SAPIENZA
UNIVERSITÀ DI ROMA

Characterizing Knowledge Graphs in the Embedding Space

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Francesco Ottaviani
ID number 1759720

Thesis Advisor

Prof. Aris Anagnostopoulos

Co-Advisors

Ing. Ilaria Bordino
Ing. Francesco Gullo
Ing. Davide Mottin

Academic Year 2020/2021

Characterizing Knowledge Graphs in the Embedding Space
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: ottaviani.1759720@studenti.uniroma1.it

Contents

Introduction

The importance to organize data in ordered structures became a fundamental topic in the last years. From social networks to the World Wide Web, networks provide a ubiquitous way to organize a diverse set of real-world information. There are complex structures to represent relations between entities, and they have become an increasingly popular research direction towards cognition and human-level intelligence, so incorporating human knowledge is one of the research directions of artificial intelligence. Therefore, given a network's structure, it is often desirable to predict missing information (frequently called *attributes* or *labels*) associated with each node in the graph. This missing information can represent a variety of aspects of the data, for example on a social network they might represent the communities a person belongs to, or the categories of a document's content on the web.
A way to organize real-world information is through **knowledge graphs** (KGs), in which *nodes* represent entities (such as people and places), *labels* are types of relations that can connect them, and *edges* are specific facts connecting two entities with a relation. In this way it is possible to represent complex relations between real-world entities such as news, people, organizations, biological entities, and financial transactions.
Because the relationships of data are critical to understand the complexity about them, knowledge graphs are nowadays becoming the prominent data model to realize the intelligence of AI in many different application domains that have to deal with the challenge of transforming massive, heterogeneous and time-evolving data sources into machine-processable knowledge.
To try to get closer to the applications used with knowledge graphs, it is essential to introduce the concept of **graph embedding**. In general, embeddings are vectors of numerical values that can be used to represent any kind of elements (e.g., depending on the domain: words, people, products, etc.). Embeddings are learned automatically, based on how the corresponding elements occur and interact with each other in datasets representative of the real world. For instance, word embeddings have become a standard way to represent words in a vocabulary, and they are usually learned using textual corpora as input data. When it comes to KGs,

embeddings are typically used to represent entities and relations using the graph structure; the resulting vectors, named KG embeddings, embody the semantics of the original graph, and can be used to identify new links inside it, thus tackling the **Link Prediction** (LP) task. This job serves to exploiting the existing facts in a KG to infer missing ones, evaluating triples (which represent facts) through the scoring function of a KG model, representing the scored triples in a ranking table (like most of the projects done so far) or in other ways.
The project is carried out completely under the supervision of the Thesis Advisor and Co-Advisors. The work of this thesis project aims to investigate the areas of the KG where the LP task might work best. Going into more details, embedding models exist to evaluate positive or negative triples according to preliminary assumptions; at this point, it is necessary to understand if there are subareas that are more *reliable* than others, in which there may be a high probability that many triples are reconstructed. Therefore it has been suggested a formula for the **reliability** of KG models, which work on the entire KG, to try to keep the information about the density (and sparsity) of the graph in certain areas.
The approach used before developing the project was to make observations in order to formulate hypotheses, such as the reliability formula, and then carry out experiments following different techniques or methods for the choice of different graphs to be analyzed, sampling them in various ways, the use of models and the building of a ML classifier for the LP task.
After giving a general background on the elements which belong to the main task of the project, it is presented a general idea on the construction of the reliability function, giving the formula and the pseudocode to try to find the most reliable areas in the KG. Then all the experiments done during the thesis project are described, to understand the steps of the work done starting from the initial approach to the subject up to techniques to work better with embeddings. Eventually, the results obtained from the main experiments and the conclusions drawn from the work carried out will be shown.

Chapter 1

Related works

During the development of the work, the research of similar projects was a fundamental step to find information on the characteristics of the networks, with approaches and techniques used by researchers to carry out tasks similar to those mentioned in the previous section.
Starting from the general concept of a **knowledge graph** [1], it was important to understand the main features of these powerful semantic models, considering various aspects including knowledge graph representation learning, knowledge acquisition and completion, temporal knowledge graph, and knowledge-aware applications, reviewing recent breakthroughs and perspective directions for the future.
A knowledge graph is a structured representation of facts, consisting of entities, relationships, and semantic descriptions. Entities can be real-world objects and abstract concepts, relationships represent the relation between entities, and semantic descriptions of entities, and their relationships contain types and properties with a well-defined meaning. Property graphs or attributed graphs are widely used, in which nodes and relations have properties or attributes. A knowledge graph can be viewed as a graph when considering its graph structure [2]. When it involves formal semantics, it can be taken as a knowledge base (KB) for interpretation and inference over facts [3]. The term of knowledge graph is synonymous with knowledge base with a minor difference. It is possible to view a KG as a visual representation of a KB, defined as a set of sentences/facts. In the context of KGs, we also have a task/problem similar to the inference (or knowledge reasoning) one in the context of KBs, which is known graph completion, which can be divided into other subtasks, like entity prediction, relation prediction and triple classification, which use knowledge graph embeddings.
Furthermore, it was considered the concept of **graph embedding** [4], an effective yet efficient way to solve the graph analytics problem. In this paper have been considered several aspects to manage the embeddings: specifically, graph embedding

converts a graph into a low-dimensional space in which the graph information is preserved. By representing a graph as a set of low-dimensional vectors, graph algorithms can then be computed efficiently. There are different types of graphs (e.g., homogeneous graph, heterogeneous graph, attribute graph, etc), so the input of graph embedding varies in different scenarios. The output of graph embedding is a low-dimensional vector representing a part of the graph (or a whole graph).
Particularly, graph embedding aims to represent a graph as low-dimensional vectors while the graph structures are preserved. On the one hand, graph analytics aims to mine useful information from graph data. On the other hand, representation learning obtains data representations that make it easier to extract useful information when building classifiers or other predictors [5].
It is also useful to consider how embeddings behave if a knowledge graph is passed as input, and how it represents the whole structure in the embedding space.
Knowledge graph embedding [6] is organized from four aspects of representation space, scoring function, encoding models, and auxiliary information. KGE provides versatile techniques for representing knowledge. These techniques can be used in a variety of applications such as completion of knowledge graph to predict missing information, recommender systems, question answering, query expansion, etc. Also presented are the main **knowledge graph embedding models**, which have different techniques to represent embeddings starting from the triples of KG.
All these main components will be presented in the next chapter, going deeper into the concepts just treated and trying to give a complete view regarding the set of elements involved in this topic.

Chapter 2

Background

This chapter aims to provide a general background for the experiments carried out during the project, analyzing the main concepts about the structures used in the work, and also describing the embeddings and the models. Moreover, characteristics of elements concerning knowledge graphs will be cited, even if they have not been fully dealt with in this work, to give a complete perspective on this topic.

2.1 Knowledge Graph

Starting from the representation of the data through a complex network and how it is built, a **knowledge graph** (KG), also known as a semantic network, represents a network of real-world entities (i.e. objects, events, situations, or concepts) and illustrates the relationship between them. This information is usually stored in a graph database and visualized as a graph structure. A knowledge graph is made up of three main components: **nodes**, **edges** and **labels**. Anything can act as a node, for example, people, company, computer, etc. An edge connects a pair of nodes and captures the relationship of interest between them, for example, friendship relationship between two people, customer relationship between a company and person, or a network connection between two computers. The labels capture the meaning of the relationship, for example, the friendship relationship between two people. More formally, given a set of **entities** E, and a set of **relations** R, a knowledge graph is a subset of the cross product $E \times R \times E$. Each member of this set is referred to as a **triple**, and can be visualized as shown below.

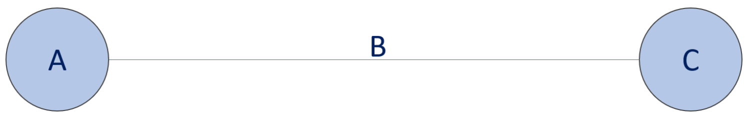


Figure 2.1. Triple composed of head entity A, relation B and tail entity C.

Triples are defined as (h, r, t), which represents the existence of a relation r between the head entity h and the tail entity t.
Triples can be represented in directed or undirected KGs. In the first case, supposing that the direction of the relation goes from the head entity to the tail one, it means that the relationship under consideration only makes sense in one way, capturing asymmetric relations.
The directed graph representation is used in a variety of ways depending on the needs of an application. A directed KG such as the one in which the nodes are people, and the edges capture friendship relationship is also known as a data graph. A directed graph in which the nodes are classes of objects (e.g., Book, Textbook, etc.), and the edges capture the subclass relationship, is also known as a taxonomy. Many interesting computations over graphs can be reduced to navigation. For example, in a friendship knowledge graph, to calculate the friends of a friends of a person A, it can be possible to navigate the KG from A to all nodes B connected to it by a relation labeled as friend, and then recursively to all nodes C connected by the friend relation to each B. The friendship knowledge graph may be an undirected KG, which captures symmetric relations between entities. In this case, considering a relation between two entities, it is supposed that if an entity A is a friend of an entity B, at the same time B is a friend of A.
The majority of KGs are directed and heterogeneous, so the number of different types of relation is variable, and also the relations themselves are asymmetrical to facilitate possible paths from a source node to a destination (as for web searches).

2.2 Graph Embedding

In this section the aim is to explain what is the **graph embedding**, going more in depth during this chapter to give more information about this method applied to different types of graph.
The problem of graph embedding is related to two traditional research problems, i.e., representation learning [5] and graph analytics [7]. Particularly, graph embedding aims to represent a graph as low-dimensional vectors while the graph structures are preserved. On the one hand, graph analytics aims to mine useful information from graph data. On the other hand, representation learning obtains data representations that make it easier to extract useful information when building classifiers or other

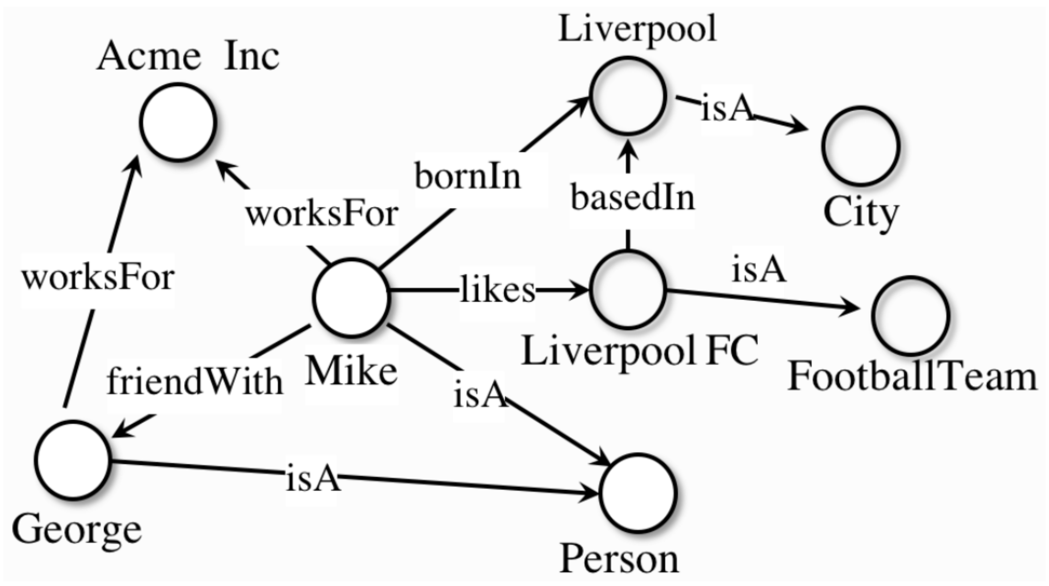


Figure 2.2. Example of a directed KG.

predictors [5]. Graph embedding lies in the overlap of the two problems and focuses on learning the low-dimensional representations. Note that we distinguish graph representation learning and graph embedding in this survey. Graph representation learning does not require the learned representations to be low-dimensional. For example, [8] represents each node as a vector with dimensionality equals to the number of nodes in the input graph. Every dimension denotes the geodesic distance of a node to each other node in the graph. Embedding graphs into low-dimensional spaces is not a trivial task. The challenges of graph embedding depend on the problem setting, which consists of embedding input and embedding output. In this survey, we divide the input graph into four categories, including homogeneous graph, heterogeneous graph, graph with auxiliary information and graph constructed from non-relational data. Different types of embedding input carry different information to be preserved in the embedded space and thus pose different challenges to the problem of graph embedding. For example, when embedding a graph with structural information only, the connections between nodes are the target to be preserved. However, for a graph with node label or attribute information, the auxiliary information provides graph property from other perspectives, and thus may also be considered during the embedding. Unlike embedding input which is given and fixed, the embedding output is task driven. For example, the most common type of embedding output is node embedding which represents close nodes as similar vectors. Node embedding can benefit node related tasks such as node classification, node clustering, etc. However, in some cases, the tasks may be related to higher granularity of a graph e.g., node pairs, subgraph, whole graph. Hence, the first challenge in terms of embedding output is to find a suitable embedding output type for the application of interest. So it is possible to categorize four types of graph embedding output, including *node embedding*, *edge embedding*, *hybrid embedding* and *whole-graph embedding*.

