POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master of Science in Computer Engineering

Master Degree Thesis

Deep Learning on Polito Knowledge Graph

Leveraging Deep Learning techniques for the inference of unseen facts inside a newly built Scientific Knowledge Graph

Supervisors
Prof. Antonio Vetrò
Prof. Juan Carlos De Martin

Candidate
Giovanni GARIFO

ACADEMIC YEAR 2018-2019

To Monia
To my Grandfather

Abstract

Summary here, one page

Acknowledgements

Acknowledgements here, half page

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Graphs are used to empower some of the most complex IT services available today, an example among all is the Google search engine [1]. Graphs can be used to represent almost any kind of information, and they are particularly capable of representing the structure of complex systems and describe the relationships between their elements.

Over the last decade, much effort has been put in trying to leverage the power of graphs to represent human knowledge and to build search tools capable of querying and understanding the semantic relations within them. RDF[2] graphs are a particular class of graphs that can be used to build knowledge bases. Ontologies are used to shape such knowledge bases, in order to have a semantically coherent representation of the domain knowledge. Given a domain and an ontology, RDF graphs allows to build a structured representation of the knowledge in such domain.

Modern machine learning techniques can be used to mine latent information from such graphs. One of the main challenges in this field is how to learn meaningful representations of entities and relations that embed the underlying knowledge. Such representations can then be used to evaluate new links within the graph or to classify unseen nodes. Deep learning

---

[1] https://blog.google/products/search/introducing-knowledge-graph-things-not/

[2] The Resource Description Framework will be introduced in section 2.1.2

techniques have proved to be first class citizens when dealing with representation learning tasks, being able to learn latent representations without any prior knowledge other than the graph structure.

1.2   Goal and contribution

Knowledge sharing is one of the main goals of research organizations. In particular, universities are among the most interested in making publicly available their research results. Today most universities have embraced the Open Science movement, making their scientific publications publicly available through web portals. An example is IRIS[3], which stores all the scientific papers produced by the Polytechnic University of Turin, publicly sharing them through Open Access. IRIS allows to explore the published papers by searching for a field of study, matching it with the keywords inserted by the authors of the publications. This implementation has some limitations: being inserted by the authors, there could be more keywords that refers to the same scientific topic. Moreover, the author may have inserted acronyms or misspelled some words. The consequence is that the search engine of IRIS is unable to correctly retrieve all the publications about a specific research topic, because the system cannot match the searched field of study with an unambiguous *semantic entity*, but only with character strings that are not uniquely identified or semantically linked each other, and also prone to lexical errors.

Our goal is to overcome such limitations and enabling new possibilities for exploring and obtaining insights about the scientific community of the Polytechnic University of Turin. A new semantic-empowered search engine can be one of the possible solutions to obtain this results, allowing for coherent and precise results to be retrieved. At the foundations of this new semantic search engine there must be a data structure capable of representing semantic relations and concepts. Once such knowledge base of the scholarly data is obtained, it can be enhanced and completed by automatically extracting latent information through the use of advanced machine learning algorithms.

In the next chapters we are going to present a newly built structured and semantically coherent representation of the scholarly data produced by the

---

[3]https://iris.polito.it/

Polytechnic University of Turin, and how implicit facts can be automatically extracted from such knowledge repository by leveraging knowledge base completion techniques, implemented by means of an advanced deep learning algorithm.

1.3 Thesis structure

1.3.1 Chapter 2

1.3.2 Chapter 3

1.3.3 Chapter 4

Chapter 2

Background

2.1 Semantic Web

2.1.1 From a Web of Contents to a Web of Data

The World Wide Web has been developed as a tool to easily access documents and to navigate through them by following hyperlinks. This simple description already resembles the structure of a graph: we can think of documents as nodes and hyperlinks as edges. The unstoppable growth of the *Web graph* led to the raise of new tools to explore such complexity. Search engines have been developed to easily navigate such a giant graph. First approaches were based on analytics evaluations, such as the number of times a document has been linked, as in the case of the PageRank [2] algorithm developed by Google.

The Web rapidly became one of the most innovative technology ever built, allowing to retrive information quickly and easily as never before. The next evolutionary step has been to think about a Web not only exploitable by human beings but also by machines. In order to build such a comprehensive system, where information can be not only machine-readable, but machine-understandable, the World Wide Web had to move from a web of content, to a web of data.

The World Wide Web Consortium (W3C) introduced the Semantic Web as an extention to the prior standard of the WWW. Its primary goal has been to define a framework to describe and query semantic information

contained in the documents available on the Web, so as to allow machines to understand the semantic information contained in web pages. In the vision of Tim Berners-Lee, the father of WWW, this would bring to the transition from a World Wide Web to a Giant Global Graph [1], where a web page contains metadata that provides to a machine the needed information to understand the concepts and meanings expressed in it.

2.1.2   The Semantic Web building blocks

The three key components of the Semantic Web standard are:

1. OWL: the Web Ontology Language [3]
2. RDF: the Resource Description Framework [4]
3. SPARQL: The SPARQL Protocol and RDF Query Language

OWL is a language used to define ontologies. In this context, an ontology is defined as a collection of concepts, relations and constraints between these concepts that describes an area of interest or a domain. OWL allows to classify things in terms of their meaning by describing their belonging to classes and subclasses defined by the ontology: if a thing is defined as member of a class, this means that it shares the same semantic meaning as all the other members of such class. The result of such classification is a taxonomy that defines a hierarchy of how things are semantically inter-related in the domain under analysis. The instances of OWL classes are called individuals, and can be related with other individuals or classes by means of properties. Each individual can be characterized with additional information using literals, that represent data values like strings, dates or integers.

The Resource Description Framework defines a standard model for the description, modelling and interchange of resources on the Web.

The first component of the framework is the *RDF Model and Syntax*, which defines a data model that describes how the RDF resources should be represented. The basic model consist of only three object types: resource, property, and statement. A resource is uniquely identified by an

---

[1]https://web.archive.org/web/20160713021037/http://dig.csail.mit.edu/breadcrumbs/node/215

Uniform Resource Identifier (URI). A property can be both a resource attribute or a relation between resources. A statement describes a resource property, and is defined as a triple between a subject (the resource), a predicate (the property) and an object (a literal or another resource).

