Fine-grained Type Prediction of Entities using Knowledge Graph Embeddings

Bachelor's Thesis of

Radina Sofronova

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr. Harald Sack
Advisor:	M.Sc. Russa Biswas

01 July 2019 – 31 October 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 31st October 2019

. . . . . .

(Radina Sofronova)

Abstract

Wikipedia is the largest online encyclopedia, which appears in more than 301 different languages, with the English version containing more than 5.9 million articles. However, using Wikipedia means reading it and searching through pages to find the needed information. On the other hand, DBpedia contains the information of Wikipedia in a structured manner, that is easy to reuse.

Knowledge bases such as DBpedia and Wikidata have been recognised as the foundation for diverse applications in the field of data mining, information retrieval and natural language processing.

A knowledge base describes real-world objects and the interrelations between them as entities and properties. The entities that share common characteristics are associated with a corresponding type. One of the most important pieces of information in knowledge bases is the type of the entities described. However, it has been observed that type information is often noisy or incomplete. In general, there is a need for well-defined type information for the entities of a knowledge base.

In this thesis, the task of *fine-grained entity typing* of entities of a knowledge base, more specifically - DBpedia, is addressed. There are a lot of entities in DBpedia that are not assigned to a fine-grained type information, rather assigned to either coarse-grained type information or to `rdf:type owl:Thing`. Fine-grained entity typing aims at assigning more specific types, which are more informative than the coarse-grained ones.

This thesis explores and evaluates different approaches for type prediction of entities in DBpedia - the unsupervised approach *vector similarity* using knowledge graph embeddings, as well as the supervised one - *CNN classification*. Knowledge graph embeddings from the pre-trained RDF2Vec model are used.

Zusammenfassung

Wikipedia ist das größte Online-Enzyklopädie, die in mehr als 301 Sprachen erscheint. Lediglich die englische Sprachversion enthält mehr als 5.9 Millionen Artikeln. Damit man die Informationen von Wikipedia verwenden kann, muss man die Artikel lesen, um die benötigten Informationen zu finden. Andererseits enthält DBpedia die Informationen von Wikipedia in einer strukturierten Weise, die einfach wiederzuverwenden ist.

Wissensdatenbanken wie DBpedia und Wikidata wurden als Grundlage für verschiedene Anwendungen im Bereich Data Mining, Informationsaufbereitung und Verarbeitung natürlicher Sprache verwendet.

Eine Wissensdatenbank beschreibt Objekte und ihre Beziehungen als Entitäten und Prädikate. Die Entitäten, die gemeinsame Merkmale aufweisen, sind einem entsprechenden Typ zugeordnet. In vorhergehenden Untersuchungen wurde festgestellt, dass die Typinformationen häufig verrauscht oder unvollständig sind. Es besteht ein Bedarf an genau definierten Typinformationen für die Entitäten der Wissensdatenbanken.

In dieser Arbeit wird die Aufgabe der *feingranularen Typvorhersage von Entitäten* einer Wissensbasis, in dem vorliegenden Fall - DBpedia, behandelt. Es gibt viele Entitäten in DBpedia, die keiner feingranularen Typinformation zugeordnet sind, sondern entweder grobgranularer Typinformation oder `rdf:type owl:Thing`. Die feingranulare Typvorhersage von Entitäten hat das Ziel, spezifischere Typen zuzuweisen, die informativer sind als die grobgranularen.

In dieser Bachelorarbeit werden verschiedene Ansätze zur Typvorhersage von Entitäten in DBpedia untersucht und bewertet - das unüberwachte Lernverfahren *Ähnlichkeit von Vektoren* mithilfe von Einbettungen in Wissensgraphen sowie das überwachte Lernverfahren - *CNN-Klassifikation*. Dabei werden Wissensgrapheinbettungen aus dem vortrainierten RDF2Vec-Modell genutzt.

Contents

Abstract i

Zusammenfassung iii

1. Introduction 1
- 1.1. Motivation and Background . . . 1
- 1.2. Objective . . . 3
- 1.3. Structure of the thesis . . . 3

2. Foundations 5
- 2.1. Semantic Web . . . 5
  - 2.1.1. Resource Description Framework . . . 5
  - 2.1.2. Knowledge Base . . . 6
    - 2.1.2.1. Entity . . . 7
    - 2.1.2.2. Entity Typing . . . 8
    - 2.1.2.3. DBpedia . . . 8
- 2.2. Deep Learning . . . 9
  - 2.2.1. Knowledge Graph Embeddings . . . 9
    - 2.2.1.1. RDF2Vec . . . 9
  - 2.2.2. Neural Networks . . . 12
- 2.3. Similarity Measures . . . 18
  - 2.3.1. Cosine Similarity . . . 18
- 2.4. Set Theory . . . 19

3. Related Work 21

4. Approach 23
- 4.1. Vector Similarity . . . 23
- 4.2. 1D Convolutional Neural Network . . . 26

5. Evaluation 29
- 5.1. Dataset . . . 29
- 5.2. Experimental Setup and Results . . . 31

6. Conclusion 37
- 6.1. Summary . . . 37
- 6.2. Future Work . . . 37

Bibliography **39**

A. Appendix **41**
A.1. List of 59 classes used in the scope of the thesis 41
A.2. Correlation 43

List of Figures

1.1. Distribution of the entities of five types in DBpedia. 2

2.1. RDF Triples. 6
2.2. RDF Graph. 7
2.3. Types in DBpedia Ontology structure. 9
2.4. RDF2Vec (Amended from [1]). 10
2.5. CBOW and Skip-Gram models [17]. 11
2.6. CBOW and Skip-Gram vector constructions [17]. 12
2.7. Components of a typical CNN layer [7]. 14
2.8. An example of 2D convolution [7]. 15
2.9. Sparse interactions and Parameter sharing [8] 16
2.10. Dropout applied to a fully-connected neural network [8]. 18
2.11. Cosine similarity between two vectors - different examples. 19

4.1. Vector Similarity Approach. 24
4.2. Generation of the class vector of `dbo:Library`. 26

5.1. Hits@1 - RDF2Vec 33
5.2. Hits@3 - RDF2Vec 34
5.3. Hits@1 - SetTheory 35
5.4. Hits@3 - SetTheory 36

A.1. Correlation - Actor. 43
A.2. Correlation - AdultActor. 43
A.3. Correlation - Airline. 43
A.4. Correlation - Artist. 44
A.5. Correlation - Athlete. 44

List of Tables

1.1. Distribution of entities in subclasses. 3

5.1. Accuracy of 1D CNN models. 32

A.1. Correlations - Part 1. 45
A.2. Correlations - Part 2. 46

1. Introduction

Wikipedia, as a multilingual online encyclopedia created and maintained as an open collaboration project, is currently among the top ten most popular websites and the most widely used encyclopedia [12]. However, the information in Wikipedia is structured and unstructured, making not all of it machine-understandable. Structured data in Wikipedia is presented in the form of an infobox containing property value pairs, which summarize the content of a Wikipedia article. On the contrary, DBpedia, as a knowledge graph, stores knowledge in a machine-readable form and provides a way for information to be organised, searched and utilised. The information in DBpedia is derived by extracting structured data from Wikipedia pages through an external framework. Knowledge bases such as DBpedia are used in the field of data mining, information retrieval and natural language processing.

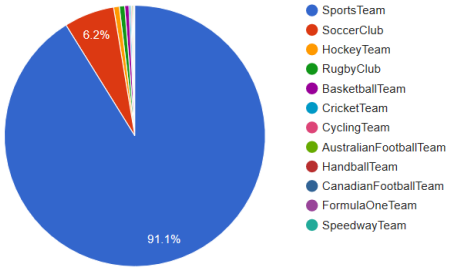
A knowledge base contains a set of facts about the entities and represents the knowledge in a structured repository [3]. Another important feature of a knowledge base is that it groups the entities in classes by defining their type, e.g. *Germany* is an entity which has the type *Country*. Type information is very important in a knowledge base.