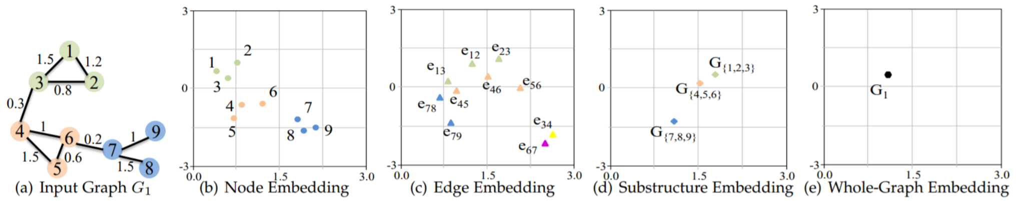


Figure 2.3. Types of graph embedding output.

Node embedding represents each node as a vector in a low-dimensional space. Nodes that are *close* in the graph are embedded to have similar vector representations. The differences between various graph embedding methods lie in how they define the closeness between two nodes, with two commonly adopted metrics for pairwise node similarity calculation (first-order proximity and second-order proximity).
In contrast to node embedding, *edge embedding* aims to represent an edge as a low-dimensional vector. This type of embedding is very useful in two scenarios: the first regards KGs, so considering an edge as a triple (h, r, t), the embedding is learnt to preserve r between h and t in the embedded space, so that a missing entity/relation can be correctly predicted given the other two components in (h, r, t). Secondly, some works embed a node pair as a vector feature to either make the node pair comparable to other nodes or predict the existence of a link between two nodes. Therefore edge embedding benefits edge (or node pairs) related graph analysis, such as link prediction, knowledge graph entity/relation prediction, etc.
Hybrid embedding (or substructure embedding) is the embedding of a combination of different types of graph components, e.g, a substructure composed by nodes and edges, or a community. The embedding of substructure or community can also be derived by aggregating the individual node and edge embedding inside it. However, such a kind of an indirect approach approach is not optimized to represent the structure. Moreover, node embedding and community embedding can reinforce each other.
The last type of output is the *whole graph embedding* usually for small graphs, such as proteins, molecules, etc. In this case, a graph is represented as one vector and two similar graphs are embedded to be closer. Whole-graph embedding benefits the graph classification task by providing a straightforward and efficient solution for calculating graph similarities. To establish a compromise between the embedding

time (efficiency) and the ability to preserve information (expressiveness), some works design a hierarchical graph embedding framework. They think that accurate understanding of the global graph information requires the processing of substructures in different scales.
Different output granularities have different criteria for a *good* embedding and face different challenges. For example, a good node embedding preserves the similarity to its neighbouring nodes in the embedded space. In contrast, a good whole-graph embedding represents a whole graph as a vector so that the graph-level similarity is preserved. In observations of the challenges faced in different problem settings, we propose two taxonomies of graph embedding work, by categorizing graph embedding literature based on the problem settings and the embedding techniques. These two taxonomies correspond to what challenges exist in graph embedding and how existing studies address these challenges. In particular, we first introduce different settings of graph embedding problem as well as the challenges faced in each setting. Then we describe how existing studies address these challenges in their work, including their insights and their technical solutions. Note that although a few attempts have been made to survey graph embedding ([9], [10], [11]), they have the following two limitations. First, they usually propose only one taxonomy of graph embedding techniques. None of them analyzed graph embedding work from the perspective of problem setting, nor did they summarize the challenges in each setting. Second, only a limited number of related work are covered in existing graph embedding surveys. E.g., [9] mainly introduces twelve representative graph embedding algorithms, and [11] focuses on knowledge graph embedding only. Moreover, there is no analysis on the insight behind each graph embedding technique. A comprehensive review of existing graph embedding work and a high level abstraction of the insight for each embedding technique can foster the future researches in the field.

2.3 Applications

Graph embedding benefits a wide variety of graph analytics applications as the vector representations can be processed efficiently in both time and space. In this chapter, the graph embedding enabled applications are categorized as *node related*, *edge related* and *graph related*. The applications that will be presented in the next sections will serve to provide a general framework on the subject. Not all will be taken into account during the project, since mainly the work done was carried out on the triples of the KG (and not only on the nodes, or on the structure of the graph in its entirety), so the focus is more on the edge related applications, more precisely to the link prediction task as previously mentioned.

2.3.1 Node Related Applications

This type of applications concerns strategies to adopt when the focus is on the nodes of the graph. Then it is possible to summarize the node related applications that graph embedding enables:

- **Node Classification**
  Node classification is the task which assigns a class label to each node in a graph based on the rules learnt from the labelled nodes. Intuitively, similar nodes have the same labels. In general, each node is embedded as a low-dimensional vector. Node classification is conducted by applying a classifier on the set of labelled node embedding for training. The example classifiers include SVM [12], logistic regression [13] and k-nearest neighbour classification [14]. Then given the embedding of an unlabelled node, the trained classifier can predict its class label.

- **Node Clustering**
  Node clustering aims to group similar nodes together, so that nodes in the same group are more similar to each other than those in other groups. As an unsupervised algorithm, it is applicable when the node labels are unavailable. After representing nodes as vectors, the traditional clustering algorithms can then be applied on the node embedding. Many existing works adopt k-means as the clustering algorithm.

- **Node Recommendation/Retrieval/Ranking**
  This task aim to recommend top K nodes of interest to a given node based on certain criteria such as similarity. In real-world scenarios, there are various types of recommended node, such as research interests for researchers, items for customers, and friends for social network users. It is also popular in community-based question answering, because given a question, they predict the relative rank of users or answers. They also rank the nodes of a particular type (e.g., "user") for a given query node (e.g., "Bob") and a proximity class (e.g., "schoolmate"), e.g., ranking users who are the schoolmates of Bob. A specific application which is popularly discussed in knowledge graph embedding is entity ranking [15]. Entity ranking aims to rank the correct missing entities given the other two components in a triple higher than the false entities. E.g., it returns the true h's among all the candidate entities given r and t, or returns the true t's given r and h.

2.3.2 Edge Related Applications

Now there will be introduced edge related applications in which an edge or a node pair is involved.

- **Link Prediction**
  Graph embedding aims to represent a graph with low-dimensional vectors, but interestingly its output vectors can also help infer the graph structure. In practice, graphs are often incomplete, for example in social networks where friendship links can be missing between two users who actually know each other. In graph embedding, the low-dimensional vectors are expected to preserve different orders of network proximity, as well as different scales of structural similarity. Hence, these vectors encode rich information about the network structure, and they can be used to predict missing links in the incomplete graph. Most attempts on graph embedding driven link prediction are on homogeneous graphs [16]. Relatively fewer graph embedding work deals with heterogeneous graph link prediction. For example, on a heterogeneous social graph, ProxEmbed [17] tries to predict the missing links of certain semantic types (e.g., schoolmates) between two users, based on the embedding of their connecting paths on the graph. D2AGE [18] solves the same problem by embedding two users connecting directed acyclic graph structure.

- **Triple Classification**
  Triple classification [19] is a specific application for knowledge graph. It aims to classify whether an unseen triple (h, r, t) is correct or not, i.e., whether the relation between h and t is r.

2.3.3 Graph Related Applications

- **Graph Classification**
  Graph classification assigns a class label to a whole graph. This is important when the graph is the unit of data. In most cases, whole-graph embedding is applied to calculate graph level similarity [20], and recently some work starts to match node embedding for graph similarity. Each graph is represented as a set of node embedding vectors. Graphs are compared based on two sets of node embedding, then a graph is decomposed into a set of substructures, embedding each substructure as a vector and comparing graphs via substructure similarities.

- **Graph Visualization**
  Graph visualization generates visualizations of a graph on a low-dimensional space [21]. Usually, for visualization purpose, all nodes are embedded as 2D vectors and then plotted in a 2D space with different colours indicating nodes' categories. It provides a vivid demonstration of whether nodes belonging to the same category are embedded closer to each other.

2.4 Knowledge Graph Embedding

Knowledge graph embedding (KGE), also referred to as knowledge representation learning, is a machine learning task of learning a low-dimensional representation of a knowledge graph's entities and relations while preserving their semantic meaning. Leveraging their embedded representation, knowledge graphs can be used for most of the applications listed in the previous chapter.

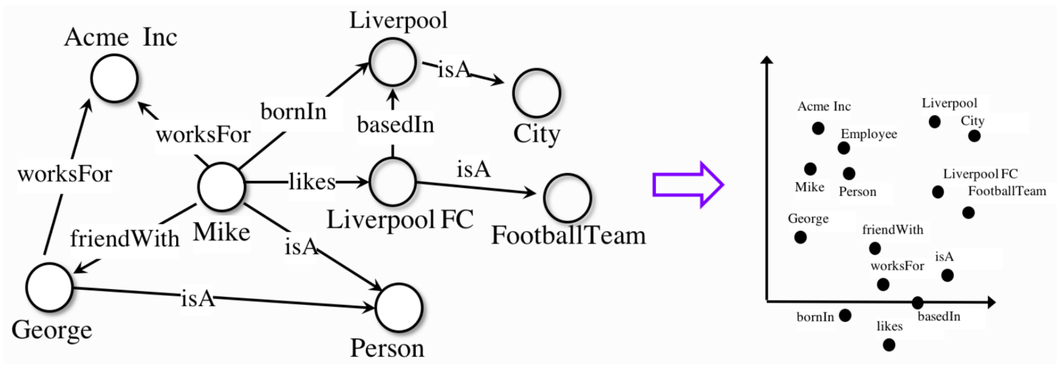


Figure 2.4. Generation of knowledge graph embeddings.

The embedding of a knowledge graph translates each entity and relation of a knowledge graph into a vector of a given dimension *d*, called embedding dimension. In the general case, we can have different embedding dimensions for the entities and the relations. The collection of embedding vectors for all the entities and relations in the KG are a more dense and efficient representation of the domain that can more easily be used for many different tasks.
A knowledge graph embedding is characterized by four different aspects:

- **Representation space**: The low-dimensional space in which the entities and relations are represented
- **Scoring function**: A measure of the goodness of a triple embedded representation
- **Encoding models**: The modality in which the embedded representation of the entities and relations interact with each other

- **Additional information**: Any additional information coming from the knowledge graph that can enrich the embedded representation. Usually, an ad hoc scoring function is integrated into the general scoring function for each additional information.