The second component of the framework is the *RDF Schema* (RDFS), that defines a basic vocabulary for describing RDF resources and the relationships between them. Many of the vocabularies and ontologies available today are built on top of RDFS, such as the Friend of a Friend (FOAF) ontology [5], for describing social networks, or the one maintained by the Dublin Core Metadata Initiative [6], that defines common terms and relations used in the definition of metadata for digital resources.
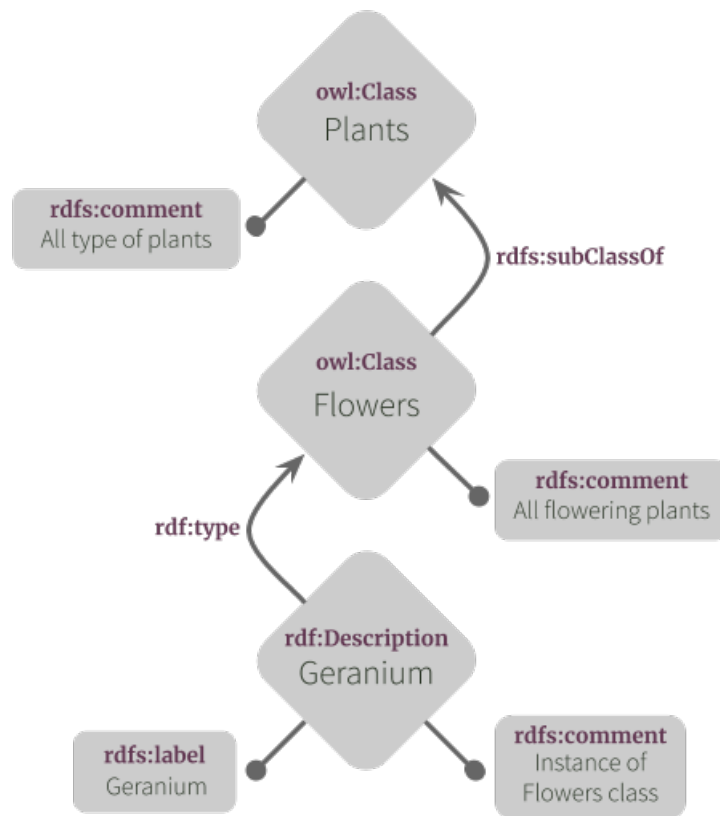

Figure 2.1.   An example of ontology defined using OWL and RDF Schema.

SPARQL is a query language for triplestores, a class of Database Management Systems (DBMS) specialized in storing RDF databases. Such DBMS often expose endpoints that can be used to query the database

17

and obtain results. Given the complexity of the stored data, the query language has been designed to be as simple as possible, for instance by allowing the use of variables, whose definition is preceded by a question mark.

The syntax of SPARQL is heavily derived from SQL, with some minor adaptations to be more suited for querying graph data. The following is an example of query which selects all the labels (human-readable description of a resource) of all the entities that match the given resource type.

```
PREXIF plants:<http://example.org/plants/>

SELECT ?name
WHERE {
    ?subject rdf:type plants:flowers .
    ?subject rdfs:label ?name .
}
```

2.1.3   Knowledge Bases as knowledge repositories

Even though the raise of the Semantic Web has suffered a slowdown in its growth due to the complexity of its vision, many new projects were born from its enabling technologies. Efforts have been put by profit and non-profit organizations in trying to build complex knowledge repositories starting from the knowledge already available in the Web. An example among all is the DBpedia[2] project, which developed a structured knowledge base from the semi-structured data available on Wikipedia. Another example is the *Google Knowledge Graph*[3], which is used to enhance the Google search engine and virtual assistant capabilities, allowing to retrieve punctual information about everything that has been classified in its ontology and described in its knowledge base, or the *Open Academic Graph*[4], a Scientific Knowledge Graph that collects more then three hundred million academic papers. A comparison between some of the biggest knowledge graphs developed to this date is available in Table 2.1.

---

[2]https://wiki.dbpedia.org/

[3]https://blog.google/products/search/introducing-knowledge-graph-things-not/

[4]https://www.openacademic.ai/oag/

Table 2.1. Comparison of some of the biggest industry-scale knowledge graphs developed to this date. Table adapted from https://cacm.acm.org/magazines/2019/8/238342

	Data model	**Graph size**	**Development stage**
Microsoft	Entities, relations and attributes defined in an ontology.	2 billion entities and 55 billion facts.	Actively used in products.
Google	Strongly typed entities, relations with domain and range inference.	1 billion entities, 70 billion assertions.	Actively used in products.
Facebook	All of the attributes and relations are structured and strongly typed.	50 million primary entities, 500 million assertions.	Actively used in products.
eBay	Entities and relation, well-structured and strongly typed.	100 million products, more than 1 billion triples.	Early stages of development.
IBM	Entities and relations with evidence information associated with them.	Various sizes. Proven on scales documents >100 million, relationships >5 billion, entities >100 million.	Actively used in products and by clients.

From an implementation perspective, knowledge bases can be created to describe a specific domain by defining an ontology and a vocabulary for such domain using OWL and RDF Schema, and then by describing the concepts of such domain using the RDF Model and Syntax. The RDF document obtained can then be stored in a triplestore and queried using SPARQL. The main effort in building knowledge bases is to have a correct understanding and prior knowledge of the domain of interest, to avoid the risk of mischaracterizing and misrepresenting concepts.

If all the requirements and cautions are met, a well formed knowledge base may prove to be a critical resource for an organization. It permits to build new services upon it, and also to improve the existing knowledge inside the organization by performing reasoning upon the available knowledge, in order to derive implicit facts strting from the existing entities and
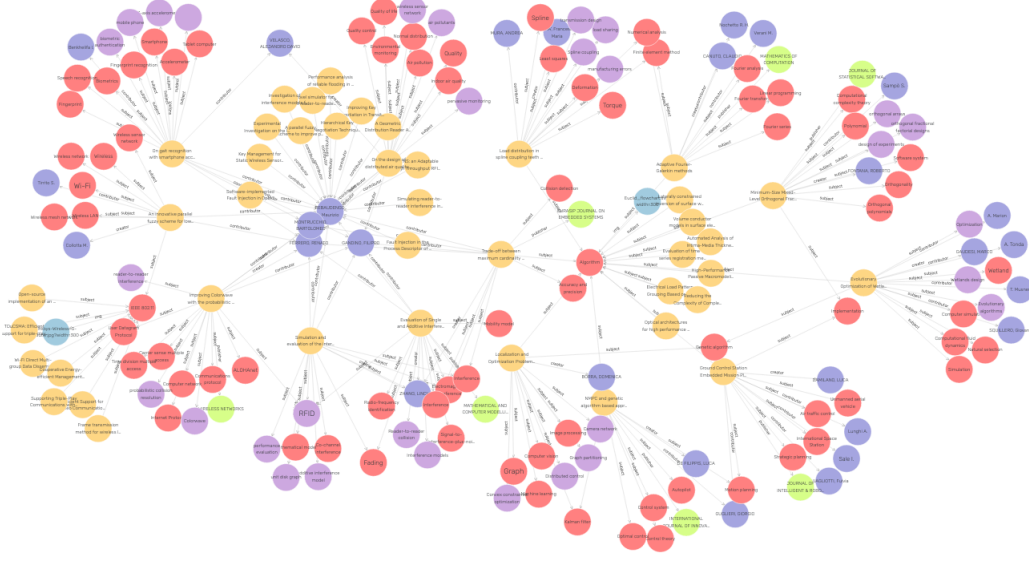
Figure 2.2. An extract of the Polito Knowledge Graph whose details will be described in section 5.1.

relationships. Another field of applications is the development of Expert Systems[5], AI software that emulates the behavior of a human decision-making process by navigating the knowledge base and taking decisions like in a rule-based system.

Today's knowledge bases are commonly composed by tens of thousands nodes and by hundreds of thousands of edges. Considering such dimensions, storing and querying giant graphs requires the adoption of specialized DBMS that are capable of efficiently store and query the RDF input representation. Moreover, performing analysis and gathering statistics from such giant graphs requires the adoption of highly efficient algorithms in order to retrieve the desired output in an acceptable time.

The availability of such a complex and informative data structure leads to the opening of interesting scenarios, especially when thinking about the latent information that can be extracted from it. In fact, a knowledge base is a structured representation of the human knowledge in a specific field, thus its comprehensiveness is restricted by the human understanding.