1.1. Motivation and Background

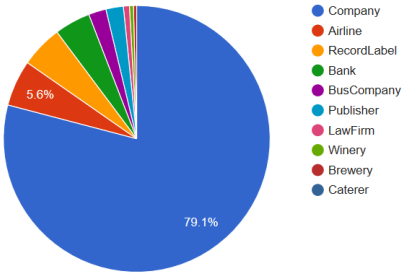
Entity typing is the process of assigning a type to an entity and is a fundamental task in knowledge base construction. Types in a knowledge base are organized as a hierarchical structure - a type hierarchy. Traditional entity typing focuses on a small set of types - coarse-grained types, e.g. Person, Organization, Settlement. Those types are at the top levels of the type hierarchy and do not provide very specific information. On the other hand, fine-grained entity typing assigns more characteristic types to an entity.

One primary problem in this domain is that the majority of the entities have a coarse-grained type. A proof to this statement is given in table 1.1 and figure 1.1.

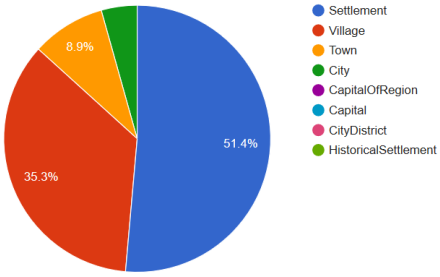
Table 1.1 gives information about the distribution of the entities of five types, namely Sports Team, Company, Settlement, Activity and Event. The first column gives the total count of entities which come under the corresponding class in the type hierarchy, whereas the second column contains the count of the entities that have exactly the given type, excluding its subclasses. The pie charts in figure 1.1 show the distribution of entities of a



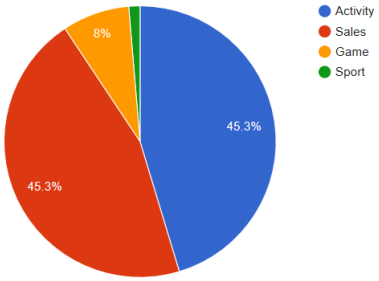
(a) Distribution of the entities of type Sports Team and its subclasses.



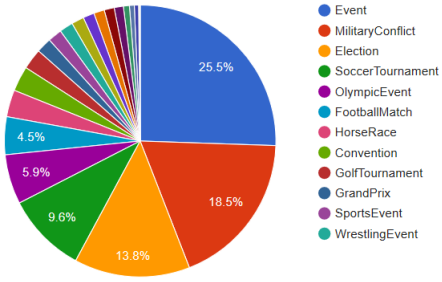
(b) Distribution of the entities of type Company and its subclasses.



(c) Distribution of the entities of type Settlement and its subclasses.



(d) Distribution of the entities of type Activity and its subclasses.



(e) Distribution of the entities of type Event and its subclasses.

Figure 1.1.: Distribution of the entities of five types in DBpedia.

Class	# total entities	# entities without subclasses	Percentage
SportsTeam	352006	320835	91.1%
Company	70208	55524	79.1%
Settlement	478906	246163	51.4%
Activity	19464	8824	45.3%
Event	76029	19418	25.5%

Table 1.1.: Distribution of entities in subclasses.

particular class in its subclasses. Each chart is divided into parts, which present a given subclass. It is clearly visible that the greater part of the entities have a coarse-grained type.

1.2. Objective

The objective of this thesis is to explore and evaluate different approaches for type prediction of entities in DBpedia.

One of the analyzed approaches is *unsupervised*, namely - vector similarity (sec. 4.1), using knowledge graph embeddings from the pre-trained RDF2Vec model. The unsupervised model relies on the knowledge mining performed by the knowledge graph embedding algorithm, instead of training a similarity estimator.

One supervised approach is also explored - convolutional neural networks (CNNs) (sec. 4.2). The vector representations of the entities from the pre-trained RDF2Vec model are used. A CNN performs a classification task - categorization of entities into different types.

1.3. Structure of the thesis

The structure of the thesis is as follows: In chapter 2, background information and relevant terms are introduced, which are important for a good understanding of the rest of the thesis. Chapter 3 describes the approach developed in the course of the thesis, followed by a presentation, an evaluation and an analysis of the results in chapter 4. Finally, chapter 5 concludes the thesis and presents future directions of research.

2. Foundations

This chapter presents the basic concepts that are important for the understanding of the rest of the thesis. They are separated in two main fields - semantic web and deep learning. The concepts introduced in this chapter are from those two domains.

2.1. Semantic Web

The *Semantic Web* represents a new vision about how the Web should be constructed so that its information can be processed automatically by machines on a large scale. It is closely related to the traditional Web. The term "Semantic Web" was coined by the inventor of the WWW, Sir Tim Berners-Lee, who defined it as follows: "Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [4]. Overall, the Semantic Web allows machines to understand the meaning (semantics) of the information on the Web [19].

2.1.1. Resource Description Framework

The *Resource Description Framework (RDF)* is the main building block for the Semantic Web, as it allows knowledge representation in a structured and machine-understandable manner. RDF is a standard for encoding data and is used for representing information about resources and their relations existing in the real world. RDF decomposes information into facts, so that each fact has a clearly defined form, in order to be machine-understandable.

RDF defines the facts as statements. Each RDF statement is a triple, which has the following form: <*subject, predicate, object*>, with the order of the elements of a triple being strict. The subject and object are the resources, which have some relationship and the predicate defines the nature of this relationship. A collection of RDF statements is called an RDF Graph and represents some knowledge as a collection of pieces of information.

The three components of an RDF triple are resources and for each resource a *Uniform Resource Identifier (URI)* is created in order to identify it uniquely and globally. The object is not necessarily identified by a URI, as it can also be a literal, in the case that it describes

data values that do not have a separate existence. URIs are written in angular brackets, whereas the literals are written in quotation marks.

In the definition of the Semantic Web, it was stated that it allows machines to understand the meaning (semantics) of the information on the Web. The RDF statements describe resources and relations between them, but the meaning is still missing. A way to introduce semantics in the RDF data is defined by the *RDF Schema (RDFS)*. "RDFS is an extendable knowledge representation language that one can use to create a vocabulary for describing classes, sub-classes and properties of RDF resources. [19]" RDFS introduces the separation of resources into groups, namely classes. The classes themselves can be related with the property `rdfs:subClassOf`, which results in an hierarchical relationship of the classes.

As previously mentioned, a collection of RDF triples forms a directed graph. The vertices of the graph are the subjects and objects, whereas the edges of the graph are the predicates. Two important predicates are `rdf:type` and `rdfs:subClassOf`. The first predicate is used to connect an entity to its type, whereas the second one is used to connect a class to one of its subclasses. An example of a collection of RDF triples is listed in fig. 2.1. The same example in a form of an RDF Graph is depicted in fig. 2.2. A short form of the URIs is used in the graphs for better visibility. The prefixes used are as follows: *dbr* stands for *<http://dbpedia.org/resource/>* and *dbo* stands for *<http://dbpedia.org/ontology/>*.

<http://dbpedia.org/ontology/Athlete> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://dbpedia.org/ontology/Person>.

<http://dbpedia.org/ontology/TennisPlayer> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://dbpedia.org/ontology/Athlete>.

<http://dbpedia.org/resource/Allan_Dwan> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Person>.

<http://dbpedia.org/resource/Carl_Lewis> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Athlete>.

<http://dbpedia.org/resource/Rafael_Nadal> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/TennisPlayer>.