Before going into the structural details of the embedding models, it is interesting to present the procedure to learn the semantic meaning of the facts, so this algorithm is followed by all the models.
First of all, to learn an embedded representation of a knowledge graph, the embedding vectors of the entities and relations are initialized to random values. Then, starting from a training set until a stop condition is reached, the algorithm continuously optimizes the embeddings. Usually, the stop condition is given by the overfitting over the training set. For each iteration, is sampled a batch of size b from the training set, and for each triple of the batch is sampled a random corrupted fact (a triple that does not represent a true fact in the knowledge graph). The corruption of a triple involves substituting the head or the tail (or both) of the triple with another entity that makes the fact false. The original triple and the corrupted triple are added in the training batch, and then the embeddings are updated, optimizing a scoring function. At the end of the algorithm, the learned embeddings should have extracted the semantic meaning from the triples and should correctly unseen true facts in the knowledge graph.

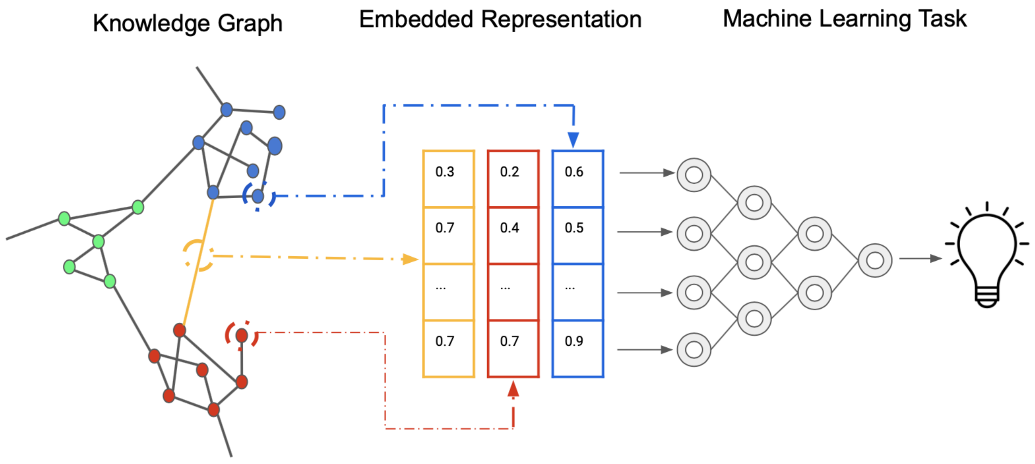


Figure 2.5. Embedding of a knowledge graph. The vector representation of the entities and relations can be used for different machine learning applications.

2.5 KGE Models

Knowledge graph embedding models (KGEMs) learn latent vector representations of the entities $e \in E$ and relations $r \in R$ in a KG that best preserve its structural properties. Besides for link prediction, they have been used for tasks such as entity disambiguation, and clustering as well as for downstream tasks such as question answering, recommendation systems, and relation extraction. These approaches are broadly classified into two main groups: *translation models* and *semantic matching models*.

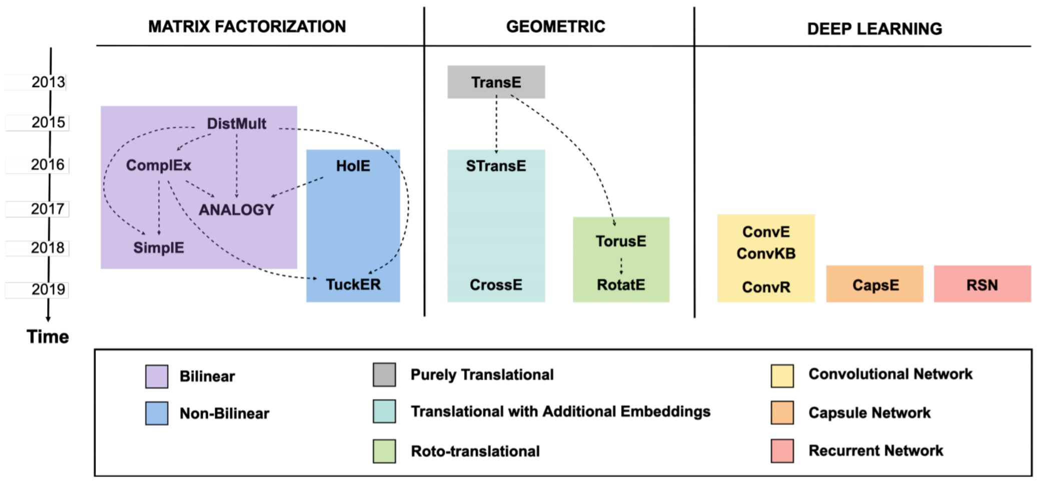


Figure 2.6. Taxonomy for the main KGEMs.

Translation-based models use distance-based measures to generate the similarity score for a pair of entities and their relationships. These models aim to find a vector representation of entities with relation to the translation of the entities. It maps entities to a low-dimensional vector space.
Semantic matching models use similarity-based scoring function. These models implicitly consider the KG as a 3D adjacency matrix (that is, a 3-way tensor), that is only only partially observable due to the KG incompleteness. The tensor is decomposed into a combination (e.g. a multi-linear product) of low-dimensional vectors: such vectors are used as embeddings for entities and relations.
There are also *deep learning models* which use deep neural networks to perform the LP task. Neural Networks learn parameters such as weights and biases, that they combine with the input data in order to recognize significant patterns. Deep neural networks usually organize parameters into separate layers, generally interspersed with non-linear activation functions.

For this project the focus is on **TransE** and **DistMult** models. The first, deducing from its name, belongs to the translation-based models, while the second uses the technique of matrix factorization to represent the embeddings.

2.5.1 TransE

The first model chosen is **TransE**, an energy-based model that produces knowledge base embeddings. It models relationships by interpreting them as translations operating on the low-dimensional embeddings of the entities. Relationships are represented as translations in the embedding space: if (h, r, t) holds, the embedding of the tail entity t should be close to the embedding of the head entity h plus some vector that depends on the relation r. It can be graphically represented as Figure 2.7.

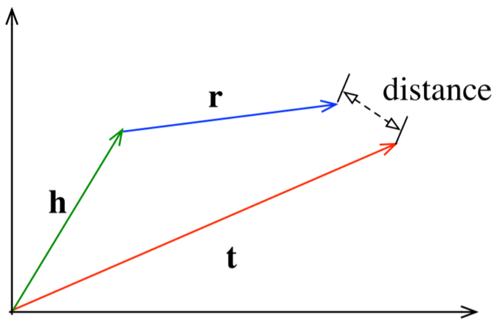


Figure 2.7. TransE.

Mathematically, this model can be stated like $e_h + e_r \approx e_t$. This equation is rearranged and the l_p norm is applied to create the TransE interaction function.

$$f(h, r, t) = -||e_h + e_r - e_t||_p$$

While this formulation is computationally efficient, it inherently cannot model one-to-many, many-to-one, and many-to-many relationships. For triples $(h, r, t_1), (h, r, t_2) \in K$ where $t_1 \neq t_2$, the model adapts the embeddings in order to ensure $e_h + e_r \approx e_{t_1}$ and $e_h + e_r \approx e_{t_2}$ which results in $e_{t_1} \approx e_{t_2}$.

2.5.2 DistMult

DistMult is a simplified RESCAL, which uses the basic bilinear scoring function.

$$f_r(h,t) = \mathbf{h}^T \mathbf{M}_r \mathbf{t}$$

These bilinear formulations are combined with different forms of regularization to make different models. In DistMult authors considered a simpler approach where they reduced the number of parameters by imposing restrictions on M_r to be a diagonal matrix. This results in a simpler model and this model enjoys the same scalable properties of TransE as well as it achieves better performance over TransE. Thus the final scoring function is given as

$$f_r(h,t) = \mathbf{h}^T diag(\mathbf{r})\mathbf{t} = \sum_{i=0}^{d-1} [\mathbf{r}]_i \cdot [\mathbf{h}]_i \cdot [\mathbf{t}]_i$$

where for each relation r, $r \in \mathbb{R}^d$ is a vector that depends on relationships.

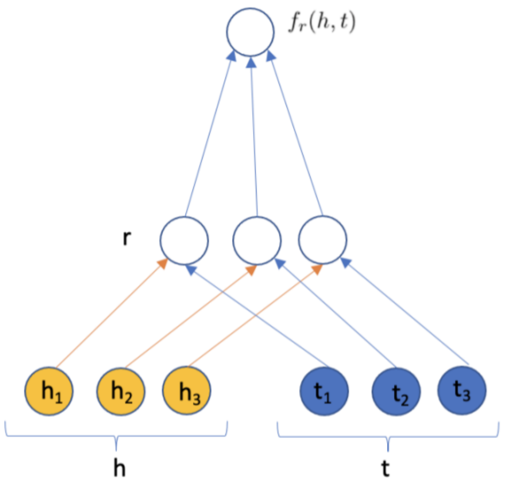


Figure 2.8. DistMult.

Because of its restriction to diagonal matrices, DistMult is more computationally than RESCAL, but at the same time it is less expressive. For instance, it is not able to model anti-symmetric relations, since $f(h,r,t) = f(t,r,h)$. This can alternatively be formulated with relation vectors $\mathbf{r}_r \in \mathbb{R}^d$ and the Hadamard operator and the l_1 norm.

$$f(h,r,t) = ||e_h \odot e_r \odot e_t||_1$$

2.6 Evaluation Metrics for KGEMs

KGEMs are usually evaluated based on link prediction, which is on KG defined as predicting the tail/head entities for $(h, r)/(r, t)$ pairs. However, given the fact that usually true negative examples are not available, both the training and the test set contain only true facts. For this reason, the evaluation procedure is defined as a ranking task in which the capability of the model to differentiate corrupted triples from known true triples is assessed.
These indexes are often used to measure the embedding quality of a model. The simplicity of the indexes makes them very suitable for evaluating the performance of an embedding algorithm even on a large scale. Given the set Q of all ranked predictions of a model, it is possible to define three different evaluation metrics: *MR*, *AMR*, *MRR* and *Hits@K*.

- **Mean rank**
  The mean rank (MR) represents the average rank of the test triples, i.e.,

$$MR = \frac{1}{|Q|} \sum_{q \in Q} q$$

  Smaller values indicate better performance.

- **Adjusted mean rank**
  Because the interpretation of the MR depends on the number of available candidate triples, comparing MRs across different datasets (or inclusion of inverse triples) is difficult. Therefore, with fewer candidates available, it becomes easier to achieve low ranks. The adjusted mean rank (AMR) compensates for this problem by comparing the mean rank against the expected mean rank under a model with random scores:

$$AMR = \frac{MR}{\frac{1}{2}\sum_{q \in Q}(\xi(q) + 1)}$$

  where $\xi(q)$ denotes the number of candidate triples against which the true triple $q \in Q$ is ranked. The AMR has a fixed value range from 0 to 1, where smaller values ($AMR \ll 1$) indicate better performance.

- **Mean reciprocal rank**
  Mean reciprocal rank (MRR) measures the number of triples predicted correctly. If the first predicted triple is correct, then 1 is added, if the second is correct $\frac{1}{2}$ is summed, and so on. MRR is defined as:

  $$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{q}$$

  where Q is a set of test triples, i.e. the MRR is the mean over reciprocal individual ranks. However, the MRR is flawed since the reciprocal rank is an ordinal scale and not an interval scale, i.e. computing the arithmetic mean is statistically incorrect. Still, it is often used for early stopping since it is a smooth measure with stronger weight on small ranks, and less affected by outlier individual ranks than the mean rank. The MRR has a fixed value range from 0 to 1, where larger values indicate better performance.

- **Hits@K**
  Hits@K ($H@K$) denotes the ratio of the test triples that have been ranked among the top K triples, i.e.,

  $$H@K = \frac{|\{q \in Q \mid q \leq K\}|}{|Q|}$$

  Common values for K are 1, 3, 5, 10. The higher the $H@K$, the better the model results. In particular, when $K = 1$, it measures the ratio of the test facts in which the target was predicted correctly on the first try. $H@1$ and MRR are often closely related, because these predictions also correspond to the most relevant addends to the MRR formula.