---

[5]https://en.wikipedia.org/wiki/Expert_system

2.2 Learning on Graphs

2.2.1 Representation learning

Machine learning (ML) algorithms are used to learn models from the available data, with the final goal to obtain a set of parameters that are fine-tuned to identify seen characteristics in the data used for training. The obtained models can be used to recognize unseen inputs by leveraging the knowledge embedded in such parameters. ML algorithms require the input data to be available in a machine-understandable vector representation. An important task in the ML field is the learning of such representations, task known as representation learning.

Natural Language Processing (NLP) is one of the research branches that in the past years has made a great use of machine learning algorithms both for language recognition and for embedding words *meaning* into words *vectors*. One of the most successful algorithms when dealing with representation learning of words is Word2Vec [7], where the model obtained is trained to learn a vector representation for each word in a vocabulary. In Word2Vec, the concept of meaning of a word is related to the context in which such word is frequently used, so two words are recognized as similar if they're used in similar contexts, thus in the vector space of the learnt representations words that have similar meaning have higher cosine similarity[6] with respect to dissimilar ones. For instance, the cosine similarity between the word vectors of "Man" and "King" is roughly the same as the one between the words "Woman" and "Queen", since such words are used in similar contexts. This has open up new scenarios for language recognition and processing, since it allowed to perform vector operations on such words which brought interesting results, as can be seen in Figure 2.3.

This idea of words characterized by the context in which they're used can be generalized and applied to other fields of research, such as the field of representation learning on graphs.

Graphs are composed by nodes and edges, and are used to describe

---

[6] Cosine similarity is a heuristic method to measure the similarity between two vectors by computing the cosine of the angle between them: $similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$
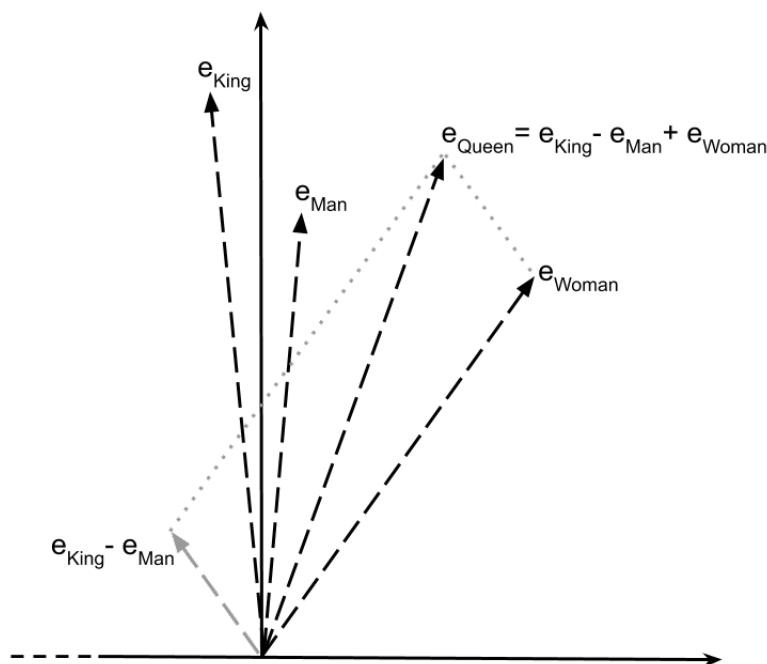
Figure 2.3. Word vectors allows to perform vector operations, the results obtained reflect the fact that Word2Vec is capable of embed the meaning of such words.

complex systems, such as social networks or the interactions in a molecular biology system. To apply machine learning algorithms to such data structures vector representations of nodes and edges are needed in order to be able to learn from the available data and predict new facts. Such vector representations are often referred to as *embeddings* because they should embed the characteristics of the graph nodes, so that similar nodes have similar embeddings. In example, in a scholarly knowledge base publications with same authors and similar subjects should have similar embeddings.

Early approaches required these representations to be learned from feature vectors that where handcrafted, task that required not only a relevant amount of effort, but also a deep understanding of the domain of interest. This has long been one of the main obstacles when dealing with representation learning tasks, since who has knowledge of the domain and who has to engineer the features were unlikely the same individual.

22

2.2.2 Deep Learning on graphs

In the latest years a big shift towards deep architectures has been made in machine learning, mainly thanks to the development of highly parallelized architectures that are able to efficiently compute at the hardware level vector and matrix multiplications, operations that are at the basis of any machine learning task. Deep Learning (DL) algorithms are able to extract relevant features from raw data by applying simple mathematical operations, such as convolution, to the input data. An example of one of the most successful applications of DL is in image recognition, where matrix representations of images are convolved with self-trained filters that are able to extract the relevant features needed to recognize patterns present in the input images.

Deep learning techniques have proven to perform well also in the field of representation learning for graph data. As can be seen in figure 2.4, a digital image is composed by pixels which can be thought of as nodes in a graph, where each pixel is connected by an edge to its immediate neighbors. This suggests that the techniques used when dealing with images can be adapted, with some major changes, to the field of representation learning on graphs, but also in other fields of research, such as learning on manifolds.
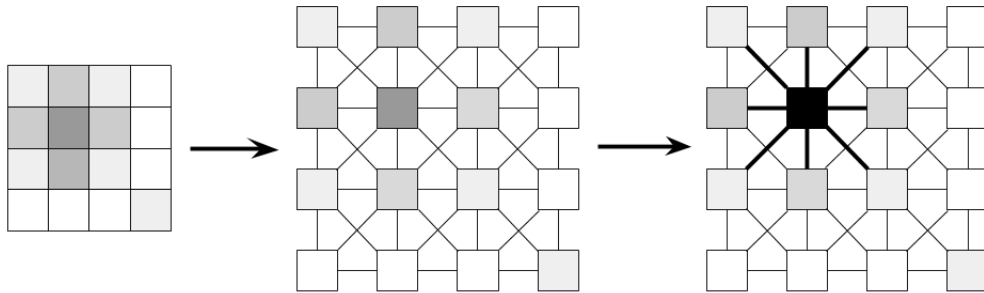
Figure 2.4.   A digital image can be thought of as a graph.

One of the issues when working with graph data is that commonly graphs are built to describe complex systems, such as the knowledge of a domain or field for knowledge graphs, and thus are composed of a fairly high amount of nodes and edges. The matrices used to store the graph structure can thus explode in dimensionality, becoming impractical as input data. Moreover, graphs are not regular structures with a given

shape and size, such a matrix of pixels for images, but they live in an irregular domain which led to highly irregular structures. The first issue can be solved by randomly sampling the graph at each training epoch, the immediate drawback being that more than one epoch is required to train over all graph nodes. The second issue can instead be solved by adapting known algorithms to work on irregular domains. One of the possible approaches, which has proven to work well, is the one based on convolutions.

Graph Convolutional Networks (GCNs) [8] are a class of semi-supervised deep learning algorithms for graphs which are based on the same convolution and backpropagation operations as the well known Convolutional Neural Networks [9] (CNNs) used for feature learning on images. The main difference between CNNs and GCNs is in how the convolution is performed, instead the backpropagation phase is the same as the one used to update the parameters of CNNs, with a task-specific loss function. In a CNN the input matrix of each network layer, which is the pixel matrix of the input image for the first layer, is convolved with a convolutional filter, whose parameters are then updated during the backpropagation phase.