<http://dbpedia.org/resource/Roger_Federer> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/TennisPlayer>.

Figure 2.1.: RDF Triples.

The red nodes in the graph depict some instances of the class `dbo:Person`. The green nodes depict classes, as well as their hierarchical relationship. In the example, `dbr:Roger_Federer` is of type `dbo:Tennis_Player`. `dbo:Tennis_Player` is a subclass of `dbo:Athlete`, which itself is a subclass of `dbo:Person`. Therefore, it can be deduced that `dbr:Roger_Federer` is of type `dbo:Tennis_Player`, as well as `dbo:Athlete`, as well as `dbo:Person`. The same applies to the resource `dbr:Rafael_Nadal`. In the presented RDF Graph `dbr:Alan_Dwan` is an instance of the class `dbo:Person`, and `dbr:Carl_Lewis` is of type `dbo:Athlete` and `dbo:Person`.

2.1.2. Knowledge Base

Knowledge bases create an organized collection of data, adding a semantic model to the data, which includes a formal classification with classes, subclasses, relationships and

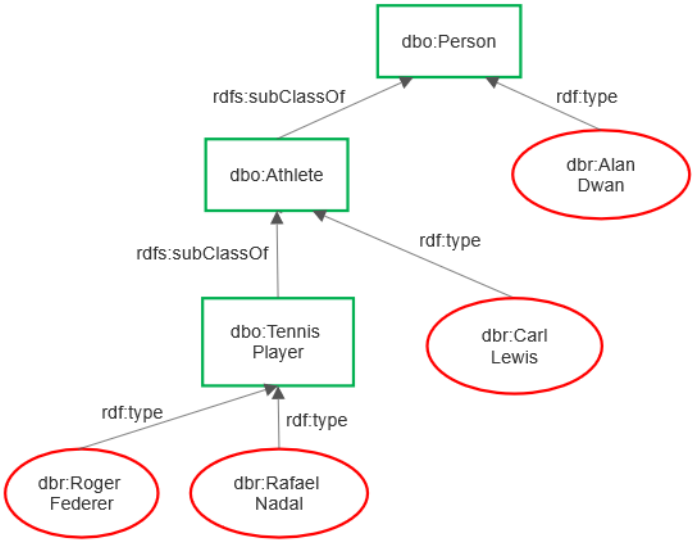


Figure 2.2.: RDF Graph.

instances (ontologies and dictionaries), on one hand, and rules for interpreting the data, on the other. Balog [3] describes a knowledge base as follows: "A knowledge base (KB) is a structured knowledge repository that contains a set of facts about the entities."

Knowledge bases consist of sets of triples $K \subseteq E \times R \times (E \cup L)$, where E is a set of resources referred to as entities, R - a set of relations and L - a set of literals. An entity is identified by a URI and represents a real-world object. A relation is a predicate and a literal is a string, date, or number. In a triple <s, r, o>, s is known as a subject, r as a relation, and o as an object [5].

Examples of knowledge bases are DBpedia, Wikidata, YAGO etc. The focus of this thesis lies on predicting type information for entities in DBpedia. Therefore, DBpedia is the knowledge base, which is mainly analyzed in the course of the thesis. DBpedia is thoroughly described in sec. 2.1.2.3.

2.1.2.1. Entity

An *entity* is an object that can be distinctly identified. Common entities are people, locations, products, events etc. Further, an entity is characterized by its identifier, type, attributes and relationships to other entities. Of central importance in the course of this thesis is the type of an entity.

2.1.2.2. Entity Typing

Entities of a knowledge base are categorized into multiple *entity types*. Types can also be regarded as classes (semantic categories) that group together entities with similar properties. Analogy can be made to object-oriented programming, whereby an entity of a type is like an instance of a class. The members of a class are known as instances of the class. The set of possible entity types are often organized in a hierarchical structure. For example, the entity `Qatar Airways` is an instance of the type `Airline`, which is a sub-type of `Organisation`. `Qatar Airways` is therefore associated with a type-path `/Thing/Agent/Organisation/Company/Airline`, having the fine-grained type `Airline` and a coarse-grained type - `Organisation`. Fine-grained entity typing is an important sub-task of knowledge base completion and will be explored in detail in the course of the thesis.

- Coarse-grained Entity Typing assigns small set of types such as Person, Organisation, Place to the entities in a knowledge base.
- Fine-grained Entity Typing assigns more specific types such as Artist, Tennis Player, Company to the entities in a knowledge base. Those types are much more informative and precise.

2.1.2.3. DBpedia

DBpedia is a knowledge base that is derived by extracting structured data from Wikipedia through an open source extraction framework [13]. It is presented in [12]. The extracted information for each Wikipedia page takes the form of an RDF graph. The collection of all RDF graphs forms a large RDF dataset. This dataset can be viewed as Wikipedia's machine-readable version, with the original Wikipedia remaining the human-readable one. DBpedia allows asking sophisticated queries against Wikipedia and linking other datasets on the Web to Wikipedia [19].

DBpedia is not a result of a one-time process but rather of a continuous community effort, with numerous releases since its establishment in 2007. The research carried out in this thesis is based on the latest release that is available at the time of writing, DBpedia 2016-10, and especially on the English version.

The *DBpedia Ontology* forms the structural backbone of DBpedia. It defines classes and properties which organize the resources. The DBpedia Ontology was created manually by considering the most frequently used infoboxes in Wikipedia. The attributes of the infoboxes are mapped to the classes and properties, when RDF statements are generated. The DBpedia Ontology is clean and consistent, but its coverage is limited to entities that have an associated infobox [19].

The DBpedia Ontology consists of 760 types, forming a 7-level type hierarchy. The type hierarchy is a directed acyclic graph, which provides a way to categorize and organize entities. Fig. 2.3 shows an example of a part of the DBpedia Ontology of height 4. From the figure can be seen that the root node is Thing. In later analysis the root will be ignored, as it gives no information regarding the type of an entity.

To summarize, DBpedia is a huge collection of RDF graphs, with precisely defined semantics. A way to access DBpedia is to directly download its RDF dump files. This process is presented in chapter 5.

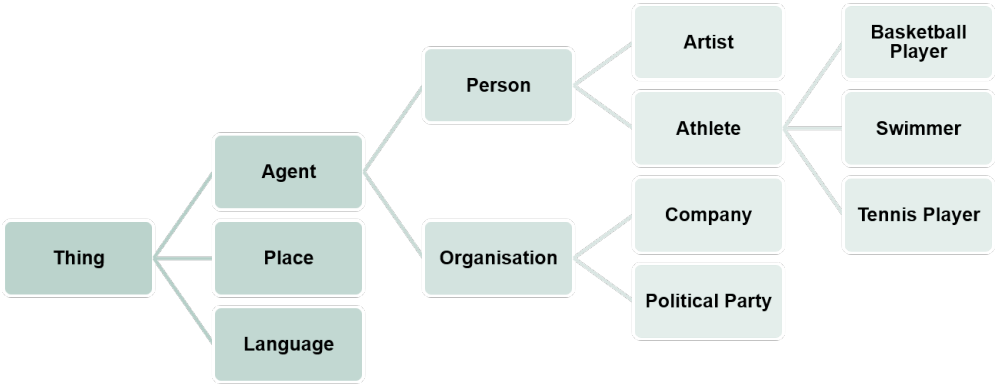


Figure 2.3.: Types in DBpedia Ontology structure.

2.2. Deep Learning

2.2.1. Knowledge Graph Embeddings

A knowledge graph is a representation of a knowledge base as a graph. Knowledge Graphs such as DBpedia are a valuable source of background information for a list of tasks in the field of data mining, natural language processing, information retrieval and knowledge extraction. The key idea in the concept of knowledge graph embedding is to represent components of a knowledge graph (i.e., entities and relations) as k-dimentional vectors in a continuous vector space in order to simplify manipulation while preserving the structure of the knowledge graph.