There are also other additional metrics like the area under the Receiver Operating Characteristic curve (AUC-ROC) and the area under the precision-recall curve (AUC-PR), but since these metrics require the number of true positives, false positives, true negatives, and false negatives, in most cases they cannot be computed because the KGs are usually incomplete.

Chapter 3

Reliability

The goal of the project is to investigate the **reliability** of knowledge graph embedding models. In particular, there is the need to find the areas of the KG in which those models evaluate triples in the right way.
To accomplish this task, it is necessary to do some considerations under certain assumption. First of all, the knowledge graph under investigation is treated assuming the **Closed World Assumption** (CWA), which means that the absence of a fact means it is necessarily false. In this case, only the existing triples in the KG are considered as positives, while all the others (that will be reconstructed) are considered negatives. Only for completeness, the other possible assumption could be the *Open World Assumption* (OWA), in which the absence of a fact does not imply fact is false (we simply do not know), and this would imply greater difficulty in identifying reliable areas in the graph, so it was initially decided to work with the CWA to try to facilitate experiments and general work.
Then the KG under examination is going to be complete, weighted, e-r-e graph containing all possible relations, with weights of the edges equivalent to the scoring function of a considered embedding model $f_r(h,t)$, the **reliability** $R(K)$ is defined as follows:

$$R(K) = \frac{1}{N}\left[\sum_{(h,r,t)\in K} f_r(h,t) + \sum_{(h,r,t)\notin K} (1 - f_r(h,t))\right]$$

Where K is a knowledge graph (provided as a set of triples), N is the total number of triples (positives and negatives), and (h,r,t) represents a triple composed of head entity, relation, tail entity.
This formula represent the mean of all the triples' scores, and it is expected to obtain a high value for a reliable subgraph, in terms of prediction of missing information. Triples scores are computed through the scoring function, which changes according

to the chosen model. The first summation is for positive triples (those present in the original KG), so a good embedding model is able to accurately evaluate that type of triples, assigning them a score (presumably) very high, and then indicating that they will also be present in the outgoing KG. The second summation, instead, is for all the triples not present in the original KG, therefore the model could evaluate them with a low score.
In order for the formula to make sense, scores must be contained in the range [0,1]. If an embedding model generate scores in another interval, normalize the values in [0,1]. For this reason, the argument in the second summation takes into account this important information, since that the constant minus the triple score is the attempt to derive a high value from there. Eventually, all the summed scores will be divided for N, so the final result will in turn be a score of the entire KG or a part thereof. Furthermore, the K parameter could be a subgraph, so the reliability could be computed only on a part of the knowledge graph.
Considering the reliability function, it is necessary to use a link prediction strategy to compare the **reliability** of a subgraph with the **accuracy** of a classifier on the subgraph triples.
Then it is implemented the pseudocode as follows:

```
• Split the input $KG$ into $TRAIN_E$ (triples to be used for training the embeddings) and $LP^+ = KG \setminus TRAIN_E$ (positive examples for the link-prediction classifier)
• Define train set and test set for the ultimate link-prediction task
    – Define $LP^-$ by sampling $|LP^+|$ non-existing triples from $KG$
    – Split $LP^+$ into $TRAIN_{LP^+}$ and $TEST_{LP^+}$
    – Split $LP^-$ into $TRAIN_{LP^-}$ and $TEST_{LP^-}$
    – $TRAIN_{LP} = TRAIN_{LP^+} \cup TRAIN_{LP^-}$
    – $TEST_{LP} = TEST_{LP^+} \cup TEST_{LP^-}$
• Train embeddings on $TRAIN_E$
• Train a classifier for Link Prediction on $TRAIN_{LP}$ (with embeddings used as feature vectors, e.g., letting the feature vector of triple $t$ correspond to the concatenation of the embeddings of $t$'s subject, $t$'s predicate, and $t$'s object)
```

- Test the Link Prediction classifier. In particular:
  - Pick integers for the size of each subgraph (k), and the number of subgraphs to be sampled (h)
  - Sample h subgraphs $S_1, ..., S_h$ of size (number of nodes) k from KG
  - For every subgraph S_i, $i = 1, ..., h$:
    - Compute accuracy ACC_i of the link-prediction classifier on $TEST_{LP} \cup S_i$
    - Compute the reliability score REL_i of subgraph S_i
    - Show correlation among the set $\{ACC_i\} i = 1^h$ of accuracy scores and the set $\{REL_i\} i = 1^h$ of reliability scores

Ideally, such a correlation should show *high accuracy* $\iff$ *high reliability*. Reliability is unlikely to report very high values even for very dense subgraphs, as the results are calculated from the scoring function of an embedding model, which could be more or less accurate. For this reason the reliability value is compared with the classifier accuracy, so if you can verify that the two values are closely related, then the general formula can be defined as valid.

Chapter 4

Experiments/Tests

In this chapter will be presented experiments and tests developed during the project. It is necessary to remember that all tests are aimed at understanding whether the hypothesis made on the concept of reliability can be validated. In other words, as explained in the initial part of this report, the assumption is made that the embedding models work well, evaluating correctly the triple positive and negative, so with different approaches and different techniques the goal is to find the areas of the graph in which these models work best.
The work was done on **Google Colab** [1], a product which allows to code with Python and provides free access to computing resources (including GPUs). This was almost an obligatory choice, as most of the experiments were carried out on very large knowledge graphs, so the use of resources in the cloud instead of those in local favored the development time of the entire project.
As with most papers dealing with this vast topic, the **PyKEEN**[2] library is also used in this work. PyKEEN is a Python package designed to train and evaluate knowledge graph embedding models, and it is very useful since it contains many datasets describing various KGs, including Freebase and WordNet. It also contains many of the main embedding models, and all the functions that allow us to explore KGs.
To create and manipulate graphs it was necessary to run to the help of **NetworkX**[3], which allows to perform all the operations of navigation within complex structures thanks to the functions implemented in the library.
This chapter will be divided into three sections, representing three main objectives set during the experimental phase. The first phase concerns a first approach to the matter, making considerations more to understand the functions to use and the

[1]https://colab.research.google.com/
[2]https://pykeen.readthedocs.io/
[3]https://networkx.org/

models to choose.
The second phase is focused on finding new techniques to use when working with a knowledge graph, such as the introduction of a new set of corrupted triples and a sampling of the graph developed through a random walk, for the localization of denser subgraphs than before. This phase is aimed only at the study of reliability and of KGs in general, therefore the LP classifier is not modified.
In the last phase the attempt is to approach the results in a more realistic scenario, trying to work with larger KGs and improving the techniques treated up to this point. In addition, various classification models are explored to better assess the correlation between accuracy and reliability measurement.

4.1 Phase one

The initial part of whole the experiments regards a first approach to the embeddings models. Specifically, during phase one the attempt was to learn the PyKEEN functions with which it has been extrapolated a public dataset, trying to represent the triples of the KG in structures (such as sets or lists) that will then be used to train the embeddings of entities and relations.
Starting from the pseudocode shown in Section 3, it is necessary to start working with a knowledge graph and then divide its triples for the reliability function and the LP task. Two relatively small graphs have been chosen from the list of library datasets: **Nations** and **Kinships**. The Nations [22] dataset contains data about countries and their relationships with other countries. The Kinships [23] dataset describes relationships between members of the Australian tribe *Alyawarra* and consists of 10,686 triples. It contains 104 entities representing members of the tribe and 26 relationship types that represent kinship terms. The choice is given by the fact that not knowing the behavior of the models in the evaluation of the KGs, at least for the initial part of the project, small graphs would have helped the execution of the code minimizing the total time of the process, and therefore maximizing the efficiency.

Table 4.1. Datasets chosen for the phase one.

Datasets			
Name	**Entities**	**Relations**	**Triples**
Nations	14	55	1992
Kinships	104	26	10686

Starting from the choice of KGs described above, triples are collected and then splitted into equal numbers: the first half ($TRAIN_E$) will be used to train the embeddings, while the second half (LP^+) will constitute the positive part of the entire dataset for the LP task, so for the ML classifier.
At this point there is a need to create the missing triples in the KG, generating all possible combinations of entity-relation-entity, therefore defining LP^- by sampling $|LP^+|$ non-existing triples from KG. The easiest way to go is to generate the combinations starting from the set of entities and relations of the knowledge graph, then the elements have been scanned through the iterative method, reconstructing the triple (h, r, t) composed by a head entity, a relation and a tail entity, and checking whether the generated triple was already contained in the original KG. Only if the triple is not present, it will be inserted in the set of negatives, until the cardinality of the latter set is equal to that of the set of positive triples. The possibility of generating all negative triples without any limit on the number of them was also

included, but limiting them to the same number of positive ones is an advantage for the LP task. Having a balanced dataset, the classifier manages to be more accurate in its evaluation, so the objective is to prepare the classifier and train it with a dataset which contains the same number of positives and negatives. So once the set of all the triples has been created, the dataset must be splitted into $TRAIN_{LP}$ and $TEST_{LP}$, therefore this procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. The $TRAIN_{LP}$ set with the size of 0.7 (70 percent) means that the remainder percentage 0.3 (30 percent) is assigned to the $TEST_{LP}$ set.
Once this operation is completed, it is necessary to perform the same procedure as for the LP classifier, so the $TRAIN_{E}$ set will be splitted into train and test, then finally it is possible to prepare the embedding model to the training phase. Also in this case, the 30 percent of samples are assigned to the test set. For this first approach to the embedding model, the decision was to use two of the main models: **TransE** and **DistMult**. As explained in Section 2.5, with TransE the relations are represented as translations in the embedding space, while DistMult simplifies RESCAL by restricting matrices representing relations as diagonal matrices. Since these KGEMs perform a different technique to represent the triples in the embedding space (and also to evaluate them), it is expected to achieve different results in terms of scores given to the various positive and negative triples. There is the possibility that the range of possible values assigned to the triples can also vary according to the parameters of the PyKEEN pipeline: at this time, the attempt is to train the embedding with the default parameters, verifying the evaluation made by the embedding models.
Now, once the models are trained, the reliability function has to be introduced. The function has been developed in two modes: the first concerns the reliability of the whole KG, while the second is calculated for each subgraph. Considering the latter option, the selection regards some areas of the KG keeping the number of nodes constant, for k subgraphs. Since that for this initial phase the two selected KGs have a small number of entities, especially in the case of the Nations dataset that contains only 14, repeating nodes in the various subgraphs is allowed. This means that several subgraphs may have some nodes in common, but the possibility that they are identical is excluded. So the strategy here is to start with the set of entities, and generate k combinations from there, so that is possible to collect all the triples within the considered substructure, and then take a complete subgraph (recovering the triples between the subgraph's nodes). To choose the subgraphs no random walks has been developed within the graph because as mentioned also at the beginning of this section, the current goal of this first experimental part is

to understand the features of the library in use, and also to have more conviction about the reliability hypothesis, verifying that for the moment the output results are in line with the expectation. Therefore the reliability function has been defined in such a way as to evaluate the triples starting from the model trained, using the scoring function of the model itself, so putting all these scores in an array and finally normalize the scores using the *min-max normalization*. It is defined as follows:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