GCNs works similarly by convolving at the l-th layer of the network the feature vector of each node with the feature vectors of its l-nearest neighbors by means of a convolutional filter. This operation is done by applying the following transformation:

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}) \tag{2.1}$$

Where $H^{(l)} \in \mathbb{R}^{N \times d^{(l)}}$ is the nodes hidden representation matrix, which is the output of the previous layer, with N being the number of nodes in the graph and $d^{(l)}$ being the dimensionality of the current layer. For the first layer, H^0 is equal to the input feature matrix X, where each row can be initialized with the respective node feature or with a one-hot encoded vector. For the last layer, $H^{(l+1)}$ is the embeddings matrix. $\tilde{A}$ is an adjacency matrix of the graph that includes self loops. $\tilde{D}$ is the node degree matrix of $\tilde{A}$ and is used to normalize it. $W^l \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$ is the convolutional filter that is shared among all nodes and is unique for each layer, just like the convolutional filter of a CNN layer. The shape of this filter will directly impact the dimensionality of the embeddings obtained, in fact H^{l+1} is shaped as a $N \times d^{(l+1)}$ matrix. Finally, σ is a non linear activation function, for example $ReLU$.

Looking at the update rule of a single node embedding will make more clear how the convolution is actually performed. The forward rule to update the embedding of a single node at the *l*-th layer of the network is the following:

$$h_i^{(l+1)} = \sigma(\sum_{j \in \eta_i} \frac{1}{c_{i,j}} h_j^{(l)} W^{(l)}) \tag{2.2}$$

Where η_i is the set of neighbors of node i, which contains the node itself because of the added self loop, and c_{ij} is a normalization constant obtained from the multiplication between the adjancency and degree matrices. The updated feature vector $h_i^{(l+1)}$ of the node i is obtained by performing the following operations:

1. The feature vectors $h_j^{(l)}$ of its neighbors are transformed by the matrix multiplication with the layer filter $W^{(l)}$.

2. The resulting $1 \times d^{(l+1)}$ shaped features vectors are multiplied with the respective normalization constants and summed together.

3. The non-linear function is applied to the result of the summation, obtaining the updated feature vector of the node i.

As a consequence of applying such transformation to all nodes, at the *k*-th layer a node is represented by its transformed feature vector, which embeds the structural information within the node's *k*-hop neighborhood. So the amount of layers of the network is an hyperparameter that controls how much information from furthest nodes has to be collected in each node embedding. This represents a fundamental architectural difference between CNNs and GCNs: while the former are commonly built by stacking a lot of layers, the latter relies on architectures that are more wider, due to the dimensionality of the graphs involved, and that consist of a fairly low amount of layers, in order to characterize the nodes only by their immediate surroundings.

As a result of the propagation step each node embedding will be characterized by its context, just like it happens in Word2Vec, but in a non-Euclidean domain. So for example, in a social graph where each person is characterized by its friends, interests, places visited and so on, two people will have similar embeddings if they are both linked to the nodes that represent such characteristics.
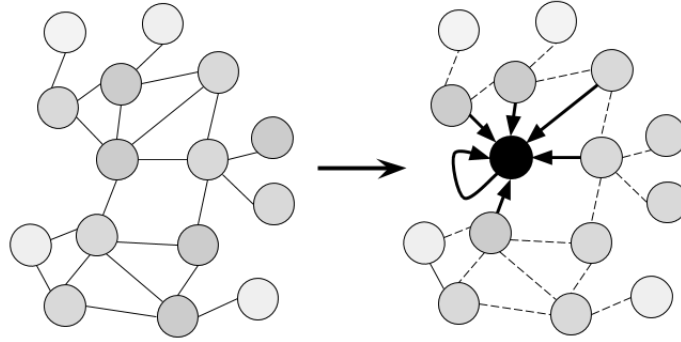
Figure 2.5. First layer of a GCN updating a node feature vector by embedding the features of adjacent nodes.

The node embeddings obtained by applying a GCN or one of its variants can then be used to perform some learning task on the graph, two examples are the classification of unseen nodes or the link prediction of non-existent edges. The latter is one of the most interesting tasks, since it allows to complete the information contained in the graph by predicting unseen facts.

2.2.3   Link prediction on knowledge graphs

Predicting new facts is one of the most common task in the field of knowledge graph completion. The goal of such task is to predict new, unseen triples that correspond to missing facts in the knowledge base, and that can be later added to the graph. Deep learning techniques for link prediction are based on the following two main steps:

1. Train an *encoder* model that is able to embed the node features and produce meaningful embeddings.

2. Apply a factorization model that act as a *decoder*, which is used to score the unseen triples under evaluation.

Deep learning techniques such as GCN can be exploited to obtain meaningful node embeddings, but fall short when dealing with graphs where nodes are connected by different relations (multi-relational graphs). In fact, if we use a single layer GCN to obtain the embeddings of two nodes that share the same neighborhood, but are connected to the neighbors via different relations, we'll obtain almost the same embeddings, even if

it's clear that they do not share the same characteristics. For example in a producer-consumer framework one node could be the producer while the other the consumer, having both a common neighborhood of nodes which are produced by the former and consumed by the latter. If a GCN is used, the embeddings obtained would be very similar, even if the two nodes clearly don't have the same role and don't belong to the same class.

To overcome this limitation, changes to the basic GCN architecture have been proposed, so to obtain models that works well when applied to multi-relational graphs.

The Relational Graph Convolutional Network [1] (R-GCN) is an extension of GCNs which is focused on modeling multi-relational graphs composed by labeled and directed edges, and thus is particularly capable of embedding both nodes and relations of a knowledge graph. R-GCN can be used for both link prediction and node classification tasks.

At an high level the R-GCN architecture can be seen as a special case of the *message passing framework* [10], which groups together under a common scheme most of the existing neural models for graph data. The framework defines two major phases: a per-edge message computation and a per-node message reduction. In the first phase a function or linear transformation is applied to each edge to obtain an edge-specific message. Then, in the reduce phase, the embedding of each node is updated by aggregating together all the messages of the incoming edges.

This two phases can be grouped together through by the following equation:

$$h_i^{(l+1)} = \sigma \left(\sum_{m \in \mathcal{M}_i} g_m(h_i^{(l)}, h_j^{(l)}) \right) \tag{2.3}$$

Where $\mathcal{M}_i$ is the set of incoming messages for node i and g_m is a message specific transformation.

The idea behind R-GCN is to have different set of parameters for different relations. At each step inside the network, the feature vector of a node is updated by convolving its first neighbors features with a convolutional filter that is different based on the kind of relation that connects the nodes. The forward rule to update the embedding of a node at the l-th layer is the following:

27

$$h_i^{(l+1)} = \sigma \left(W_0^{(l)} h_i^{(l)} + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} \right) \tag{2.4}$$

Where $h_i^{(l)}$ is the embedding (or, for the first layer, the input feature vector) of the node i, $W_0^{(l)}$ is the learnt kernel for the self loop relation, $\mathcal{N}_i^r$ is the set of indices of the neighbors of node i under the relation $r \in \mathcal{R}$, with $\mathcal{R}$ being the set of all the relations present in the graph. $W_r^{(l)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}}$ is the learnt filter for the relation r. As for the GCN architecture, σ is a non linear activation function and $c_{i,j}$ is a normalization constant, commonly initialized to $|\mathcal{N}_i^r|$.