2.2.1.1. RDF2Vec

In this thesis, a pre-trained knowledge graph model is used. This subsection introduces RDF2Vec - a generic method for embedding entities in knowledge graphs into lower-

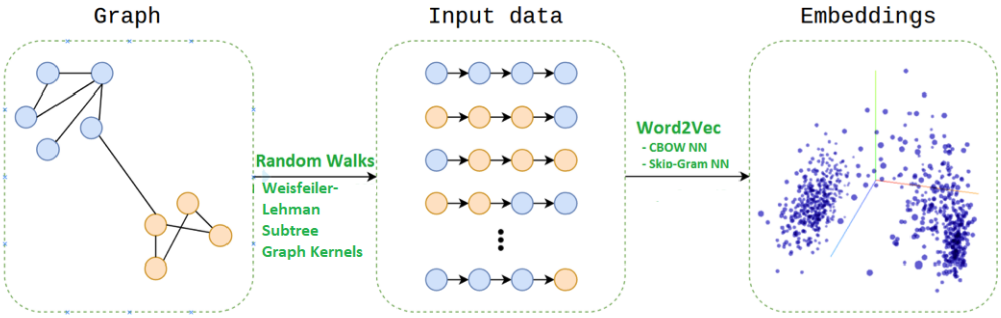


Figure 2.4.: RDF2Vec (Amended from [1]).

dimensional vector spaces. The generation of the entities' vectors is task and dataset independent, meaning that, once the vectors are generated, they can be used for any task and algorithm, e.g., SVM, Random Forests, Naive Bayes, Neural Networks, KNN etc. In this thesis, the entity vectors generated by RDF2Vec are used to determine the cosine similarity between the vector representations of the entities in the vector space. The vectors are also used for training a 1D Convolutional Neural Network model. A visualisation of the function of the model is presented in fig. 2.4.

RDF2Vec [17] is an approach that uses language modeling approaches for unsupervised feature extraction from sequences of words and adapts them to RDF graphs. The vector representations of DBpedia entities are provided as ready-to-use files.

First Step - Entity Sequences

RDF2Vec adapts the Word2Vec neural language model for RDF graph embeddings. The Word2Vec approach uses the word order in text documents and models the assumption that closer words in the word sequence are statically more dependent [17]. RDF2Vec considers entities and relations between entities instead of words and word sequences. This is also the first step of the RDF2Vec algorithm - to extract paths of entities from a knowledge graph. In order to transform the graph data into paths of entities, two different approaches are used - Random Graph Walks and Weisfeiler-Lehman Subtree Graph Kernels.

- The *Random Graph Walks* approach generates all graph walks of depth d rooted in the vertex v through breath-first algorithm. These graph walks are generated for every vertex in the graph. The final set of sequences is the union of all graph walks found for all the vertices in a graph [17].

- The *Weisfeiler-Lehman Subtree RDF Graph Kernels* approach evaluates the distance between two instances by counting common subtrees in the graph. The kernel computes the number of subtrees shared between two or more graphs by using the Weisfeiler-Lehman test of graph isomorphismus [17]. An adaptation of the algorithm is needed and is done by considering directed edges and reusing the labels of the previous iteration, if they are identical with the ones of the current iteration.

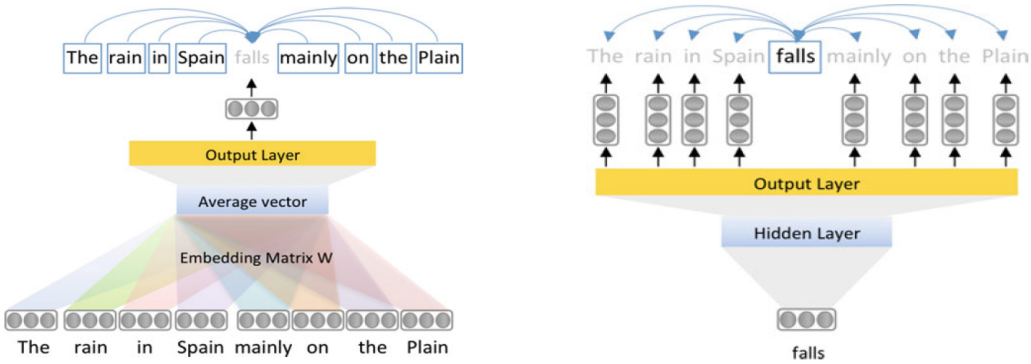
For each vertex v, all the paths of depth d within the subgraph of the vertex on the relabeled graph are extracted. Then original label of the vertex v is set as the starting token of each path. This process is repeated until the maximum number of iterations is reached. The union of the sequences of all the vertices in each iteration is the final set of sequences [17].

After having obtained the sequences of the entities, the neural language model can be trained, in order to represent each entity in the RDF graph as a vector of numerical values in a feature space.

Second Step - Word2Vec

Word2Vec is the most popular and widely used neural language model. It was proposed by Mikolov [14]. It is a two-layer neural net model for learning word embeddings from raw text. In the case of RDF2Vec, Word2Vec estimates the likelihood of a sequence of entities appearing in the graph. There are two algorithms, namely *Continuous Bag-of-Words (CBOW)* and *Skip-Gram model*. In following, both algorithms are presented.

- The *Continuous Bag-of-Words (CBOW)* (fig. 2.5) predicts target words from context words within a given window. The *input layer* contains the average of the input vectors of all surrounding words from the weight matrix. The result of the first layer is projected in the projection layer. Finally, a score for each word is computed by using the weights from the output weight matrix. This score is the probability that a word is a target word [17].
- The *Skip-Gram model* (fig. 2.5) does exactly the opposite - predicts context words from target words.

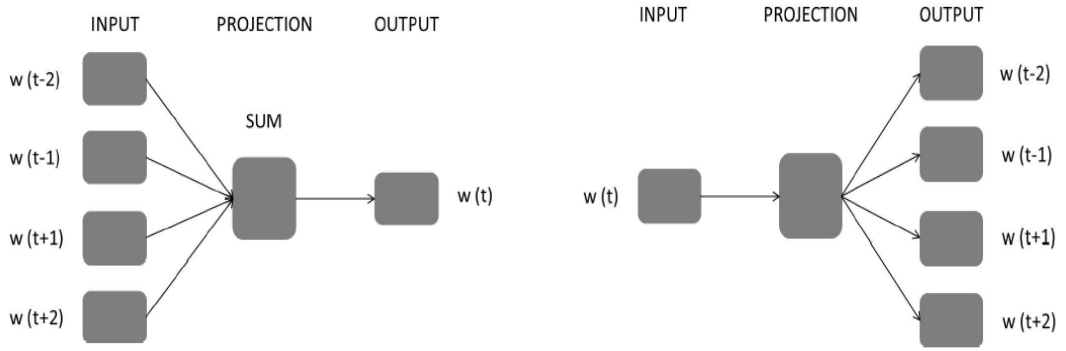


(a) CBOW model (b) Skip-Gram model

Figure 2.5.: CBOW and Skip-Gram models [17].

The vector constructions of both models are presented in fig. 2.6. As optimizations hierarchical softmax and negative sampling are used. After the training, all words (in the case of RDF2Vec - entities and properties in the path) are projected into a lower-

dimensional feature space and semantically similar words (entities) are positioned close to each other.



(a) CBOW vector construction (b) Skip-Gram vector construction

Figure 2.6.: CBOW and Skip-Gram vector constructions [17].