Where x_i is the current score, x is the array of scores, and $max(x)$ and $min(x)$ represent the maximum and minimum values among all the scores.
Min-max normalization is one of the most common ways to normalize data. For every score, the minimum value of that score gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. Taking a step back, it is necessary to talk about the classifier describing the choice made regarding the type of classification and the construction of the dataset. As mentioned before, in addition to the set of triples present in the original KG were generated all missing triples, defining them as negative (since the *CWA* is followed). This implies that the classification model will have to work with two classes, which will represent exactly positives and negatives. So the task used to classify triples is the **binary classification**. For this type of classification task, the interest falls in the classification of data into one of two binary groups, where the class for the positive triples is assigned the class label 1 and the class with negative triples is assigned the class label 0. Since at this moment there is no information about the embedding that represents every single triple, since it has not been extrapolated from the model, we only know the labels of the elements that make up the triple (in terms of entities and relations) and the normalized score of each triple. So the *feature vectors* (i.e. n-dimensional vectors of numerical features that represent some objects) will not be composed by embeddings, but by the data listed above, with the additional parameter regarding the class. To do this, we have to start from the $TRAIN_{LP}$ and $TEST_{LP}$ sets created at the beginning of the process. The first check to be made on the triples of these sets is the belonging of entities in the current subgraph, so if the head entity and the tail entity are both nodes of the subgraph, then the triple will be taken into account for the construction of the dataset, otherwise it will be discarded. The second check, of course, is the membership of the triple in the original graph, so if the triple is positive it is labeled with 1, otherwise (if it is negative) it will have the label 0. If the number of positives and negatives is not balanced, we need to limit the number

of negatives and eventually performing again a 70/30 split on all the subgraph's triples.
Once the dataset is ready, only the choice of the binary classification algorithm remains. In machine learning there are many methods used for binary classification, one of the most common is the **logistic regression**, which is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. Since all the elements have been described, it is enough to train the classifier on the total set of triples and evaluate the triple of tests for each sub-graph. The accuracy is very high in this case, even more than 95%, but this is an information not to be taken into account since the model is probably overfitting. This problem will be partially solved in the second phase of the experiments, since the classifier will be slightly modified in its structure, for example by varying the feature vectors (which will be described by the embeddings). Also in phase two will be explored different possibilities for the sampling of the graph, in addition to testing the functions on larger datasets and considering a visit of the graph more accurate to extrapolate subgraphs that could turn out to be communities. In the next section will be listed all changes made, showing through plots the first experimental results of greatest relevance. In addition, other datasets will be presented, since with those used until now there was no way to evaluate the performance (which proved to be particularly insignificant) due to the fact that the graphs taken into consideration are very dense. KG sampling was also of little significance, so it's time to switch to larger graphs, also trying to perform a more accurate sampling that can be used for all the structures considered.

4.2 Phase two

The main objective of this phase is to improve the visit of the graph to find high density subgraphs and to explore other methods to generate missing triples, since no information has yet been collected about the distribution of sets of triples and this could also be useful to understand the results obtained by the reliability function and the classifier. Being able to get this additional information would also help to understand if the model is able to correctly evaluate the triples, and then to understand if there is a clear separation between the two sets (in terms of scores). Furthermore, an attempt is made to consider another class of triples (in addition to positive and negative ones), that is, those generated between two entities that already contain different relationships between them. This concept will be elaborated later in the section also to explain how it was integrated into the elements created in the initial phase of the project.
Therefore the focus in this part of the project is only on the attempt to make the reliability function more meaningful than before. This means that the experiments carried out at this stage are focused on exploring other possibilities of visiting the graph, to give input to the function a true subgraph, since with a classic combination generated from a list there is a risk that the nodes are far away, and probably not even connected. The fact of creating simple combinations and not making any visits was born from the use of small (and complete) graphs, but this led to other problems as results of little value. This premise has been made to explain that for this phase the classifier will not be modified, but the attention will be shifted on the reliability function and the elements that belong to this task.
To realize this need was considered another dataset, **Unified Medical Language System** (UMLS) [46], which is an ontology that describes relationships between high-level concepts in the biomedical domain. This dataset represents a sparse KG and it might be useful to analyze it to explore any good areas.

Table 4.2. Dataset chosen for the phase two.

Datasets			
Name	**Entities**	**Relations**	**Triples**
UMLS	135	46	6529

One of the first observations of this phase concerns the division of the dataset between the reliability function and the Link Prediction task. There is of course the possibility to consider different positive examples to train embeddings and LP classifier, but this implies having a small amount of examples for both tasks, especially if we think that very often you have in input a sparse knowledge graph, so a limited number of positive examples. Then it was decided not to do any initial split and to consider all triples for both tasks. Remember that in the case in which it is necessary to train the embeddings, are to consider only the positive triples, while to train the classifier for the LP task must be considered also those negatives. Another goal of the project is to find groups (or communities) within the graph, which would represent an area where you have a high number of relationships between the entities at stake. To be able to find these groups would be very helpful for the reconstruction of the missing edges, as dense subgraphs would lead to the idea of deducing new relations from those already existing. An experiment to verify this property is the construction of a toy graph, in which 2 communities are created with 100 nodes each. For each community there is a type of relationship, in this case there are 2 relations (named a and b) and will be connected within the respective community. The two groups are complete subgraphs, connected by an intermediate node to make a connected graph exist.

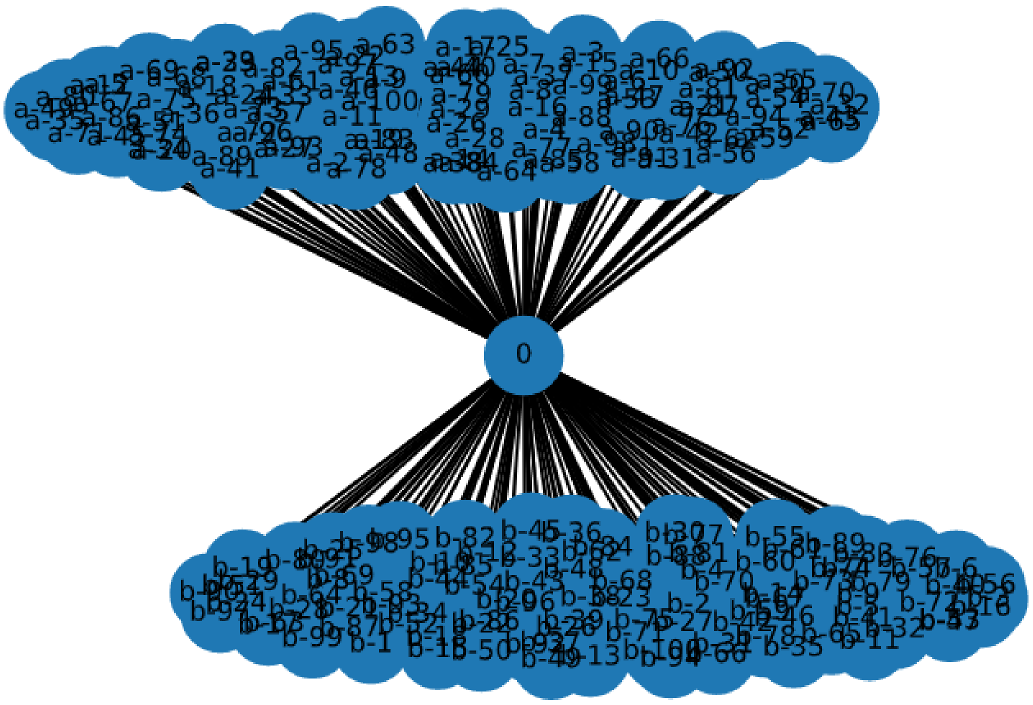
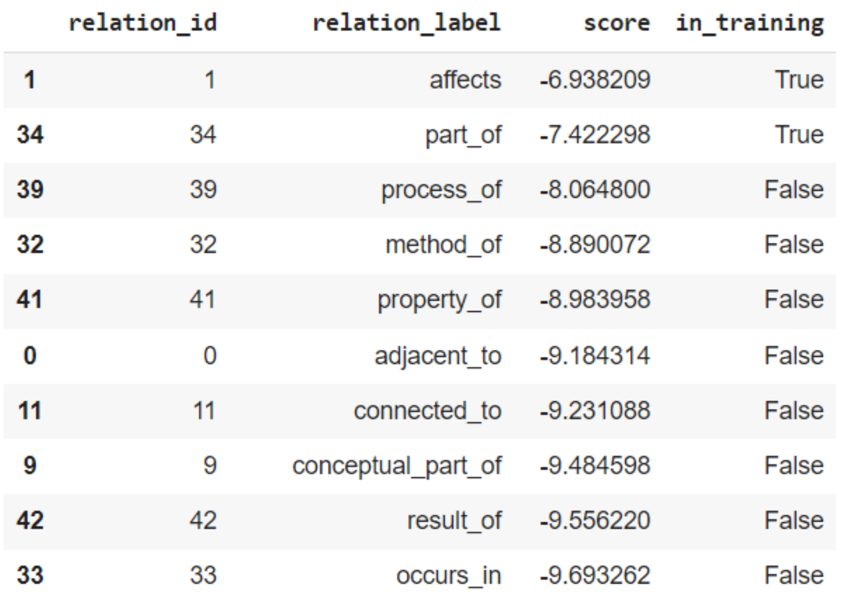


Figure 4.1. Distribution of triples' scores in the toy graph with TransE.

The intermediate node (the 0 one) is connected to all the other nodes. In the upper part it was designed a complete subgraph, so there are 100 nodes all connected to each other, with the only common relations referred to as a, consequently from the node 0 up to all the nodes present in the upper subgraph has been created the relation a. Similarly in the lower part all the nodes are connected by the relation b. The idea of building a graph of this type was born from the fact that, ideally, evaluating the model on a complete subgraph containing a single relation, should maximize the score distance between positive and negative triples (which in this case will belong to the different relation set, since all nodes are close together). In other words, following the theory described by the papers it is possible to rely on the embedding-based ranking method, which firstly learn embedding vectors based on existing triples. By replacing the existing relation with each relation $r \in R$, this method calculates scores of all the candidate entities and ranks the top k triples. For this particular case the ranking list contains only two triples (since there are only two different types of relation), and the first one should be the most likely of all.

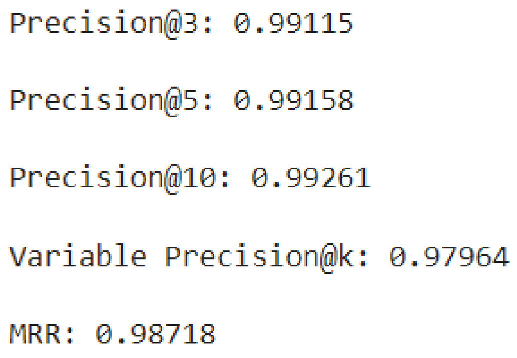


	relation_id	relation_label	score	in_training
1	1	affects	-6.938209	True
34	34	part_of	-7.422298	True
39	39	process_of	-8.064800	False
32	32	method_of	-8.890072	False
41	41	property_of	-8.983958	False
0	0	adjacent_to	-9.184314	False
11	11	connected_to	-9.231088	False
9	9	conceptual_part_of	-9.484598	False
42	42	result_of	-9.556220	False
33	33	occurs_in	-9.693262	False

Figure 4.2. Ranking of the top 10 relations between a pair of entities, in the UMLS dataset.

In many other cases, such as the one shown in the Figure 4.2, the list of the relationships between a certain pair of entities is shown. This list is sorted by the probability that the triple will be rebuilt, then by the score assigned to a given triple by the embedding model. This information could be of help in case you want to look for dense subgraphs exploiting the ranking, so starting from relationships with a high score it is very likely that these are reconstructed. So starting out in areas like that can help with the final work.
Also it can be possible to take advantage of the evaluation metrics to understand if the ranking is at least a reliable measure. Referring to the Chapter 2.6, different metrics can be used in the LP task. For this project the choice falls on *Precision@K* and *MRR*. The first one is built in two versions: the first one assigns to the parameter K fixed values, i.e. 3, 5 and 10, that are usually assigned values in these task types, while the second one uses the parameter K in a variable way. In this way for every ranking relative to a specific pair of entities, K comes set in base to the last positive

relation in the list.



```
Precision@3: 0.99115

Precision@5: 0.99158

Precision@10: 0.99261

Variable Precision@k: 0.97964

MRR: 0.98718
```

Figure 4.3. Evaluation metrics of the ranking list.

