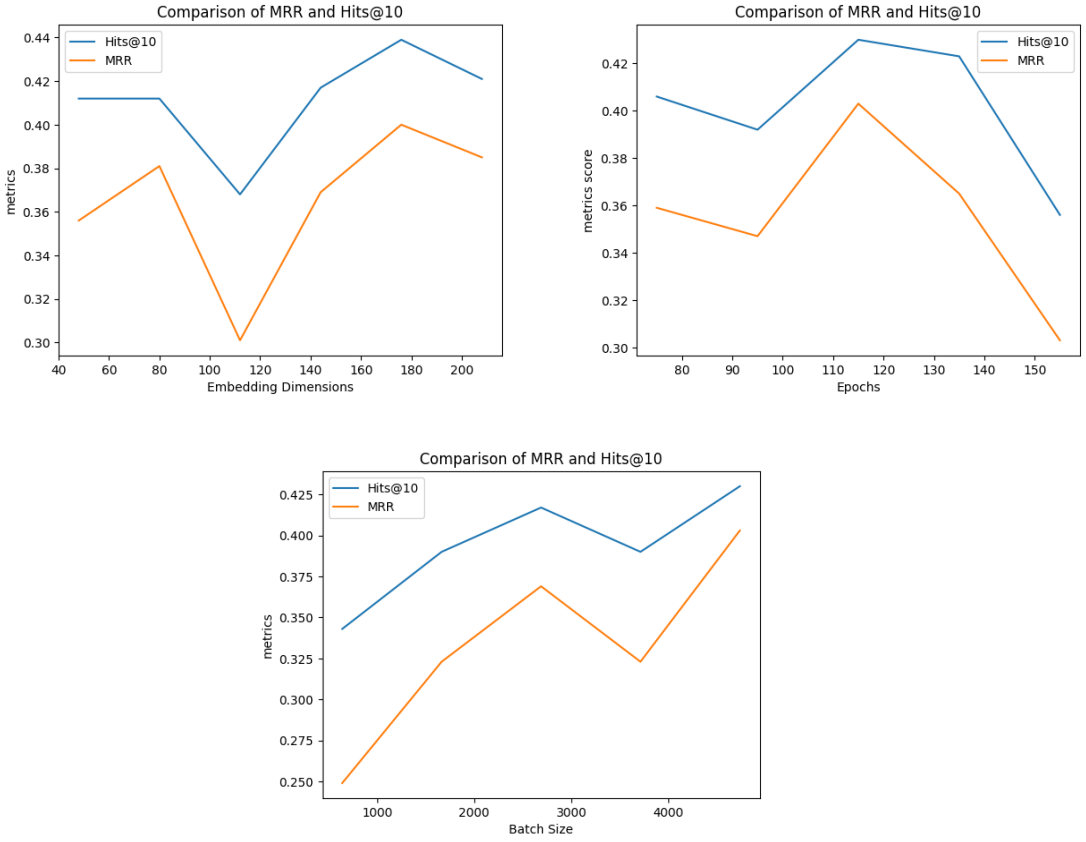


Figure 6.5: The line chart clearly shows the impact of hyperparameters on model performance. The similar trend of MRR and Hits@10 can prove that hyperparameters affect the performance of the evaluation metrics.

note that there can only be one "Endpoint" or one "path_to_data" at the same time. "model" is used to set the type of model that users want to test in this round of experiments, and HPO_trials is used to set the ability of the PyKEEN HPO module to optimize the model. The three variables are used to define the input data of the model, and the prediction content, Independent_variable is the target Entities, "Independent_Optional_variable" can be regarded as the enrichment of the target Entities, and "Dependent_variable" is the setting for the prediction task. "Independent_Optional_variable" is simplified compared to the actual experiment. For instance, for the testing of multiple models, routinely need to repeat a number of data acquisition and preparation, parameter setting, model building, and other operations. Whereas, experimental results and experimental evaluations are displayed

directly in the process each time, or stored in a folder separately, and need to be manually integrated again by the experimenter for further comparisons and analyses. These pipeline overcomes such issues efficiently and automates these processes so that the user only needs to use SPARQL queries at the final stage of the pipeline to get the interpretability of the models decisions from the InterpretME KG.