2.2.2. Neural Networks

Neural Network

A network consists of several layers, namely an input layer, an output layer and one or more hidden layers. The *input layer* receives the data and sends it to the first hidden layer. The *hidden layers* themselves transform the input through a non-linear function with the purpose of getting a more abstract representation of the data. Having more than one consecutive hidden layers is preferable, as the deeper the neural network is, the more abstract representation can be found. The multiple hidden layers are also the reason for the name of the term "deep" learning. After all hidden layers, the *output layer* transforms the data to an output format, which can be a set of label scores (in the case of classification tasks) or a binary true or false.

Each layer of the neural network consists of neurons. A *neuron* is a mathematical function that models the functioning of a biological neuron. Typically, a neuron computes the weighted sum of all inputs. The connection between two neurons of successive layers has a weight. The *weight* defines the influence of the input to the output for the next neuron. In a neural network, the initial weights are random and are updated iteratively to learn to predict a correct output during the model training [15]. The weighted sum of all the inputs is passed through a non-linear function, called *activation function.* The input of a neuron is also influenced by a bias - an individual bias value is associated with every neuron and is updated in each iteration.

More formally,

$$y = a(w^\intercal x + b).$$

In this equation, x is the input in the form of a vector, y is the output. w is the vector, containing the weights and b is the bias. The function a is the activation function. There is a series of activation functions that can be used, but all of them have the following three properties in common: they are differentiable, bound the output and are non-linear. Examples of activation functions are the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^x}$$

and the hyperbolic tangent function:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Neural networks are used to perform supervised learning and prediction tasks. Input data is provided to the neural network. This input data is named a *training dataset*. A training dataset contains labeled data, which means that the result is known. In this thesis, the training dataset consists of the vector representation of some entities and their types as labels. The model is evaluated, using a *test dataset*. It has the same structure as the training dataset, but contains data that is not present in it. Using this data, the performance of the model is evaluated. It is important that the training and test datasets are disjunctive.

Convolutional Neural Network

Convolutional neural networks (CNNs), first introduced by Le Cun [11], are a specialized kind of neural networks for processing data that has a known grid-like topology, e.g. time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, or image data, which can be thought of as a 2D grid of pixels [7].

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution, which is a kind of linear operation. CNNs are neural networks that use convolution in place of matrix multiplication in at least one of their layers [7]. Convolutional neural networks (CNNs) are a building block in constructing complex deep learning solutions for various natural language processing, speech, and time series tasks [8].

A typical convolutional layer consists of three stages, as depicted in fig. 2.7. The first stage performs several convolutions in parallel in order to produce a set of linear activations. In the second stage, each of those linear activations is run through a non-linear activation function. In the third stage, a pooling function is used to further modify the output of the layer [7].

In a regular neural network, the transformation between two subsequent layers involves multiplication by the weight matrix which has the same size as the input. By contrast, in convolutional neural networks, a weight matrix of a size smaller than the input size is slided over the input. When this weight matrix is placed over a set of input values, the input values are multiplied by the weights on top of them, summed and the central input

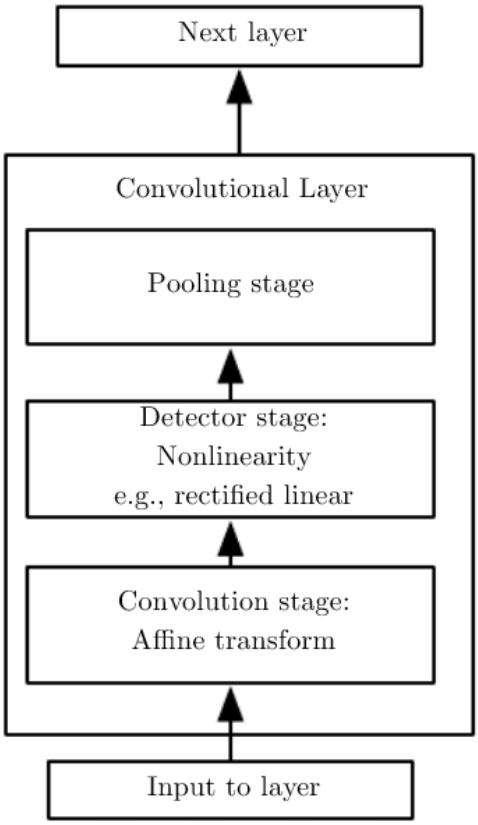


Figure 2.7.: Components of a typical CNN layer [7].

value is replaced by the sum.

Convolution

Convolution is the mathematical operation, performing the sliding. A convolution is initially an operation performed on *linear time-invariant systems*. A system or transformation is linear time-invariant, if for two functions *x(t)* and *y(t)* the following applies: If $y(t) = T(x(t))$, then $y(t - s) = T(x(t - s))$. As already discussed, convolution is the operation, in which an input function *x(t)* is combined with a function *h(t)* to give a new output that indicates an overlap between *x(t)* and the reverse translated version of *h(t)* [8]. The convolution operation is typically denoted with an asterisk and is defined as follows in the continuous domain:

$$y(t) = (h \times x)(t) = \int_{\infty}^{-\infty} h(\tau) \times (t - \tau) d\tau$$

and in the discrete domain it is defined as:

$$y(i) = (h \times x)(i) = \sum_n h(n) \times (i - n).$$

Both formulas are given for the 1-dimension case, as the data used in this thesis is 1-dimensional.

In convolutional network terminology, the first argument to the convolution (the function h) is referred to as the *input*, and the second argument (the function x) - as the *kernel* (or *filter*). The output is referred to as the *feature map* [7].

Convolution improves a machine learning system. Two of the characteristics responsible for the improvement are *sparse interactions* and *parameter sharing*.

Sparse Interactions

As previously mentioned, neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. Convolutional networks, on the contrary, have sparse interactions. This is accomplished by making the *kernel* smaller than the input [7]. An example with a 2D convolution is given in fig. 2.8. The filter size is 2x2, being smaller than the input size of 3x4.

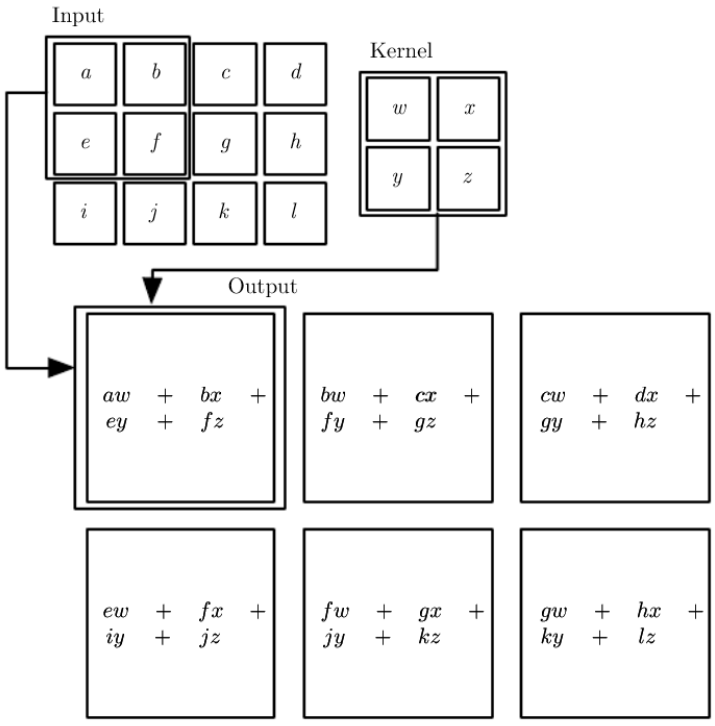
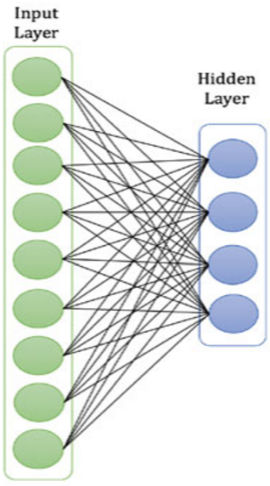
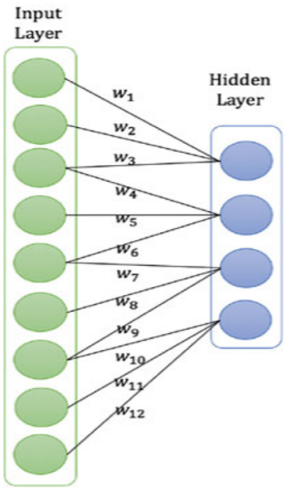