As mentioned at the beginning of this section, it was considered the possibility of inserting a part of the missing triples in a third set, in addition to those of the positive and negative. This set is called **different relation**, and contains the portion of negative triples that were generated between a pair of neighboring entities, so that they have at least one relation with each other. The set of negatives at this point contains the triples generated between two nodes that have no common relationship. The addition of this new set arose from the fact that the embedding model, knowing the positive triples and their related entities, should evaluate differently the elements present in the different relation set compared to the negative one. Triples which belong to this new set will be considered as negatives in the main two tasks, therefore for what regards the reliability function, they will be added to the second summation, since in general it is expected to get a negative score from that triples, while for the LP task they will be labeled with 0. The training part of the embedding models remained substantially the same. The concentration of work has always remained on the two selected models (TransE and DistMult) and it has been tried to play with the parameters of the PyKEEN pipeline, mainly varying the dimensionality of the embedding in output. In addition to this, other alternatives have been sought to use models, using other pipelines and exploiting functions of other libraries. Then a test was carried out on the comparison of the partial results obtained by the PyKEEN pipeline with those of another library, *AmpliGraph*[4]. It is a suite of neural machine learning models for relational Learning, a branch of machine learning that deals with supervised learning on knowledge graphs, and works in a very similar way to the library used until now. Once verified that both libraries incorporate the same models and scoring functions to evaluate the triples, then it was decided to leave the pipeline structure intact because already satisfactory in its results. Another improvement made at this stage is the visit of the graph to determine h subgraphs

[4]https://docs.ampligraph.org/

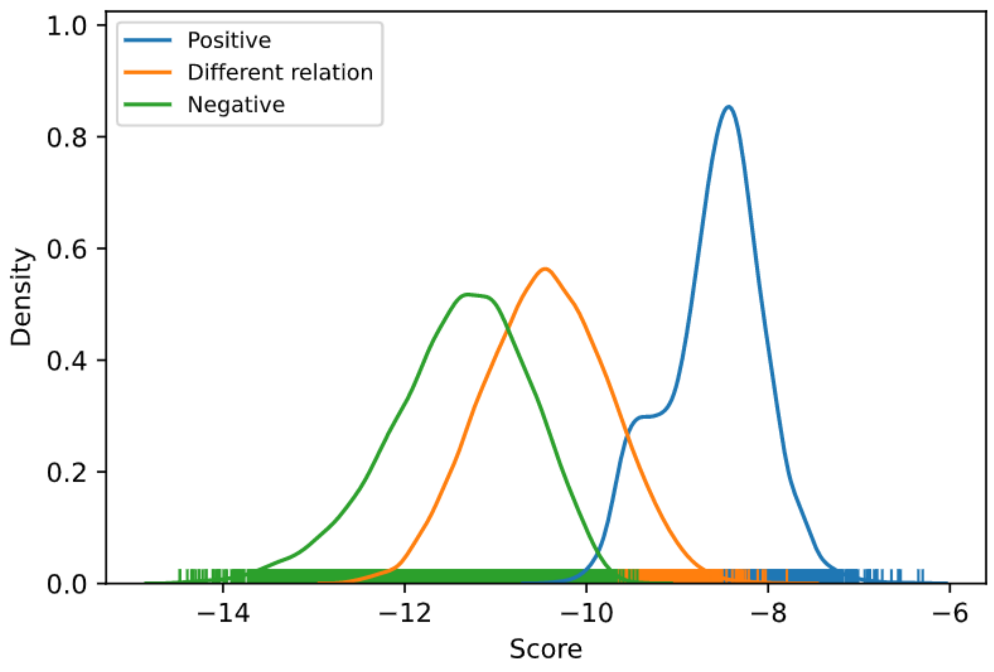


Figure 4.4. Introduced the different relation set. These distributions represent the scores given by TransE on the toy graph.

to be evaluated. Not knowing the structure of the KG it was necessary to choose the option that could generalize the path within the graph, to try to find areas good for the reliability function. The sampling used at the previous stage was constructed in such a way as to collect a fixed number of subgraphs by generating combinations from sets of entities and relations. This operation is inefficient for small graphs but especially for large graphs, so it is necessary to resort to an alternative that does not require the generation of combinations but a simple visit of the graph. The choice fell on a **random walk** strategy, in which starting from a node you are able to collect a graph structure composed of the number of nodes desired. The random walk will be defined as follows:

- Pick a random node in the graph
- Choose a neighbour with probability α or jump back to the starting node with probability $1 - \alpha$
- Continue until you have selected k nodes
- Return the induced subgraph of the k nodes

This sampling works much better than the previous one, both for running time but also for finding dense areas. The fact that it starts from a random node and takes the neighbour of the selected node causes it to come to create a path within the graph, then a connected component in which each entity has at least one relationship with another. The random walk built in this way can refer to a **depth-first search** (DFS), since the output results are also equal in their shape, as they are often trees and follow the same logic to be built. It was decided to collect a total of 100 subgraphs for each test performed, with number of entities ranging from 10 to 30. With more than 30 nodes there is the risk that it is not possible to find a compact structure, and that the target is not able to reach, for this reason the subgraphs (at least for this phase of experiments) must not have a very high size. That said, we can now move on to other considerations of reliability. The task carried out previously was insignificant, given the reasons expressed for the size of the datasets used and the strategies adopted for the visit of the graph. Now that ideally there are all the ingredients to improve the function, it can be made a try to explore other ways to calculate the reliability of subgraphs, always keeping as a reference point the general hypothesis described in the Chapter 3. Starting from the fact that the scores need to be processed to be in the range [0,1], the normalization could be done in various ways, especially if there are mathematical functions that allow this without elaborating an overly complex calculation or requiring additional elements. In addition to the min-max normalization, the idea of trying to apply a **sigmoid function** on every single score, such as the logistic function, has also popped into mind:

$$f(x) = \frac{1}{1 - e^{-x}}$$

Where x is the score of a triple.

Usually this function is used to transform numerical values into probabilities, but since the goal is to place the scores within the range [0,1] can also be used in this case (since the codomain of the function is equal to the desired range). Note that the function domain admits all values in the set of reals, but output values in (0,1) will only be given if the score is between -6 and 6. This means that a study must also be made on the score range of the embedding models, since choosing different models there is a risk that the output values are different depending on the model used. For example, since TransE and DistMult have different methods to represent embeddings, surely the output values will be different, but with high probability they will be correlated since the evaluation of the triples is almost similar. As you can see in the Figure 4.5, the cut-off point of the logistic function is equal to 0.5, this means that the values that will be found above this value will be positively considered from the model (for the greater part of the cases) while those below will be assessed negatively.

So right now there are two different methods to show reliability, which only vary in

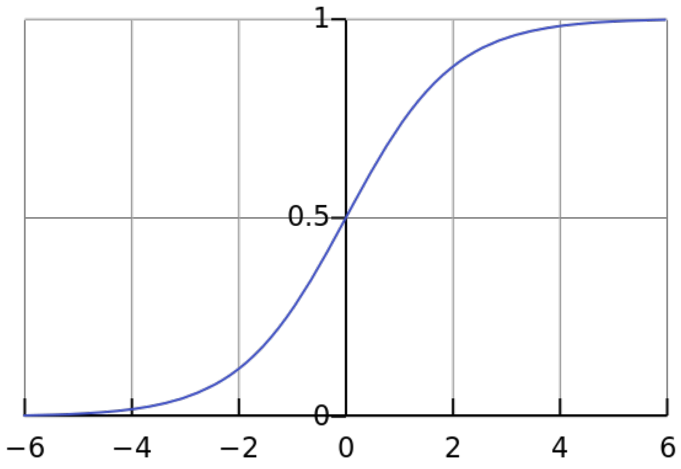
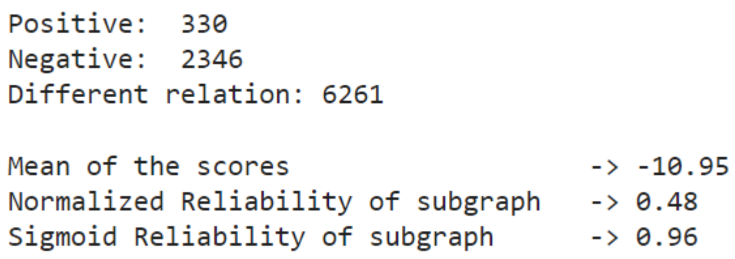


Figure 4.5. Logistic function.

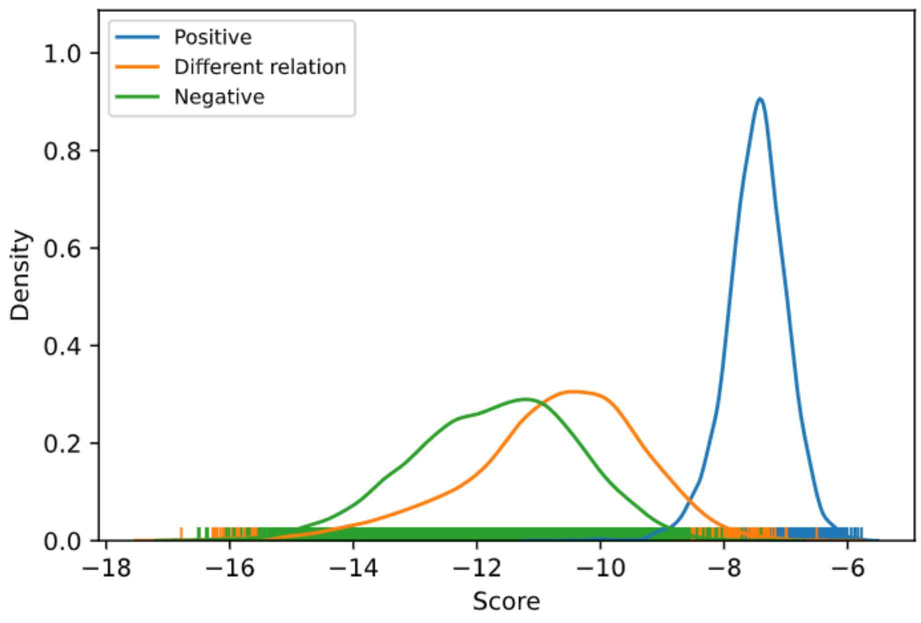
the way the scores are normalized since the general formula remains the same. In addition to this, it is also useful to show the average score of all the triples in the subgraph, to also evaluate the correlation between the results obtained. From the



```
Positive:  330
Negative:  2346
Different relation: 6261

Mean of the scores                     -> -10.95
Normalized Reliability of subgraph     -> 0.48
Sigmoid Reliability of subgraph        -> 0.96
```

Figure 4.6. Evaluation of a subgraph in the UMLS graph.

Figure 4.6 it is easy to understand that there is distance between the two reliability values. This is because, as mentioned before, there is often a danger that the range of score values is completely different from the one desired for the sigmoid. The sigmoid reliability has a value close to 1 since, in the general formula of reliability, the expected scores are all positive, but as can be seen also from the average of the total scores of the subgraph the output values are all negative, therefore the second summation (which includes the negative triples) placing the minus sign in front of the score makes the score positive. Since for any subgraph there will be more negative than positive triples, the second summation affects more than the first, for this reason the values shown are obtained.