Input JSON configuration

```
{
"Endpoint": "https://labs.tib.eu/sdm/InterpretME-og/
   sparql",
"model": ["TransE","TransH","TransR","TransD"],
"HPO_trials": {"TransE": 20,"TransR": 20,"TransH": 20,"
   TransD": 20},
"Independent_variable": {
      "person": "?person <http://www.w3.org/1999/02/22-
         rdf-syntax-ns#type> <http://dbpedia.org/ontology
         /Person> ."
    },
"Independent_Optional_variable": {
      "father": "?person <http://dbpedia.org/ontology/
         father> ?father.",
      "parent": "?person <http://dbpedia.org/ontology/
         parent> ?parent.",
      "name": "?person <http://xmlns.com/foaf/0.1/name> ?
         name.",

  },
"Dependent_variable": {
         "hasSpouse": " ?person <http://dbpedia.org/
            ontology/numSpouses> ?numspouses. BIND(IF(?
            numspouses>0 ,\"Yes\",\"No\" )as ?hasSpouse )
            "
    }
}
```

Listing 6.1: Example of input setting required from User.

6.5.5 SPARQL queries over the IntepretME KG

After the execution of the pipeline, an InterpretME KG will be generated. Figure 6.6 demonstrates an exemplar of a Run_ID instance (i.e., intr:Run_1705961838474) which includes the traced metadata such as features definition, data sources, KGE model characteristics, which with all hyperparameters suggested by HPO function. The Target Entity could have multiple Models, and these model entities will have the same number of relations and corresponding subjects as TransE. To query all instances same as the instance shown in the Figure 6.6, we need to build a query to access the IntepretME KG, one example could be found in Listing 6.2, SELECT section defined all the columns we want to extract, WHERE section defines the graph patterns to make sure we get correct feedback, (?runID rdf:type intr:RunID) going to make sure the instances in ?runID is class Intr:RunID, and also the same purpose of (?Model rdf:type intr:Model), with this two restrict, make sure we get correct target entity in instances, and other patterns used to get the corresponding nodes and literals. The complete flow of this process will be like in the Figure 6.7.

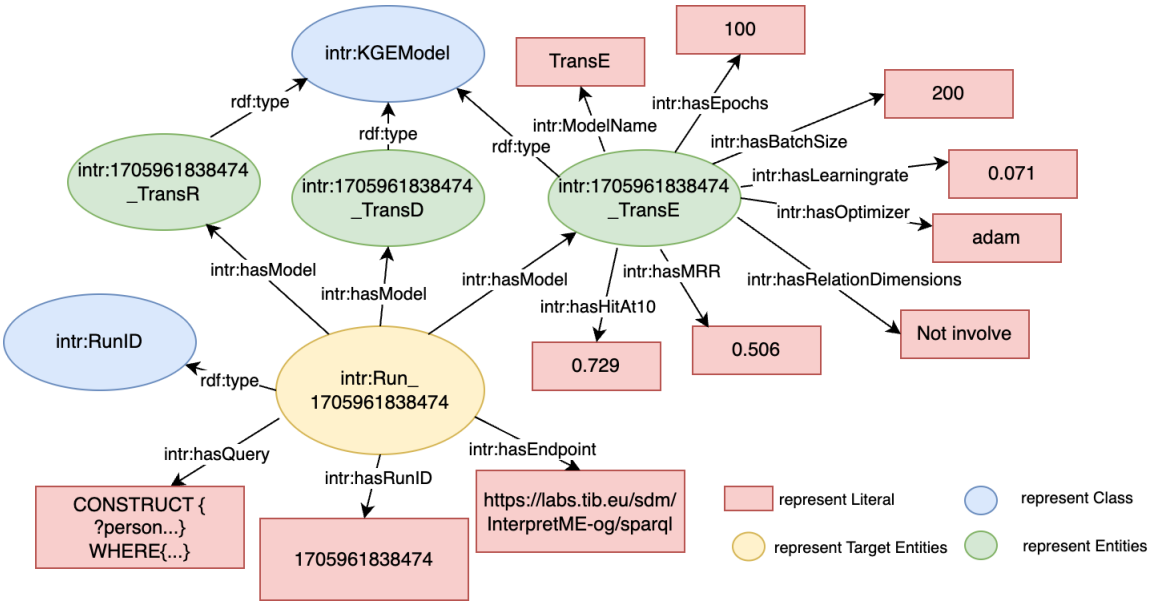
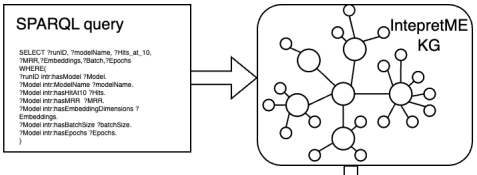


Figure 6.6: An entity instance of InterpretME KG, where the triple owned by TransR and TransD is omitted and the target entity could have one to four intr:hasModel relation, it depends on the setting of the corresponding run.