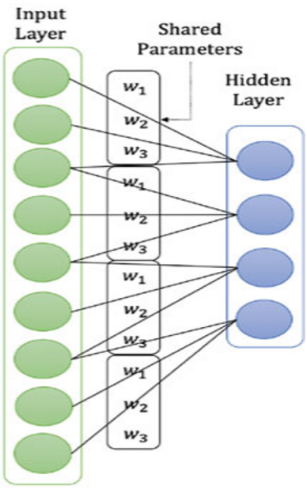
Figure 2.8.: An example of 2D convolution [7].



(a) Fully connected layers.



(b) Locally connected layers.



(c) Locally connected layers with parameter sharing.

Figure 2.9.: Sparse interactions and Parameter sharing [8]

Fig. 2.9 gives a graphical demonstration of sparse connectivity with an example of a fully-connected layer, as well as an example with sparse interactions (the two layers being locally connected). The fully-connected layer has 36 connections, respectively weights, that the neural network has to learn. With the sparse interactions the connections are reduced to 12. There is an improvement in efficiency, as there are fewer parameters to be stored and also computing the output requires fewer operations.

Parameter Sharing

In a convolutional neural network, each member of the kernel is used at every position of the input. The parameter sharing used by the convolution operation results in learning only one set, rather than learning a separate set of parameters for every location [7]. The example of fig. 2.9 is given also with a locally-connected layer with parameter sharing. It can be claimed that the number of parameters is further reduced from 12 to 3 (the dimension of the filter) [8].

Detector stage

The detector stage takes the output of the convolution, which is an affine transformation and feeds it into a non-linear function. This non-linear function is normally the sigmoid, hyperbolic tangent (as defined previously), or ReLU - $f(x)=max(0,x)$. In most cases, ReLU has proven to achieve best results [15].

Pooling

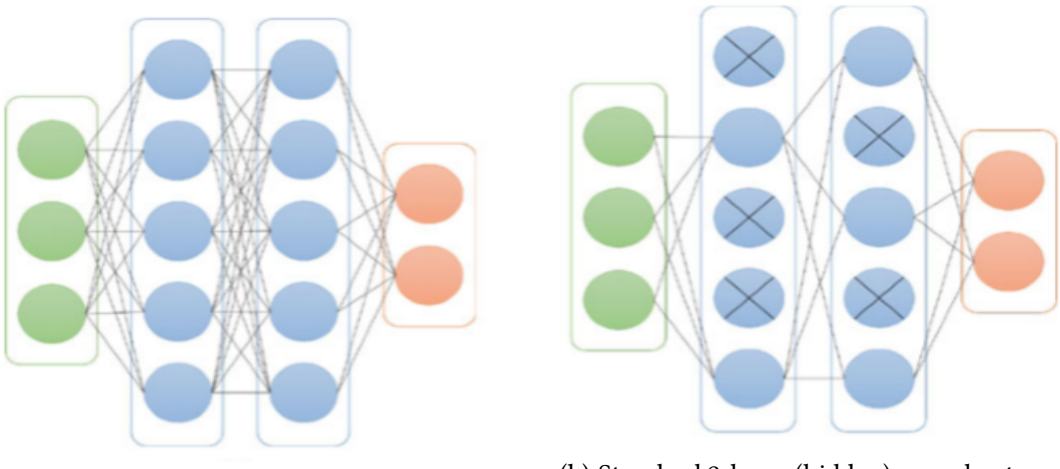
The output from the detector stage is the input for the pooling layer. A pooling function captures summary statistics of sub-regions. The nearby inputs to a certain location influence the result in this particular location after the pooling. There are different pooling methods - max pooling, average pooling, L2-norm pooling, stochastic pooling, spectral pooling. *Max pooling*, e.g., as the name suggests, chooses the maximum value of neurons from its inputs. In *average pooling*, the local neighborhood neuron values are averaged to give the output value [8].

Regularization

Regularization is used in order to prevent overfitting. Some methods focus on reducing the capacity of the models by penalizing the abnormal parameters in the objective function by adding a regularization term. Other approaches limit the information provided to the network (e.g., dropout) or normalize the output of layers (batch normalization) [8].

In the model build in the course of the thesis, dropout was used as a regularization method. It is one of the most common regularization methods in deep learning. Dropout is a simple and highly effective method to reduce overfitting of neural networks [8]. Some learned connections may work for the training data but do not generalize to the test data. Dropout aims to correct this tendency by randomly “dropping out” connections in the neural network training process. Applying dropout to a network is actually applying a

random mask matrix elementwise (multiplication by 0) during the feed-forward operation. This results in a prediction not depending on any single neuron during training. The regularization method dropout is depicted in fig. 2.10.



(a) Standard 2-layer (hidden) neural network.

(b) Standard 2-layer (hidden) neural network with dropout.

Figure 2.10.: Dropout applied to a fully-connected neural network [8].

Fully-connected Layer

The output of the pooling stage is passed to the next layer. It can be again a convolutional layer, or an output layer. In the case it is an output layer, it can be a fully-connected layer using a softmax activation function. Its output is the probability distribution over the categories (labels) in the case of a classification task.

2.3. Similarity Measures

Calculating entity relatedness and similarity are fundamental problems in numerous tasks in information retrieval, natural language processing, and web-based knowledge extraction [6]. As previously mentioned, in the RDF2Vec embedding space (see sec. 2.2.1.1) semantically similar entities appear close to each other in the feature space. Therefore, the problem of calculating the similarity between two instances is a matter of calculating the distance between two instances in the given feature space. To do so, similarity measures are used and applied on the vectors of the entities.

2.3.1. Cosine Similarity

Cosine similarity is a measure that computes the cosine of the angle between two vectors projected in a multi-dimensional space. Typically, the angle between two vectors is used

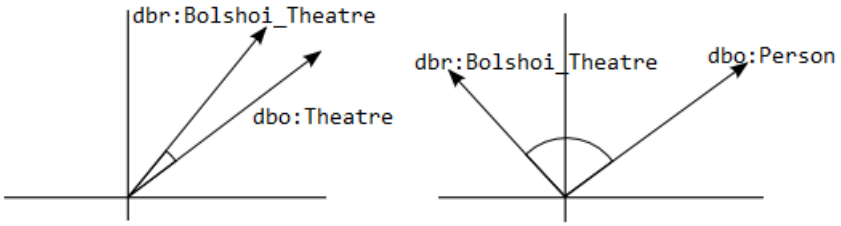


Figure 2.11.: Cosine similarity between two vectors - different examples.

as a measure of divergence between the vectors, and cosine of the angle is used as the numeric similarity - since cosine has the nice property that it is 1 for identical vectors and 0 for orthogonal vectors [18].