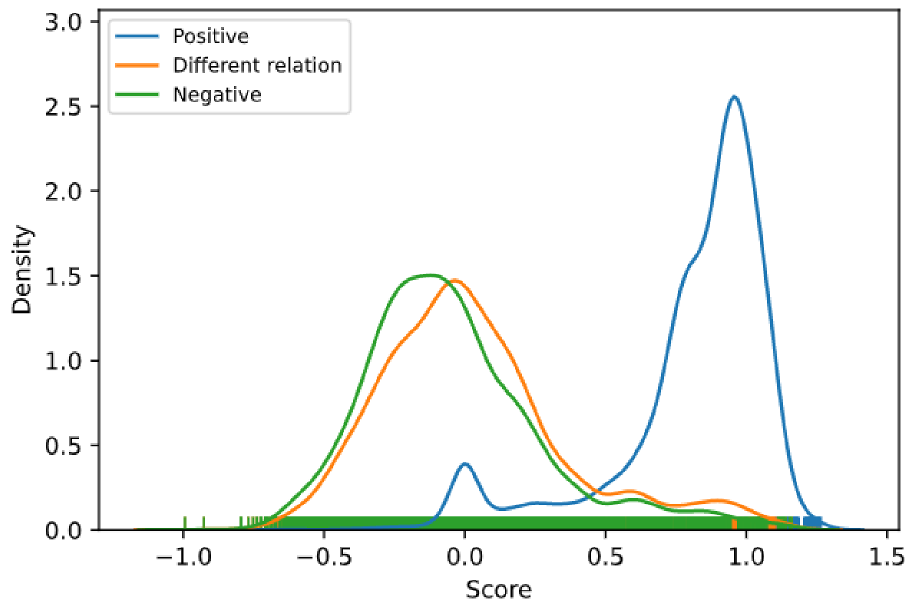
```
** Mean **
Positive: -7.478021504579568
Different relation: -10.748706786628887
Negative: -11.72066083018839

** Standard deviation **
Positive: 0.4969520535767382
Different relation: 1.4220439523630526
Negative: 1.3189671120230386
```

Figure 4.7. Distribution of triples' scores in the UMLS graph with TransE.

As can be seen in Figure 4.7 and Figure 4.8, after plotting the distributions of the score in the various sets belonging using first TransE and then DistMult, the output values that characterize the score of the triples are completely different between the two models, since they have different techniques for representing embeddings and different scoring functions for evaluating triples. This problem can be solved by generalizing a pre-processing calculation on the scores, so before performing the reliability function it is necessary to normalize the values in the same reference range.

One consideration that needs to be highlighted is the placement of the three distributions. As initially assumed, the average of the scores in the different relation set should be between positive and negative scores, since embedding models should consider the reconstruction of this type of triple more likely than negative ones, but less likely (of course) than the positive ones. This fact is confirmed by the plots shown above, and it is a fact to be taken into account also for the next phase of

experiments.
In addition, in the next phase, the selection of triples can also be improved. Going into more detail, as you can see from the figures, there are values in all three sets that are very far from the average distribution. This can lead to untrue results in the general evaluation phase, since these values (which are real outliers) could negatively affect the final result.



```
** Mean **
Positive: 0.8053691607574274
Different relation: 0.027719382437290243
Negative: -0.04181299130177102

** Standard deviation **
Positive: 0.2869263260966495
Different relation: 0.34067449419344653
Negative: 0.3195236923145709
```

Figure 4.8. Distribution of triples' scores in the UMLS graphwith DistMult.

4.3 Phase three

This is the final stage of experiments and tests. After all the considerations made, in this part the idea is directed more on the realization of strategies that can have more impact on realistic scenarios. Then it is considered the idea to look for subgraphs in the KG that are even denser than those taken previously, the choice of triples that can help in the calculation of reliability, and finally the improvement of the LP classifier, that can be done by testing other models in addition to logistic regression, and creating a dataset that is composed of embedding instead of node labels and reports. Eventually, some correlation measurements are selected to verify the correspondence between the classification accuracy and the reliability measurement for each sampled subgraph.
To motivate the results using a more realistic scenario, to test the new version of the project it is necessary to consider larger knowledge graphs. The KGs taken into consideration are **FB15k-237** and **WN18**.
FB15k-237 is a subset of FB15k built by Toutanova and Chen [24], inspired by the observation that FB15k suffers from test leakage, consisting in test data being seen by models at training time. In FB15k this issue is due to the presence of relations that are near-identical or the inverse of one another. In order to assess the severity of this problem, Toutanova and Chen have shown that a simple model based on observable features can easily reach state-of-the-art performance on FB15k. FB15k-237 was built to be a more challenging dataset: the authors first selected facts from FB15k involving the 401 largest relations and removed all equivalent or inverse relations. In order to filter away all trivial triples, they also ensured that none of the entities connected in the training set are also directly linked in the validation and test sets.

Table 4.3. Datasets chosen for the phase three.

Datasets			
Name	**Entities**	**Relations**	**Triples**
FB15k-237	14505	237	310079
WordNet-18	40943	18	151442

WN18, also introduced by the authors of TransE [25], was extracted from WordNet3, a linguistic KG ontology meant to provide a dictionary/thesaurus to support NLP and automatic text analysis. In WordNet entities correspond to synsets (word senses) and relations represent their lexical connections (e.g. "hypernym"). In order to build WN18, the authors used WordNet as a starting point, and then iteratively filtered out entities and relationships with too few mentions. Until now we have only spoken about the type of the chosen graphs, which are all heterogeneous and directed,

but what happens in the case in which the graph is undirected, and therefore the relations are symmetrical? In the analysis of embedding models, for example, it is proved that DistMult works more efficiently with symmetric rather than asymmetric relationships, therefore with undirected KGs rather than directed ones. So it can be possible to consider this by representing the dataset in two different ways, then creating the triples of the knowledge graph keeping their original direction (in the directed case) or making all relationships symmetrical. For this second case the structure of the graph is necessarily modified, this because if you have two triples between a pair of entities, and the two triples characterize the same relationship but have opposite directions, the edges will be merged into one only, describing the bidirectionality of the triple. In this way the number of true facts in the KG will drop, as well as the number of corrupted triples in different relation and negative sets.
Two strategies can be adopted to model the structure of the output graph:

- **Symmetric case**
  Starting from the original graph, since that it will be managed as an undirected graph, triples (h, r, t) and (t, r, h) will be considered identical, because the relation is the same. Therefore, in this specific case, there are two true facts between h and t in the original KG (assuming that there is only the relation r between the pair of entities), but in the induced graph there will be only one positive triple. This means that the total number of triples in the output graph, also considering the corrupted triples, will be halved compared to the observations made so far.

- **Asymmetric case**
  Here we deal with a directed graph, therefore between each pair of entities the relations can be repeated maximum twice, since there is the possibility of representing the same triple in both directions. For this reason, the total number of triples will be double in comparison to the case where the graph is undirected.

As already mentioned in the previous section you need to improve the selection of triples for tasks to be performed. There are outliers that negatively affect the evaluation of the graph, so it is appropriate to remove them both to feed them to the reliability function, both for the link prediction classifier. To remove outliers you can take advantage of the parameters of the distributions, i.e. the mean and the standard deviation.
For the reliability function, if you want to apply a sigmoid on top, firstly it is necessary to normalize scores in the range $[-6, 6]$, i.e. the values eligible by logistic

function, to make the reliability parameter falls between 0 and 1. This can be done with the following formula:

$$z_i = (b-a) \cdot \frac{x_i - min(x)}{max(x) - min(x)} + a$$

Where a and b are are the extremes of the desired normalization range, x_i is the current score, x is the array of scores, and $max(x)$ and $min(x)$ represent the maximum and minimum values among all the scores. For this particular case, since the target range is $[-6, 6]$, therefore $a = 6$ and $b = -6$.
In this way, taking any dataset available, it is always possible to perform the reliability function without any difference in interpretation.
In addition, the possibility of assessing another formula for the reliability function was considered at this stage. The general formula presented in the Chapter 3 provides that the sum of the scores is first performed (in the case of negatives, 1 - score), and then the result is divided by the number of total triples. The other option considered at this time is the partial sum of the triples contained in the three sets, dividing each sum by the cardinality of the reference set, then adding the three results and dividing them by 3 (value representing the number of sets). Thus is defined the formula assumed at this stage:

$$R(K) = \frac{1}{3}\left[\frac{1}{|P|}\sum_{(h,r,t)\in P} f_r(h,t) + \frac{1}{|D|}\sum_{(h,r,t)\in D}(1 - f_r(h,t)) + \frac{1}{|N|}\sum_{(h,r,t)\in N}(1 - f_r(h,t))\right]$$

Where P, D, N denotes the positive, different relation and negative sets respectively, $f_r(h,t)$ is the scoring function of an embedding model and (h,r,t) represents a triple composed of head entity, relation, tail entity.
This is an opportunity to consider for further comparisons, because the fact that the reliability result is not totally satisfactory does not depend only on the triple considered and the scoring function of the chosen model, but also by how the reliability formula itself was built. Remember that the formula was built on the basis of a hypothesis, so it is not said that it is correct but you can still check its validity.
By observing the distributions and the number of triples generated for different relation and negative set, further considerations can be made on the reliability hypothesis. Assuming that the subgraphs selected with the random walk are composed of nodes mostly close together, it means that the cardinality of the different relation set will be always greater than that of the negative set, since there will be more triples corrupted from a pair of neighboring nodes instead of two unrelated entities. Considering both sets in the reliability formula would lead to

results that tend to be at the center of the range [0,1], because the distribution of different relation triples is located approximately in the middle of the score range of each embedding model, and having more weight than other types of triples should affect the final result.
Therefore an attempt is made in which only the purely negative triples are generated (discarding all those belonging to the different relation set), calculating the reliability of the subgraphs only with the triples taken into consideration.
Random walk is also a concept to explore in this part of the project, to understand if you can improve by finding areas in the graph that can be visited in a better way. Until now a random walk similar to a DFS has been built, so starting from an initial node we move further and further looking for a neighbor of the previous node. In the worst case the distance between the first and last nodes considered in the subgraph is $k - 1$, where k is the number of nodes in the subgraph. Getting a distance of this kind is not a favorable information, since the goal is to look for a dense substructure and then taking all nodes close together, trying to locate a community within the graph.
So an alternative random walk can be defined as follows:

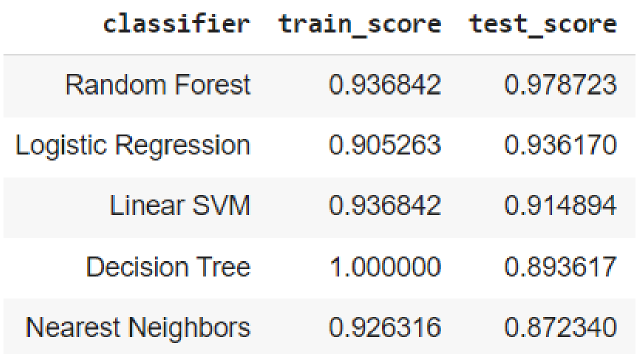
- Pick a random node in the graph
- Choose all the neighbors of the current node that have not yet been visited
- If the list of visited nodes does not yet contain k nodes, select a random node from the list and go back to the previous point
- Otherwise return the induced subgraph of the k nodes

This sampling of the KG can refer to a **breadth-first search** (BFS). In this way the selected subgraphs should present themselves in more compact and dense structures, since the list of all nodes near the current node is taken for each iteration, and this results in a shorter distance between the two nodes farther away in the graph.
Returning to the construction of the classifier, its structure has remained unchanged since the first phase of the project, because as explained in the previous section, the general focus is always aimed at reliability, but since the two values need to be compared to ensure that an area of the graph is reliable, we must try to make the classification model as accurate as possible.
The first change is on the dataset that is passed to the ML classifier. Since embeddings were not extracted from models before, there was no information about the representation of entities and relationships in the embedding space, so triples were used (i.e. entities and relations labels) as feature vectors for the training phase of the model. Now, in addition to evaluating the triples through the scoring function,

embedding models also return the embeddings of entities and relations, which will be represented through n-dimensional arrays. Depending on the parameters chosen for the training of the models of embedding changes the dimensionality of the arrays, considering the current case are generated arrays composed of 50 cells, where each array represents exactly an entity or a relation in the embedding space. So if before each feature vector had size 5, in which the first 3 elements are the labels of the head entity, relation and tail entity, the forth element is the triple score, and the last one represents the class belonging to the triple, now for each feature vector there will be 3 sequential embedding size 50, each for each element of the triple, plus the last column (unchanged) representing the class label. Having said that, the dataset will be of size $n \times 151$, where n is the total number of triples. There is an additional consideration to do before switching to the models selected for this task. To train the classifier we need a balanced dataset, and since there are two classes (i.e. binary classification), the dataset must consist of 50% of positive samples and the other 50% of negative samples. The samples referred to are not the triples of the considered subgraph, but the whole KG, and since $|P| < |N|$, with P and N representing the set of positive and negative triples of the KG, it is necessary to limit the cardinality of N to that of P, i.e. $|N| = |P|$, to make sure that the dataset is composed in equal number of samples belonging to the two classes. Then the classifier will be trained once on the selected triples of the KG, and then will be tested on each subgraph with the triples of the corresponding subgraph.
With regard to the models selected at this phase, it was decided to test more than one of them to verify the difference of evaluation between the models themselves. Five of the most used models for binary classification have been chosen:

- **Support Vector Machines**
- **Logistic Regression**
- **Nearest Neighbors**
- **Decision Trees**
- **Random Forest**

For each model will be described the train score, test score and train time, in addition to the representation of the *ROC curve*, graph that relates the sensitivity and specificity of a diagnostic test based on the variation of the cut-off value.



classifier	train_score	test_score
Random Forest	0.936842	0.978723
Logistic Regression	0.905263	0.936170
Linear SVM	0.936842	0.914894
Decision Tree	1.000000	0.893617
Nearest Neighbors	0.926316	0.872340

Figure 4.9. Accuracy of the main binary classification models on a subgraph of WordNet.