Example of Query over the InterpretME KG

```
SELECT ?runID, ?modelName, ?Hits_at_10, ?MRR,?Embeddings
   ,?Batch,?Epochs
WHERE{
?runID rdf:type intr:RunID.
?runID intr:hasModel ?Model.
?Model rdf:type intr:Model
?Model intr:ModelName ?modelName.
?Model intr:hasHitAt10 ?Hits.
?Model intr:hasMRR  ?MRR.
?Model intr:hasEmbeddingDimensions ?Embeddings.
?Model intr:hasBatchSize ?batchSize.
?Model intr:hasEpochs ?Epochs.
}
```

Listing 6.2: This is a sample statement to access all the runs and the model used by that run, as well as the hyperparameters of the model with the model's performance from the InterpretME KG. By accessing the IntepretME KG, we are able to track down previously run models, quickly access the model's performance, and through the hyperparameters stored in the KG, quickly reproduce the results of the experiment.



Run_ID	Model	Hits@10	MRR	Dimension	Batch Size	Epochs
1706103208087	TransR	0.429	0.381	168	3712	15
1706103692128	TransR	0.432	0.406	**200**	3712	15
1706104410104	TransR	0.420	0.388	**136**	3712	15
1706104450287	TransR	0.324	0.236	168	3712	**100**
...	TransR	...	...	...	...	...

Figure 6.7: With the SPARQL query in Listing 6.2, we are able to get a list of corresponding results Table 6.4 from the InterpretME KG.

6.6 Summary of the Chapter

This chapter presents three research questions, and introduces benchmarks, and metrics for the experiments. With the RDF example, we were able to directly observe what kind of features their entities have in different databases. We experimentally evaluated and compared the performance of the KGE model provided by PyKEEN and tested the usefulness of PyKEEN's HPO module. Based on the results of these two sets of experiments, we conducted extended experiments to additionally test the utility of TransR's hyperparameters. **RQ2** in Chapter 6 was tested and answered through multiple sets of experiments. In the evaluation phase of the experiments, we accessed the IntepretME KG generated by the pipeline and used the data therein to help us analyze the performance of the model in each run and the possible reasons for the good and bad performance, which showed that the pipeline was able to address the issues raised by **RQ1** in Chapter 6 well.

Chapter 7

Conclusions and Future Work

This chapter presents the evaluation of the proposed pipeline with the contributions. Also, the current limitations and future research for the pipeline are discussed.

7.1 Conclusions

This thesis proposes a model-agnostic framework, InterpretME, that tackles the problem of tracing and interpretability of KGEs over KGs especially with the translation distance models. The proposed approach enables or improves the functionality of the pipeline to adapt the CONSTRUCT clause build query and supports multiple ways to load RDF KG. In addition, a new mode is created to execute the KGE model, which provides the flexibility to easily take other updated ML models into consideration. Firstly, InterpretME extracts the RDF data and then transfers the optimal parameters to learn the KGE models in an efficient way. The approach reduces a large number of repetitive experiments for the selection of hyperparameters. The current version of InterpretME utilizes the optimization techniques from PyKEEN HPO module. Further, the proposed approach has shown that the translation models have an influence of certain parameters in the training phase. Additionally, the pipeline traces all the relevant data and generates an InterpretME KG, to enable users to access various data and metadata for a better understanding of the model-decision making process. Thus, the approach sheds a perception of having the semantic representation of the models' interpretability in human and machine readable format. In related work, different approaches have been explored for extracting the interpretations such as LIME and SHAP for the KGE models. While, such data-driven frameworks is less compatible to KGs where the data is represented by a RDF triples. Henceforth, the generated InterpretME KG aids the user to understand and analyze

the model performance. A large number of experimental results covered in the aforementioned section are generated through InterpretME pipeline. With the help of abundant available traced metadata that are relevant to the experimental context, the researchers are able to better understand the justification for the KGE model outcomes. This pipeline not only provides interpretations, but also provides structured visualization functionality that enlightens the empirical results with intuitive explanations.

7.2 Limitations

- According to the evaluation part and results from multiple experiences, we can see that the hyperparameters used by the model are not guaranteed to be optimal. The HPO module of PyKEEN is well optimized for TransE but is not reliable for other models, we need to find a better way to help users find optimal hyperparameters and achieve a better performance of the model. Possible solution is to find other stat-of-the-art AutoML tools to help users obtain better model performance, or by providing various charts to provide optimization information, which is then selected by the user to ensure that the hyperparameters used are as optimal as possible.