Therefore, this metric indicates the degree of similarity. The cosine of 0° is 1, and for any angle in the interval (0,180°] is less than 1. It is thus a judgment of orientation: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at 90° relative to each other have a similarity of 0. Even if a vector is pointing to a point far from another vector, they still could have a small angle if they point in the same direction. That is the central point on the use of cosine similarity. The mathematical formula for the cosine similarity between two instances is given as:

$$sim(e_1, e_2) = cos(\theta) = \frac{V_1 \cdot V_2}{\|V_1\| \cdot \|V_2\|}. \tag{2.1}$$

In this thesis, the cosine similarity between a vector of an entity and a vector of an entity type is explored. The correlation is the following: the smaller the angle, the higher the similarity. Fig. 2.11 gives an example. The left part is an example of how similar the entity dbr:Bolshoi_Theatre to the type dbo:Theatre is - we would expect that those two vectors have a high similarity score, as Bolshoi Theatre is a theatre. The second is an example of the similarity of the same entity dbr:Bolshoi_Theatre to the type dbo:Person - they have a low similarity score and the angle between their vectors could be around 90°.

2.4. Set Theory

Sets are viewed as collections of things while *elements* are viewed as those things which belong to sets. Normally, a set is defined in terms of certain *properties* shared by its elements. These properties must be well described, with no ambiguities, so that it is always clear whether a given element belongs to a given set or not [2]. The expression $x \in A$ denotes that x is a element in A.

Some basic axioms of the Set Theory, worth mentioning, are:

- **Axiom of extend** For a member x and sets A and B, $[A = B] \Leftrightarrow [x \in A \Leftrightarrow x \in B]$.
- **Axiom of pair**: If A and B are sets, then (A, B) is also a set.
- **Axiom of subsets**: If A is a set and ϕ is a describing property, then the class of all sets in S, which satisfy this property ϕ, is a set. Moreover, every subclass of a set of sets is a set.

As previously mentioned, a set is represented by its members, which on the other hand exhibit the same properties. An average is a single member taken as representative of a set of members. This is the statement, which was used in our approach in sec. 4.1. For obtaining a representative of a class of vectors - the class (type) vector, we computed the average of the vectors of all members of a class.

3. Related Work

A series of recent studies has focused on RDF type prediction. In this chapter, some related methods are presented.

A supervised hierarchical classification approach has been proposed in [10]. A hierarchy of support vector machines (SVM) classifiers is trained on the bag-of-words representation of short abstracts and categories of Wikipedia articles. The SVM models output probability distributions for a given entity over a subset of DBpedia ontology classes. The aggregated distribution is then processed with respect to the DBpedia ontology in order to make a reliable prediction of a specific type.

Another approach was proposed in [16]. SDType is a statistical heuristic link based type prediction mechanism. The algorithm assigns types based on ingoing properties of a given instance. For each property p the SDType algorithm computes the probability that a specific entity e is of certain type, if e appears as a subject in a fact with the property p. In the same way, the probability is computed for the case that e in an object in a fact with the property p. Further, each property p is assigned a weight, which reflects its capability of predicting the type.

The dataset generated by SDType contains multiple types assigned to a given entity, but a deeper observation of the types shows that they are mostly types that belong to the same type-path. Nevertheless, SDType is the current state-of-the-art type inference algorithm. The dataset generated by it is a part of the DBpedia 2016-10 version[1] - *Instance Types Sdtyped Dbo*.

[1] `https://wiki.dbpedia.org/downloads-2016-10`

4. Approach

In this chapter, the approach developed in this thesis is presented. First, the unsupervised approach vector similarity is thoroughly described and after that, the supervised approach using a convolutional neural network model.

Both approaches take into account the structure of the type hierarchy. Fine-grained entity typing requires more attention to be paid to specific types (which are lower in the type hierarchy), because they are much more informative. Let's take the example of the entity `dbr:Arnold_Schwarzenegger`, which has the type `dbo:OfficeHolder`. This means that `dbr:Arnold_Schwarzenegger` can be associated with the following type-path: `/Thing/Agent/Person/OfficeHolder`, having the fine-grained type `dbo:OfficeHolder` and a general coarse-grained type - `dbo:Person`. In the course of this thesis, the type `dbo:Agent` is ignored and its descendands (e.g., `dbo:Person` and `dbo:Organisation`) are used as top-level types instead.

Going back to the example with Arnold Schwarzenegger - according to the Wikipedia page about him, he is not only office holder (politician), but also actor, filmmaker, businessman, author, and bodybuilder. All those fine-grained types about the entity Arnold Schwarzenegger are missing in the DBpedia knowledge graph. Our approach tries to predict them as taking into account the entity `dbr:Arnold_Schwarzenegger` and computing the vector similarity between the entity vector and the class vectors of all subclasses of `dbo:Person`, which is the top-level class in the type hierarchy, beginning from the current type - `dbo:OfficeHolder`. The approach does not consider the similarity between the entity `dbr:Arnold_Schwarzenegger` and the class `dbo:Settlement` e.g., as an entity can not be a person and a settlement at the same time. On the contrary, an entity can be an office holder and a bodybuilder simultaneously, as both of them describe a person.

4.1. Vector Similarity

The basic idea of this approach is to take an entity first and derive its vector representation from the pre-trained RDF2Vec dataset. Next, the current type of this entity is retrieved, using the DBpedia Instance Types dataset. After that, the top-level type in the type-path of the current type is determined, using recursive SPARQL queries. Finally, all subclasses of the top-level class of the current class are determined. After this process, the cosine

similarity between the entity vector and the vectors of all subclasses is computed, in order to find the most similar types to the given entity.

The approach was explored with two alternatives for the class vectors. The first takes advantage of the pre-trained RDF2Vec vectors of the classes. The second computes the class vectors using the rules of the Set Theory. Both alternatives use the pr-trained RDF2Vec vectors of the entities.

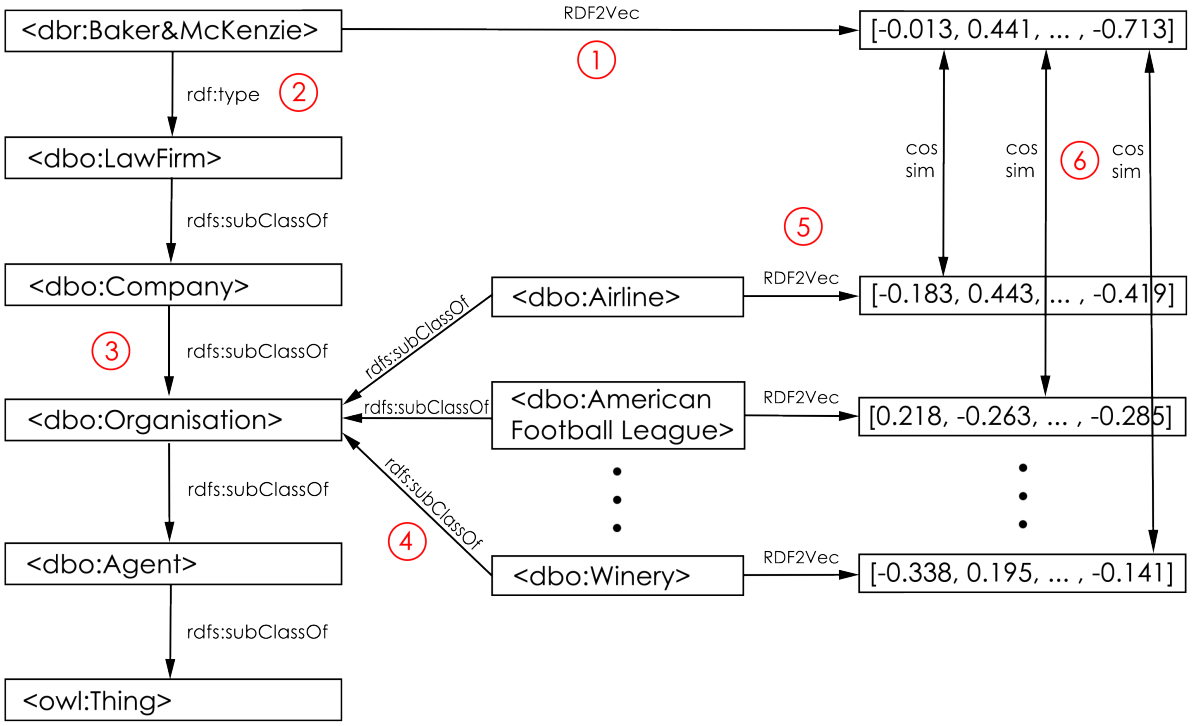


Figure 4.1.: Vector Similarity Approach.