In Figure 4.9 are shown the accuracy values of the binary classification models on a sampled subgraph of 20 nodes, taken from the WN18 dataset. The classifier can still be improved in its structure, working on the hyperparameters of each model, or even developing a preliminary work on the dataset reducing its dimensionality. It is inconceivable to understand if the results are satisfactory, and in order to better understand it we can refer to correlation metrics that assess the correspondence between elements of different sets, in this case between the reliability values and the accuracy values of the classifier in the LP task.
Now there are all the elements to verify the correlation between the values resulting from the two main tasks of the project. Three main correlation measures are considered:

- **Pearson**
  The Pearson correlation coefficient is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations; thus it is essentially a normalised measurement of the covariance, such that the result always has a value between -1 and 1.

- **Spearman**
  Spearman's rank correlation coefficient is a nonparametric measure of rank correlation (statistical dependence between the rankings of two variables). It assesses how well the relationship between two variables can be described using a monotonic function. The Spearman correlation between two variables is equal to the Pearson correlation between the rank values of those two variables; while Pearson's correlation assesses linear relationships, Spearman's correlation assesses monotonic relationships (whether linear or not). If there are no repeated data values, a perfect Spearman correlation of 1 or -1 occurs when each of the variables is a perfect monotone function of the other.

- **Kendall-Tau**
  The Kendall rank correlation coefficient, commonly referred to as Kendall's τ coefficient, is a statistic used to measure the ordinal association between two measured quantities. It is a measure of rank correlation: the similarity of the orderings of the data when ranked by each of the quantities. Intuitively, the Kendall correlation between two variables will be high when observations have a similar (or identical for a correlation of 1) rank between the two variables, and low when observations have a dissimilar rank between the two variables.

Focusing on Pearson's measure, it is possible to study in depth the calculation of the correlation parameter through covariance and standard deviation.
Given two statistical variables X and Y, the Pearson correlation index is defined as their covariance divided by the product of the standard deviations of the two variables:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

where σ_{XY} is the covariance between X and Y, and σ_X, σ_Y are the two standard deviations.
The coefficient always assumes values between -1 and 1:

$$-1 \leq \rho_{XY} \leq 1$$

In practice, there are various "types" of correlation.

- If $\rho_{XY} > 0$, the variables X and Y are said to be *directly related*, or *positively related.*
- If $\rho_{XY} = 0$, the variables X and Y are said to be *unrelated.*
- If $\rho_{XY} < 0$, the variables X and Y are said to be *inversely related*, or *negatively related.*

In addition, direct (and similarly inverse) correlation distinguishes:

- If $0 < |\rho_{XY}| < 0,3$ there is *weak correlation.*
- If $0,3 < |\rho_{XY}| < 0,7$ there is *moderate correlation.*
- If $|\rho_{XY}| > 0,7$ there is *strong correlation.*

To use these measures an important hypothesis must be made, i.e. the samples of a population follow a *normal* (or Gaussian) distribution, therefore even if the correlation measurements give results that at first impact seem to move away from the target, actually could be good because the error was committed in the phase of capture of samples. Another factor that could compromise the significance of the results is the number of samples. The choice fell on the collection of 100 samples, then the localization of 100 subgraphs, both because the number is considered high but also for efficiency reasons. Increasing the number of samples would become a problem if several tests were to be carried out on different KGs. The obtained results of these tests will be shown in the next section.

Chapter 5

Results

This chapter will present the results of experiments carried out at all stages of the project. Starting from the reliability function, two general formulas and different techniques have been proposed for the normalization of scores. This has led to different output values due to data processing with different methods, such as removing outliers from triple sets, or normalizing with a logistics function. In addition, the choice of the dataset or the size of the subgraphs also affected the final result.
Turning to the classifier, it is difficult to make a general assessment of the results obtained. The methods used for the classification of triples were not as convincing as hoped, as not all of them followed the same line of assessment as some of them returned values of accuracy opposite to the other methods. This led to follow only one method, i.e. logistic regression, trying to compare its accuracy value with the reliability one, which is shown with various techniques.
In the following tables will be shown values of the mean of triples scores (SCORE), the min-max normalized reliability (NORM), the sigmoid reliability (SIGM), the second version of the reliability with a sigmoid on top (SIGM_2), and the accuracy of the LP classifier (ACC).

Table 5.1. TransE on directed WN18. Mean values are presented for different subgraph dimension.

Mean values					
#NODES	**SCORE**	**NORM**	**SIGM**	**SIGM_2**	**ACC**
10	-7,09	0,71	0,75	0,78	0,93
20	-7,86	0,71	0,77	0,8	0,92
30	-8,03	0,67	0,72	0,69	0,92

The Table 5.1 shows the values related to the evaluation experiment of the TransE model on the subgraphs of the WN18 dataset. In this case, the KG was designed to be directed. By observing the results, taking larger size subgraphs the values derived from the reliability function decrease. This leads to the deduction that subgraphs composed of too many nodes are not able to represent a reliable area, since increasing the number of nodes increase also the number of negative triples, therefore having in proportion less positive triples, the reliability function returns lower results. Note that in this case the classifier has a very high accuracy even in large subgraphs.

Table 5.2. Pearson correlation.

Correlation					
	SCORE	**NORM**	**SIGM**	**SIGM_2**	**ACC**
SCORE	1	0,99	0,99	0,85	0,85
NORM	0,99	1	0,98	0,86	0,81
SIGM	0,99	0,98	1	0,89	0,82
SIGM_2	0,85	0,86	0,89	1	0,68
ACC	0,85	0,88	0,87	0,68	1

The Table 5.2 shows the correlation between the values shown above. No distinction is shown here for subgraph nodes, as the average of the three values related to the three dimensionality of subgraphs (i.e. 10, 20 and 30 nodes) has been calculated. There is a strong correlation between the average of the triple score and the normalized score, this means that the normalization operation has been performed in the correct way. However, normalization with the sigmoid using the second version of the reliability function does not appear very reliable compared to the others, so this option in the case of evaluation of direct knowledge graphs can be discarded.

Table 5.3. DistMult on undirected FB15k-237. Mean values are presented for different subgraph dimension.

Mean values					
#NODES	**SCORE**	**NORM**	**SIGM**	**SIGM_2**	**ACC**
10	0,21	0,74	0,81	0,84	0,81
20	0,14	0,70	0,77	0,78	0,82
30	0,10	0,70	0,78	0,77	0,84

Table 5.4. Pearson correlation.

Correlation					
	SCORE	**NORM**	**SIGM**	**SIGM_2**	**ACC**
SCORE	1	0,99	0,99	0,88	0,55
NORM	0,99	1	0,96	0,87	0,56
SIGM	0,99	0,96	1	0,94	0,67
SIGM_2	0,88	0,87	0,94	1	0,55
ACC	0,55	0,56	0,67	0,55	1

Since that DistMult works better with undirected graphs, this second test regards the evaluation of the FB15k-237 graph (constructed as undirected) with the DistMult model. The considerations to be made in this case are similar to the previous scenario, but there is an aspect not to be underestimated. This aspect concerns the LP classifier: the results are much lower than the previous case, and on the other hand, increasing of the nodes also increases the accuracy. This trend goes in the opposite direction to the reliability function, as also confirmed by the Table 5.4 where the correlation values of the classifier are drastically lower than the other elements.

Chapter 6

Conclusions

In this thesis have been analyzed different characteristics of knowledge graphs seeing them from another point of view, such as the exploration of different areas depending on the density of the subgraphs, or the type of visit carried out.
The hypothesis of reliability is not to be excluded, as verified also in the results, but steps can be improved that would certainly bring better results. Sampling the graph is definitely one of the things that can be improved. The fact that with increasing the dimensionality of the subgraph considered the performance decreases, means that the visit works well only in the case in which it is considered a low number of nodes, and it is definitely an aspect to consider.
Many works have focused on Link Prediction, using embedding models to evaluate entire knowledge graphs, but analyzing specific areas of the graph is an even more interesting task to analyze. It was also very interesting to know different types of embedding models, and how their accuracy varies depending on the type of graph or sampling method. Even the classifier needs to be analyzed: such high accuracy results lead to the deduction that it can be built well, but it should be noted that even in this case the graph structure can affect the overall performance. Perhaps the possibility of considering further models can be considered.
General work has been partly facilitated by the *Closed World Assumption*, so in the future it is also possible to consider the *Open World Assumption.*

Bibliography

[1] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[2] Frans N Stokman and Pieter H de Vries. Structuring knowledge in a graph. In *Human-computer interaction*, pages 186–206. Springer, 1988.

[3] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[4] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[6] Shivani Choudhary, Tarun Luthra, Ashima Mittal, and Rajat Singh. A survey of knowledge graph embedding and their applications. *arXiv preprint arXiv:2107.07842*, 2021.

[7] Nadathur Satish, Narayanan Sundaram, Md Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. pages 979–990, 2014.

[8] Arif Mahmood, Michael Small, Somaya Ali Al-Maadeed, and Nasir Rajpoot. Using geodesic space density gradients for network community detection. *IEEE Transactions on Knowledge and Data Engineering*, 29(4):921–935, 2016.

[9] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[10] SS Nishana and Subu Surendran. Graph embedding and dimensionality reduction-a survey. *International Journal of Computer Science & Engineering Technology (IJCSET)*, 4(1):29–34, 2013.

[11] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[12] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[14] Tuan MV Le and Hady W Lauw. Probabilistic latent document network embedding. In *2014 IEEE International Conference on Data Mining*, pages 270–279. IEEE, 2014.

[15] Shu Guo, Quan Wang, Bin Wang, Lihong Wang, and Li Guo. Semantically smooth knowledge graph embedding. 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)*, pages 84–94, 2015.

[16] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable graph embedding for asymmetric proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[17] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. Semantic proximity search on heterogeneous graph by proximity embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[18] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. Distance-aware dag embedding for proximity search on heterogeneous graphs. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] 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*, volume 28, 2014.

[20] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

[21] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[22] Rudolph J Rummel. Dimensionality of nations project. Technical report, Hawaii Univ Honolulu Dept Of Political Science, 1968.

[23] Woodrow W Denham. *The detection of patterns in Alyawara nonverbal behavior*. PhD thesis, University of Washington., 1973.

[24] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.

[25] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.