From a message passing framework perspective, the message function (per-edge transformation) is equal to the linear transformation $W_r h_j$, and the reduce function (per-node transformation) is just the sum of all the messages computed for the edges connected to each node.

As can be seen the update rule looks similar to the one for GCNs (2.2), with the major difference that in the case of a R-GCN the filters used to convolve the feature vectors of neighboring nodes are relation specific, so the number of filters at each layer will be equal to the number of relations inside the graph. As a consequence, the kind of relation that connect the node to its neighbors has an important role in determining the transformed node embedding.

Some form of regularization is required in order to avoid overfitting on rare relations, thus to obtain a more generalized model, and also to avoid the rapid growth in the number of parameters of the network for highly multi-relational graphs. One of the solutions proposed by the original paper [1] is to decompose each relation filter W_r using basis decomposition:

$$W_r^{(l)} = \sum_{b=1}^{B} a_{r,b}^{(l)} V_b^{(l)} \tag{2.5}$$

This allows to store only the relation-specific coefficients and the basis, which will be shared by all the relations.

The model obtained by training a R-GCN can then be used to build an encoder that given as input a graph, gives as output the embeddings of all nodes and the relations parameters. Then a factorization method can be used to evaluate unseen facts inside the graph, exploiting the resulting
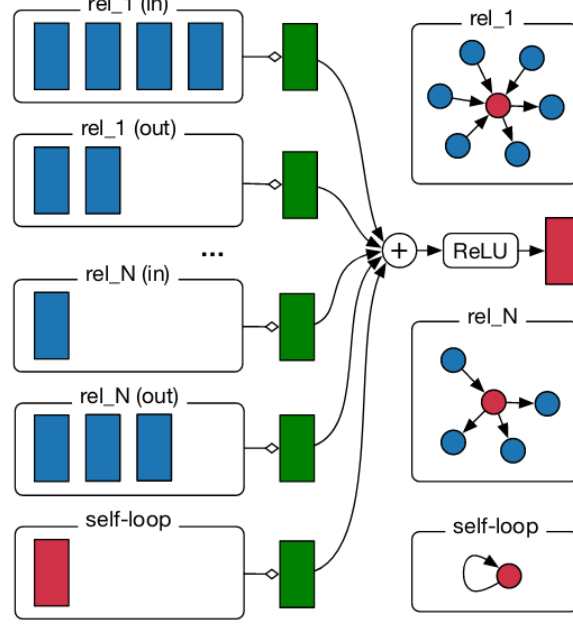
Figure 2.6.   R-GCN encoder model. Image taken from [1].

embeddings. Such methods are used as scoring functions in order to obtain, starting from the entities embeddings, a real value that can be used to score the unseen triples under evaluation.

DistMult [11] is one of the most common and simple factorization methods used to score unseen triples. Given a triple (s, r, o), where s is the source node, o is the destination node, and r is the relation of the edge that links the source to the destination, DistMult allows to compute an associated real valued score as follows:

$$score_{(s,r,o)} = f(s, r, o) = e_s^T R_r e_o \qquad (2.6)$$

Where e_s and e_o are the embeddings of the source and the destination node, obtained by means of an encoder model like R-GCN, and $R_r \in \mathbb{R}^{d \times d}$ is obtained by transforming the embedding of the relation r into a diagonal matrix. The score obtained by applying such function can then be used to evaluate wether the triple (s, r, o) is a good candidate to be added to the graph: an high score has to be interpreted as a high confidence of the model in the fact that the triple should belong to the knowledge graph.

Chapter 3

Approach and methodology

This chapter introduces the architecture developed to build, enhance and visualize the Polito Knowledge Graph (PKG), an academic RDF graph built to organize in a structured and semantically coherent way the publications produced by the researchers of the Polytechnic University of Turin. The graph also includes publication-related entities, such as authors, journals, and field of study.

The architecture, which is showed in Figure 3.1, is structured as a pipeline composed of three main building blocks:

1. The *graph builder*, which creates a first version of the RDF graph.

2. The *graph enhancer*, which implements ML techniques to predict unseen facts. Such facts can then be added to the graph to improve its completeness.

3. The *viewer component*, a web application that allows to query and visualize the graph data.

At an higher level the components of the architecture can be divided in producers and consumers, where the former produces the graph data, and the latter consumes the data produced. The *builder* act as producer taking as input the IRIS data and producing a set of RDF statements that together composes the Polito Knowledge Graph. The *enhancer* act as both a producer and a consumer, given that it takes as input the PKG and use it to predict unseen facts that can be later added to it as RDF

statements. The *viewer* consumes the graph by storing it a triplestore and exposing a public web interface for querying and visualizing the data.
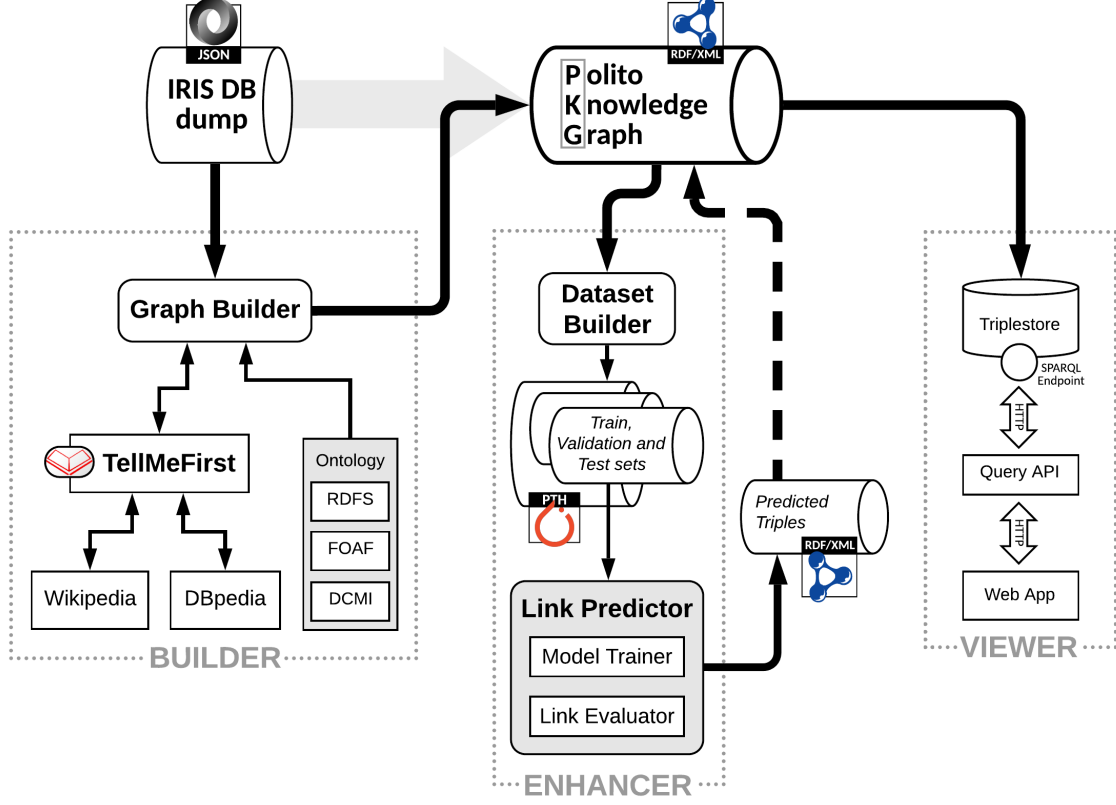
Figure 3.1.   Pipelined software architecture developed to build, enhance and visualize the Polito Knowledge Graph.