- Currently, the generated InterpretME KG for tracing the KGE models have excluded SHACL constraints which validate and control the quality of the RDF data. Shortly, inclusion of SHACL constraints will add one more layer to the interpretability for the prediciton task.

7.3 Future Work

- Knowledge Graph Embedding (KGE) models can be expanded to include additional models, such as RotatE, MuRE, and others [6, 36]. Validating the pipeline enhances interpretability, and incorporating these state-of-the-art models into the pipeline offers users a broader range of choices to address various usage needs and scenarios.

- We are also eager to explore improved and more accurate HPO solutions to enhance the performance of the model. Resulting in less concern for users in the hyperparameters selection and model-building phases.

- This current work focuses on improving model interpretability by understanding the performance of the model, but there is still a lack of explanations for the link prediction task, e.g. for (Dirk_V_Count_of_Holland, hasSpouse, ?), which particular fact leads the model to predict missing link as No. Additionally, one area for research will be to explore removing such neighborhood facts impact to change in the link prediction result.
- The current version of InterpretME offers SHACL validaiton for ML models. Adding SHACL validation to ensure the high quality input data for training the KGE models could be one research direction.
- Moreover, one area of improvement for the performance of KGE models could be detecting bias and the role of mediator nodes [3] in KGs.

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG].

[3] Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. "Realistic Re-evaluation of Knowledge Graph Completion Methods: An Experimental Study". In: *CoRR* abs/2003.08001 (2020). arXiv: 2003.08001. URL: https://arxiv.org/abs/2003.08001.

[4] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. "Bringing Light Into the Dark: A Large-Scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12 (2022), pp. 8825–8845. DOI: 10.1109/TPAMI.2021.3124805.

[5] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. "PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings". In: *Journal of Machine Learning Research* 22.82 (2021), pp. 1–6. URL: http://jmlr.org/papers/v22/20-825.html.

[6] Ivana Balažević, Carl Allen, and Timothy Hospedales. *Multi-relational Poincaré Graph Embeddings*. 2019. arXiv: 1905.09791 [cs.LG].

[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. "Translating Embeddings for Modeling Multi-Relational Data". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 2787–2795.

[8] Jeremy Carroll and Graham Klyne. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. W3C, Feb. 2004.

[9] Arthur D Chapman. *Principles and methods of data cleaning*. GBIF, 2005.

[10] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. "Data cleaning: Overview and emerging challenges". In: *Proceedings of the 2016 international conference on management of data*. 2016, pp. 2201–2206.

[11] Yashrajsinh Chudasama. "Exploiting Semantics for Explaining Link Prediction Over Knowledge Graphs". In: *The Semantic Web: ESWC 2023 Satellite Events*. Ed. by Catia Pesquita, Hala Skaf-Molli, Vasilis Efthymiou, Sabrina Kirrane, Axel Ngonga, Diego Collarana, Renato Cerqueira, Mehwish Alam, Cassia Trojahn, and Sven Hertling. Springer Nature Switzerland, 2023.

[12] Yashrajsinh Chudasama, Disha Purohit, Philipp D Rohde, Julian Gercke, and Maria-Esther Vidal. "InterpretME: A Tool for Interpretations of Machine Learning Models Over Knowledge Graphs". In: ().

[13] Yashrajsinh Chudasama, Disha Purohit, Philipp D. Rohde, and Maria-Esther Vidal. "Enhancing Interpretability of Machine Learning Models over Knowledge Graphs". In: *Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems co-located with 19th International Conference on Semantic Systems (SEMANTiCS 2023), Leipzing, Germany, September 20 to 22, 2023*. Ed. by Neha Keshan, Sebastian Neumaier, Anna Lisa Gentile, and Sahar Vahdati. Vol. 3526. CEUR Workshop Proceedings. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3526/paper-05.pdf.

[14] Richard Cyganiak, Seema Sundara, and Souripriya Das. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. https://www.w3.org/TR/2012/REC-r2rml-20120927/. W3C, Sept. 2012.

[15] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. "Dermatologist-level classification of skin cancer with deep neural networks". In: *nature* 542.7639 (2017), pp. 115–118.

[16] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. "Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning". In: *arXiv:2007.04074 [cs.LG]* (2020).

[17] Ronald Gould. *Graph theory*. Courier Corporation, 2012.

[18] Nicholas Halliwell, Fabien Gandon, and Freddy Lecue. "User scored evaluation of non-unique explanations for relational graph convolutional network link prediction on knowledge graphs". In: *Proceedings of the 11th on Knowledge Capture Conference*. 2021, pp. 57–64.