3.1   Builder module

The graph builder input is a dump of the IRIS[1] database, which is the platform used by the Polytechnic University of Turin to store and share all the publications produced by its researchers. The dump is a JSON file that contains all the information available for every scientific paper published in a period of five years that goes from 2013 to 2017.

---

[1]https://iris.polito.it/

The goal of the builder is to translate all the information contained in the dump in a set of semantically coherent RDF triples. To do so, an ontology that describes the domain of the IRIS scholarly data has been defined.

The builder uses as reference the defined ontology to analyze each record in the JSON dump, and builds facts as RDF triples by matching the information contained in the record with the concepts defined by the ontology. For example, given that a publication may have more then one contributor, the ontology that we defined differentiates the principal author from the contributors by using two different statements to tie the corresponding entities to the publication.

One of the fields of the publication record is the abstract. We wanted to tie each publication to its main topics, to do so we used the abstract as input text for TellMeFirst [12], a tool for the automatic extraction of semantic keywords from texts. This keywords, to which from now on we'll refer to as *topics*, are retrieved from the DBpedia taxonomy and are uniquely identified by their corresponding URI. Exploiting TellMeFirst we are able to automatically tie each publication to its relevant topics, by adding the corresponding RDF statements to the graph. This has as consequence that the topics are added to the graph, and being uniquely identified, all the publications that share a topic are bounded to the same topic entity.

The result of this process is a semantically coherent description of all the publications linked with their authors, contributors and relevant topics. The set of all this RDF statements composes the Polito Knowledge Graph.

3.2   Enhancer Module

Once the RDF graph has been built, it could be used as input for the Enhancer Module, whose main goal is to leverage ML techniques to predict new triples that could be good candidates to be included in the graph.

The enhancer architecture is splitted in three main components:

1. The Dataset Builder.

2. The Model Trainer.

3. The Link Evaluator.

The first component is in charge of translating the RDF graph into a usable dataset for the model trainer, given that RDF statements cannot be used as input data for the link prediction algorithm. The dataset will then be splitted into three disjoint[2] sets: one for training, one for validation and one for testing. The model trainer uses this three sets to train a link predictor. The training set is used to train the model at each epoch, while the validation set is used to evaluate which model parameters to keep, identifying the best epoch. The test set is used to evaluate the accuracy of the model loaded with the best epoch parameters upon unseen triples.

Once the model has been trained, it can be used to predict unseen facts. The link evaluator loads the best model found during the training phase and uses it to evaluate unseen facts. The evaluator creates an set of unseen RDF triples and produces a score for each of them. Only the triples that obtain an high score (and so an high probability to be true facts) and are correct in domain and range with respect to the ontology are kept. The predicted triples are then saved as RDF statements and could be added to the RDF graph, obtaining an enhanced knowledge graph completed with latent information automatically extracted.

3.3   Viewer Module

The last module of the architecture is the Viewer, which is composed of a triplestore, an API layer, and a front end web interface.

The triplestore is a specialized DBMS for storing and retrieving RDF statements. It stores an online copy of the Polito Knowledge Graph and expose a SPARQL endpoint that allows to query the graph itself.

The API layer exposes a REST API that allows to retrieve the information contained in the graph without the need of SPARQL queries. It accept HTTP requests with URL-encoded parameters and proceeds to query the SPARQL endpoint of the triplestore by matching the parameters upon some predefined queries. Then it translates the response obtained from the triplestore in a JSON file, which is sended back to the requesting client, which is typically the front end.

The front end is a responsive web application that act as the entry point for the user, mimicking the functionalities of a modern search engine. It

---

[2]Two sets are said to be disjoint if they have no element in common.

allows to query and visualize the data contained in the graph by performing HTTP requests to the API layer and displaying the results received in a structured and comprehensible way.

Chapter 4

Related work

Currently there is a growing interest in scientific knowledge graphs, both by academic institutions and by private organizations. An example is the Open Academic Graph[1] (OAC), an academic knowledge graph built by unifying two of the largest scientific KG available, the Microsoft Academic Graph[2] and AMiner[3]. It has been publicly released to allow the study of citation networks, papers content and more. The first version of the graph, released in 2017, has been built by merging together the aforementioned graphs and by linking the matching publications, obtaining a graph that is composed by more then three hundred million publications. In the first version the only type of entity in the graph was the publication: authors, journals, and all the other publications information were added as attributes, and not as entities.

In January 2019 the second version of the OAC has been released, adding even more publications to the graph. However, the biggest change of this new version is the addition of authors and venues as graph entities, instead of being simple publications attributes.

However, the OAC does not contain the publications topics as graph entities, but as author keywords, thus being prone to the same limitations of IRIS: the keywords are the ones chosen by the authors and are not referencing to semantic concepts, being simple character strings.

---

[1]https://www.openacademic.ai/oag/

[2]https://academic.microsoft.com/

[3]https://www.aminer.cn/

Regarding the development of tools similar to the Polito Knowledge Graph by other academic institutions, an example of employment of knowledge graphs and natural language processing techniques is Wiser [13], an expertise search tool developed by the University of Pisa and publicly released at the beginning of 2019. The KG of Wiser is composed of approximately 1'500 authors, 65'000 publications and 35'000 topics. This numbers will be compared to the one of the Polito Knowledge Graph in the following chapter. The main function of Wiser is to allow the search of expertise in a given research field. The system has proven to be particularly effective, representing a strategic tool and being actively used by the university technological transfer office.

As saw in the previous chapter, one of the main components of the PKG pipeline is TellMeFirst, tool used to automatically extract the topics of interest from the publications abstracts. By automatically extracting the topics, TMF allows to add them as entities in the Polito Knowledge Graph, so that each publication is directly linked to its main topics, and each topic is linked to all the publications of which it is a subject (reverse relation).

Other tools that are able to extract the subjects of a publication exists, an example is the CSO Classifier [14]. This tool is able to automatically classify a research paper according to the Computer Science Ontology[4] (CSO), an automatically generated ontology of research topics in computer science. The fact that the CSO Classifier relies on a predefined ontology has some disadvantages with respect to TellMeFirst, the biggest being the fact that the Computer Science Ontology is restricted to the computer science field only, while TMF, using DBpedia as its source of knowledge, is able to extract topics (and so classify a research paper) regarding every field of research. However, the approach of CSO Classifier has also its advantages: being the ontology more restricted, the classification could be more accurate, and the structure of the ontology itself may be tailorized for such classification task.

Regarding the prediction of unseen links inside the graph, to the best of our knowledge we are the first to employ a Relational GCN for the completion task of a scholarly knowledge graph.

---

[4]https://cso.kmi.open.ac.uk/home

Chapter 5

Development and implementation

In this chapter we will describe how we developed and implemented the architecture behind the Polito Knowledge Graph, particularly focusing on the components in charge of the graph creation and enhancement, whose architecture has been introduced in the previous chapter.

We will firstly introduce a detailed description of the input data used to create the graph. We will also describe the ontologies used to shape the knowledge of the scholarly domain and the resulting graph schema, obtained by shaping the available IRIS data using such ontologies. Then we will discuss how we implemented the builder module, which is responsible for the actual creation of the Polito Knowledge Graph starting from the metadata made available by IRIS.

In the second part of the chapter we will focus on the implementation of the link predictor module, which is used to predict unseen facts. We will describe how starting from the Polito Knowledge Graph we created a usable dataset for the training of a machine learning model, how we trained such model and which hyperparameters we used. We will also discuss the metrics used to evaluate the model during the training phase, and how we employed the trained model to obtain predictions about missing links inside the graph.

5.1 Building the Polito Knowledge Graph

As already mentioned when introducing the pipeline, the input data used to build the PKG is a dump of the IRIS database which contains the scientific papers published by the researchers of the Polytechnic University of Turin from the beginning of 2013 to the end of 2017. The dump is a JSON file composed of 23'268 records, where each record contains the metadata about a single scientific publication. Many of the metadata are generated by the IRIS platform itself in order to manage its internal processes of acceptance and update of the publication status. We have discarded such metadata, since they do not represent significant information for the characterization of the publication.

We selected the following metadata to build the graph, being the ones that truly represent semantic information about a publication:

1. The publication identifier.

2. The title.

3. The abstract.

4. The author name, surname and identifier.

5. The contributors and coauthors names, surnames, and identifiers (if present).

6. The date of publication.

7. The journal title and ISSN (if present).

8. The keywords inserted by the authors.

The publication identifier is a unique numeric code associated by IRIS to each scientific paper. Also the authors, contributors and coauthors should be uniquely identified by an alphanumeric identifier, however, only the researchers of the Polytechnic University of Turin have such identifier assigned. External researchers that may be contributors or co-authors of the publication have only their name and surname listed. The author is instead always a Polito researcher. If the information about the journal is present, then the journal ISSN is used as identifier. The title, the abstract, the names, surnames and the keywords are simple character strings, while the date is in "dd-mm-yyyy" format.

As already discussed in previous chapters, the keywords inserted by the authors cannot be treated as semantic concepts. We wanted to solve this

issue, in order to have all the publications that refer to a specific topic linked to a same semantically unambiguous graph entity that uniquely represents such topic. To do so, we employed TellMeFirst (TMF), a tool for the automatic extraction of semantic keywords that relies on the DBpedia ontology and knowledge graph as its source of knowledge. In the following sections we will describe how we developed the graph builder and how the use of TMF allowed us to add the semantic concepts extracted from the publications abstracts as graph entities of the Polito Knowledge Graph.

5.1.1   PKG ontology and schema

Starting from the metadata discussed above, we defined the PKG ontology as composed of five different classes: the *Publication*, the *Author*, the *Journal*, the *AuthorKeyword* and the *TMFResource*.

In order to build a knowledge graph, the instances of such classes must be linked together by means of semantic relations, called predicates. To do so, we employed some predicates already defined by the FOAF [5] and the DCMI [6] ontologies, together with some terms defined by the RDF Schema standard. The graph structure obtained is represented by the schema in Figure 5.1, where the classes, the attributes of such classes and the predicates that link them together are showed.

As can be saw from the schema of Figure 5.1, the *Publication* class is linked to the author class by means of two relations both defined by the DCMI ontology: *dc:creator*[1] and *dc:contributor*[1].

We decided to only differentiate the first author (creator) from the others (contributors) due to the inability to discriminate which are the co-authors and which are the collaborators.

The *AuthorKeyword* class refers to the keywords inserted by the authors, they're kept for the sake of completion with respect to the metadata available in the dump, but they do not represent any semantic information.

Instead, the *TMFResource* class is used to instantiate the entities directly referred to the DBpedia resources that are extracted by TMF starting from publication abstract. Being uniquely identified by their DBpedia URI, each instance will be unique, and so if more publications shares

---

[1]The *dc* keyword is the prefix of the DCMI namespace: http://purl.org/dc/terms/
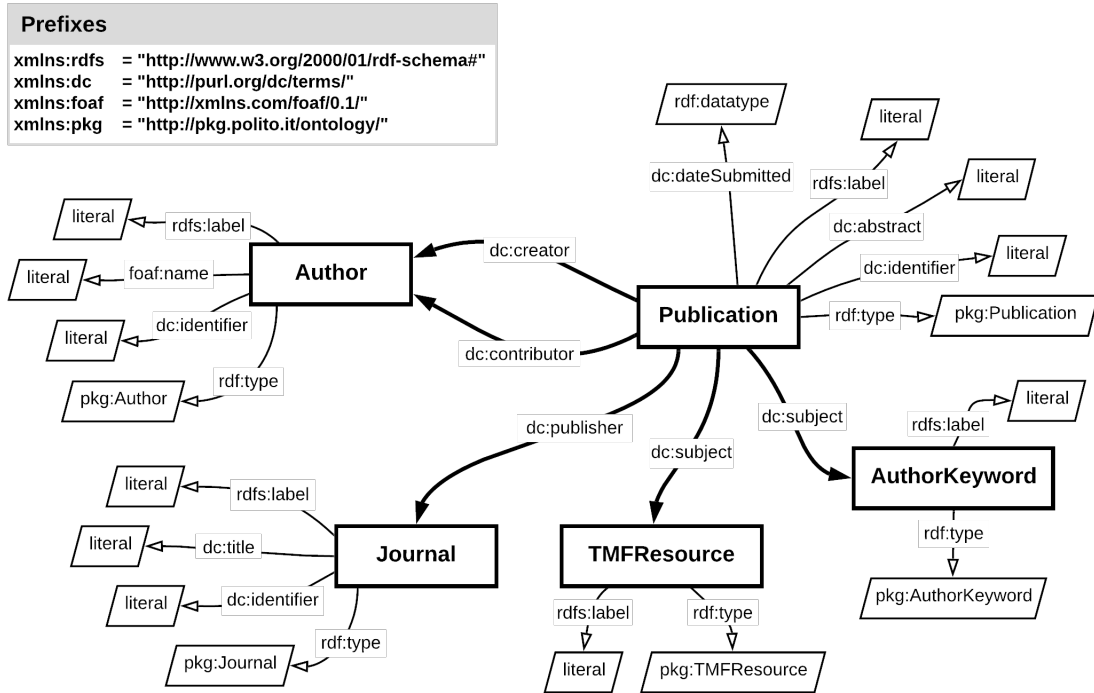
Figure 5.1.   Schema of the Polito Knowledge Graph. The ontologies used to define the structure are showed in the prefixes table.

the same extracted topic, they will be linked by the *dc:subject* relation to the same instance of the *TMFResource* that has been istantiated for such topic. An example is showed in Figure 5.2, where are showed all the publications that are linked to the TMFResource of the topic *Knowledge Base*.