[19] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. "Knowledge Graphs". In: *ACM Computing Surveys* 54.4 (July 2021), pp. 1–37. ISSN: 1557-7341. DOI: 10.1145/3447772. URL: http://dx.doi.org/10.1145/3447772.

[20] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. "Knowledge graph embedding based question answering". In: *Proceedings of the twelfth ACM international conference on web search and data mining*. 2019, pp. 105–113.

[21] Enrique Iglesias, Samaneh Jozashoori, David Chaves-Fraga, Diego Collarana, and Maria-Esther Vidal. "SDM-RDFizer: An RML interpreter for the efficient creation of RDF knowledge graphs". In: *Proceedings of the 29th ACM international conference on Information & Knowledge Management*. 2020, pp. 3039–3046.

[22] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. "Knowledge graph embedding via dynamic mapping matrix". In: *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 2015, pp. 687–696. DOI: 10.3115/v1/P15-1067. URL: https://aclanthology.org/P15-1067.pdf.

[23] Michael I Jordan and Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[24] Shristi Shakya Khanal, PWC Prasad, Abeer Alsadoon, and Angelika Maag. "A systematic review: machine learning based recommendation systems for e-learning". In: *Education and Information Technologies* 25 (2020), pp. 2635–2664.

[25] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA". In: *Journal of Machine Learning Research* 18.25 (2017), pp. 1–5.

[26] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. "Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia". In: *Semantic web* 6.2 (2015), pp. 167–195.

[27] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. "DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia." In: *Semantic Web* (2015). URL: http://dblp.uni-trier.de/db/journals/semweb/semweb6.html#LehmannIJJKMHMK15.

[28] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. "Learning Entity and Relation Embeddings for Knowledge Graph Completion". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 29.1 (Feb. 2015). DOI: 10.1609/aaai.v29i1.9491. URL: https://ojs.aaai.org/index.php/AAAI/article/view/9491.

[29] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[30] Alireza Osareh and Bita Shadgar. "Machine learning techniques to diagnose breast cancer". In: *2010 5th international symposium on health informatics and bioinformatics*. IEEE. 2010, pp. 114–120.

[31] Disha Purohit, Yashrajsinh Chudasama, Ariam Rivas, and Maria-Esther Vidal. "SPaRKLE: Symbolic caPtuRing of knowledge for Knowledge graph enrichment with LEarning". In: *Proceedings of the 12th Knowledge Capture Conference 2023*. K-CAP '23. Pensacola, FL, USA: Association for Computing Machinery, 2023, pp. 44–52. DOI: 10.1145/3587259.3627547. URL: https://doi.org/10.1145/3587259.3627547.

[32] Faiza Qamar, Seemab Latif, and Asad Shah. "Techniques, datasets, evaluation metrics and future directions of a question answering system". In: *Knowledge and Information Systems* (2023), pp. 1–34.

[33] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

[34] Andrea Rossi, Donatella Firmani, Paolo Merialdo, and Tommaso Teofili. "Explaining Link Prediction Systems Based on Knowledge Graph Embeddings". In: *SIGMOD*. 2022.

[35] Andy Seaborne and Eric Prud'hommeaux. *SPARQL Query Language for RDF*. W3C Recommendation. https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. W3C, Jan. 2008.

[36] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. "Rotate: Knowledge graph embedding by relational rotation in complex space". In: *arXiv preprint arXiv:1902.10197* (2019).

[37] Yi-Lin Tuan, Sajjad Beygi, Maryam Fazel-Zarandi, Qiaozi Gao, Alessandra Cervone, and William Yang Wang. "Towards large-scale interpretable knowledge graph reasoning for dialogue systems". In: *arXiv preprint arXiv:2203.10610* (2022).

[38] Denny Vrandečić and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase". In: *Communications of the ACM* 57.10 (2014), pp. 78–85.

[39] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 29.12 (2017), pp. 2724–2743. DOI: 10.1109/TKDE.2017.2754499.

[40] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. "Kgat: Knowledge graph attention network for recommendation". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 950–958.

[41] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. "Knowledge Graph Embedding by Translating on Hyperplanes". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 28.1 (June 2014). DOI: 10.1609/aaai.v28i1.8870. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8870.

[42] Jianfeng Zhao, Bodong Liang, and Qiuxia Chen. "The key technology toward the self-driving car". In: *International Journal of Intelligent Unmanned Systems* 6.1 (2018), pp. 2–20.