5.1.2   The graph builder

We implemented the graph builder as a Python command-line script that uses the *rdflib*[2] library to create and manage an in-memory RDF representation of the graph that could be then serialized and saved as an XML file. The script takes as arguments the path of the JSON dump of IRIS, together with some options that allows to trigger specific functionalities of the script, such as:

---

[2]https://github.com/RDFLib/rdflib

Figure 5.2. Visualization of the results obtained from the Polito Knowledge Graph when running a SPARQL query that returns all the publications that have *Knowledge Base* as subject. The following is an example of a triple that links one of the publication retrieved to the above topic: *(pkgpub:2670709, dc:subject, dbp:KnowledgeBase)*

1. The update of an already existing RDF graph, this allows to add new RDF triples without the need of rebuilding the graph from scratch.

2. The number of topics that must be extracted from each abstract by TellMeFirst, the default value is seven.

3. The addition of the topics images, which are scraped from DBpedia and added as attributes of the *TMFResource* instances.

The script firstly declares the namespaces and the ontology used to define the RDF representation of the entities and the relation inside the graph as written in the previous section, then parses the arguments and options received and execute the corresponding activities.

If the creation of a new graph is requested, the script reads the JSON dump in a Python list and instantiates a *ThreadPoolExecutor*, a Python abstraction that allows to execute a function asynchronously by spawning

a predefined number of threads. Each thread asynchronously executes a function that process a single record of the dump. The concurrent access to the records list is not a problem being the Python lists implemented as thread safe containers.

Each record is processed by:

1. Matching the record metadata with the ontology classes and instantiating the corresponding entities.

2. Requesting to TellMeFirst, via its API, the extraction of the topics from the publication abstract.

In the first step when a field that matches a class is found a corresponding RDF triple that instantiates a new object is created, so a new entity is added to the graph. If the entity is already present, an RDF triple that links the publication to the already existing entity is added.

The topic extraction is requested to TMF by sending an HTTP POST request containing the publication abstract and the number of topics to be extracted to its REST API[3]. The response contains the list of the DBpedia resources that TMF found as topic of interest of the publication. This topics are added to the graph by instantiating the corresponding *TMFResource* entities, and are linked to the publication by means of the *dc:subject* relation.

Even if the Python interpreter implementation poses some limitations in the actual advantage of executing multithreaded code in CPU-bound scenarios, in our case the use of multiple threads greatly improved the time required to build the graph, given that the graph builder implementation is I/O-bound due to the communication with the REST API of TellMeFirst.

The two steps described above allows to generate a full RDF description of a publication starting from its metadata, and enriched by the topics extracted by TellMeFirst. The result is an RDF graph composed by entities that are uniquely identified and linked together by meaningful relations. Such graph fully represents the Polito scientific community, linking authors, publications, research fields and journals together.

For example, the listing that follows contains the RDF description of a publication generated by the graph builder. Not only a new publication

---

[3]http://tellmefirst.polito.it/TellMeFirst_API_Guide.pdf

44

entity is created, but also the author, contributor and subject are added to the graph, completing the knowledge base.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:dc="http://purl.org/dc/terms/"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:pkg="http://pkg.polito.it/"
    xmlns:dbpr="http://dbpedia.org/resource/"
    xmlns:xmls="http://www.w3.org/2001/XMLSchema#">

    <rdf:Description rdf:about="pkg:publications/2679709">
        <rdf:type rdf:resource="pkg:ontology/Publication"/>
        <dc:identifier>2679709</ns1:identifier>
        <rdfs:label>Title</rdfs:label>
        <dc:abstract>Abstract of the publication</ns1:abstract>
        <dc:subject rdf:resource="dbpr:Knowledge_Base"/>
        <dc:creator rdf:resource="pkg:authors/rp00000"/>
        <dc:contributor rdf:resource="pkg:authors/rp11111"/>
        <dc:dateSubmitted rdf:datatype="xmls:date">
            2017-01-01
        </dc:dateSubmitted>
    </rdf:Description>

    <rdf:Description rdf:about="dbpr:Knowledge_Base">
        <rdf:type rdf:resource="pkg:ontology/TMFResource"/>
        <rdfs:label>Knowledge Base</rdfs:label>
    </rdf:Description>

    <rdf:Description rdf:about="pkg:authors/rp00000">
        <rdf:type rdf:resource="pkg:ontology/Author"/>
        <foaf:name>Surname, Name</foaf:name>
        <dc:identifier>rp00000</dc:identifier>
        <rdfs:label>Surname, Name</rdfs:label>
    </rdf:Description>

    <rdf:Description rdf:about="pkg:authors/rp11111">
        <rdf:type rdf:resource="pkg:ontology/Author"/>
        <foaf:name>Surname, Name</foaf:name>
        <dc:identifier>rp11111</dc:identifier>
        <rdfs:label>Surname, Name</rdfs:label>
    </rdf:Description>
</rdf:RDF>
```

After all the publications have been processed, the internal representation obtained can be serialized and exported as an XML file that contains the definition of all the RDF triples that forms the Polito Knowledge Graph. Such RDF representation will be then used as input for the enhancer module or loaded into a triplestore (like Blazegraph[4]) in order to be queried by the viewer component.

5.2 Enhancing the Polito Knowledge Graph

5.2.1 The model trainer

5.2.2 The link predictor

---

[4]https://blazegraph.com/

Chapter 6

Evaluation

Chapter 7

Conclusions and future work

Bibliography

[1] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-3-319-93417-4_38.pdf

[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

[3] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[4] O. Lassila, R. R. Swick *et al.*, "Resource description framework (rdf) model and syntax specification," 1998.

[5] M. Graves, A. Constabaris, and D. Brickley, "Foaf: Connecting people on the semantic web," *Cataloging & Classification Quarterly*, vol. 43, no. 3-4, pp. 191–202, 2007. [Online]. Available: https://doi.org/10.1300/J104v43n03_10

[6] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf, "Dublin core metadata for resource discovery," *Internet Engineering Task Force RFC*, vol. 2413, no. 222, p. 132, 1998. [Online]. Available: http://www.hjp.at/doc/rfc/rfc2413.html

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv e-prints*, p. arXiv:1301.3781, Jan 2013, provided by the SAO/NASA Astrophysics Data System. [Online]. Available: https://arxiv.org/pdf/1301.3781.pdf

[8] T. N. Kipf and M. Welling, "Semi-Supervised Classification with

Graph Convolutional Networks," *arXiv e-prints*, p. arXiv:1609.02907, Sep 2016, provided by the SAO/NASA Astrophysics Data System. [Online]. Available: https://arxiv.org/pdf/1609.02907.pdf

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1263–1272. [Online]. Available: https://arxiv.org/pdf/1704.01212.pdf

[11] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014. [Online]. Available: https://arxiv.org/pdf/1412.6575.pdf

[12] O. R. Rocha, I. Vagliano, C. Figueroa, F. Cairo, G. Futia, C. A. Licciardi, M. Marengo, and F. Morando, "Semantic annotation and classification in practice," *IT Professional*, vol. 17, no. 2, pp. 33–39, Mar 2015.

[13] P. Cifariello, P. Ferragina, and M. Ponza, "Wiser: A semantic approach for expert finding in academia based on entity linking," *Information Systems*, vol. 82, pp. 1–16, 2019. [Online]. Available: https://arxiv.org/pdf/1805.03947.pdf

[14] A. Salatino, F. Osborne, T. Thanapalasingam, and E. Motta, "The cso classifier: Ontology-driven detection of research topics in scholarly articles," *TPDL 2019: 23rd International Conference on Theory and Practice of Digital Libraries*, 2019. [Online]. Available: http://oro.open.ac.uk/62026/