Vector Similarity using pre-defined RDF2Vec vectors of the entities, as well as of the classes

The approach, as already described above, is furthermore presented graphically in fig. 4.1. The steps are denoted in the figure and are as follows:

Step 1: For a given entity - a 200-dimensional vector is retrieved from the RDF2Vec pre-trained knowledge graph embedding.

Step 2: The current type of the entity is found.

Step 3: The type-path of this type is found and the top-level type is determined.

Step 4: All subclasses of the top-level type are found.

Step 5: A 200-dimensional vector of each of the subclasses is retrieved from the RDF2Vec pre-trained knowledge graph embedding.

Step 6: The similarity between the entity and the vectors of the subclasses is determined by computing the cosine similarity between the corresponding vectors.

In the example of fig. 4.1 for the entity `dbr:Baker&McKenzie` the top-level type `dbo:Organisation` is selected. As previously mentioned, in the course of this thesis, the type `dbo:Agent` is ignored and its descendands (e.g., `dbo:Person` and `dbo:Organisation`) are used as top-level types instead.

Vector Similarity using pre-defined RDF2Vec vectors of the entities, but vectors of the classes - generated using the Set Theory

This approach is similar to the one described above. The only difference is the step before the computation of the cosine similarity (step 5) - the class vectors from RDF2Vec are not used. Instead, a class vector is generated, using the Set Theory (sec. 2.4).

The approach of generating a class vector is described as follows. A set is a collection of objects that share some common properties. Therefore, a set is represented by the objects in it, namely its members. An average is a single value taken as representative of a set of values. In this approach this way of thinking is used for obtaining a representative of a class of vectors - the class (type) vector. The average of the vectors of all entities of a given class is computed and the resulting vector is used for further computations as a class vector. The following formula is used, where n ist the count of entities of a given type and the $v_i, i \in [1, n]$ are the RDF2Vec vectors of all entities of the class:

$$mean = \frac{1}{n}(v_1 + v_2 + ... + v_n).$$

An example of the calculation of the class vector of `dbo:Library` is provided in fig. 4.2. The mean of a set of vectors is calculated component-wise. In other words, for the

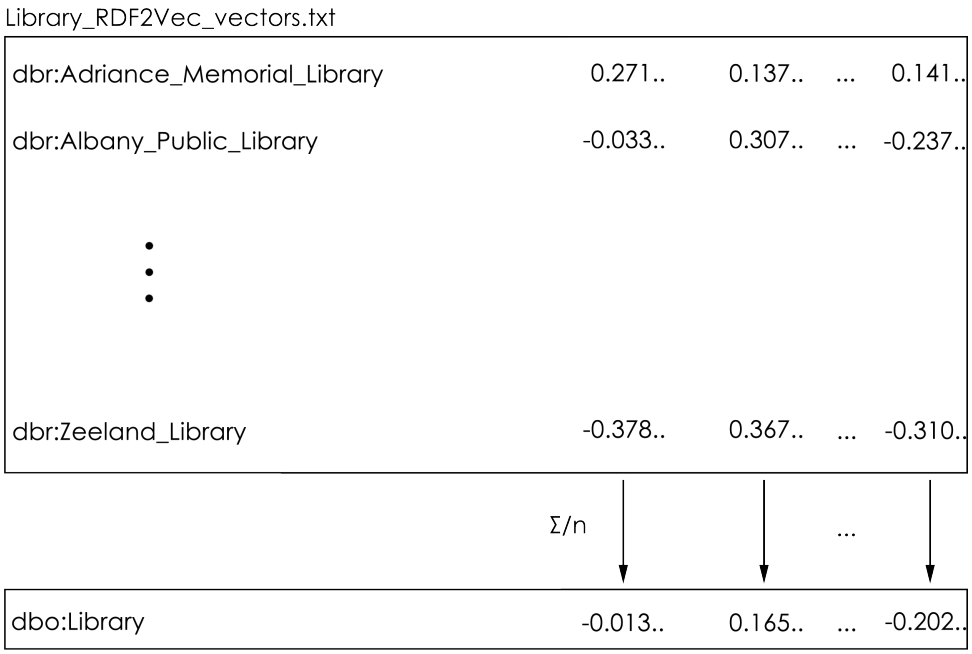


Figure 4.2.: Generation of the class vector of `dbo:Library`.

200-dimensional RDF2Vec vectors the mean of the first coordinates, than the mean of the second coordinates and so on is computed. The found coordinates are the coordinates of the mean vector. The vectors used for the computation are the RDF2Vec vectors of all entities of the given class.

The RDF2Vec model (sec. 2.2.1.1) projects the entities into a lower-dimensional feature space and semantically similar entities are positioned close to each other. Thereby, the cosine similarity values of each entity are sorted in a descending order, resulting in an ordered list of potentially types for that entity. The most similar types to a given entity are placed first in the list.

4.2. 1D Convolutional Neural Network

The 1D CNN model built on top of RDF2Vec is presented here. The architecture is similar to the one described in [9], with only one channel. We implemented and trained the model for classification of entities in multiple classes. 1D CNNs differ from 2D CNNs in the dimensionality of the input data and in the way the feature detector (filter) slides across the data. 1D CNNs are used in natural language processing, as one word (entity) is represented as a vector. The filter covers the whole vector of a word and its height determines how many words are considered at a time. The sliding of the filter is performed in only 1 direction.

The *input data* consists of the entity vectors from the RDF2Vec pre-trained model. They are of strict dimension, namely 200-dimensional.

Convolutional Layer (Conv1D)

The input data is fed into the *convolutional layer*. The convolution operation involves a filter (also called feature detector) , which is applied to a window of h entities (called kernel size) to produce a new feature [9]. We set the number of filters to 128 and the kernel size to 3. This allows us to train 128 different features on the first layer of the network. The output matrix holds the weights of one single filter and is called a feature map. The feature map then goes through a non-linear activation function, such as ReLU in our case.

Pooling Layer (GlobalMaxPooling1D)

A *pooling layer* is used after the 1D CNN layer. We apply a global max pooling operation over the feature map which finds the maximum value of each filter. The idea behind the pooling is to capture the most important feature, namely the one with the highest value for each feature map [9]. The pooling also prevents overfitting of the data. We chose a global pooling, which means that the pool size is equal to the size of the input.

Regularization (Dropout)

For regularization, we use *dropout* on the output of the pooling layer. Dropout randomly sets to zero some proportion of the hidden units during forward propagation. We chose the dropout rate of 0.2.

Fully-connected Layer (Dense)

The final layer is a *fully-connected layer*, which reduces the size of the vector to a vector of size equal to the number of classes that we want to predict. Softmax is used as an activation function. The output is the probability distribution over the labels (in our case - the entity classes).

After constructing the model, it was trained in 100 epochs with a batch size of 128, in order to produce predictions for a set of entities. We experimented with two models - the first one using two Conv1D layers and the second one with only one Conv1D layer.

5. Evaluation

This chapter describes the dataset, gives an insight into how the data is processed and finally presents the results. For computing the cosine similarity, python scikit-learn[1] library has been used. For the CNN model, Keras[2] - an API used on top of the machine-learning framework TensorFlow[3], has been used. The input files for the CNN model are loaded using pandas[4], numpy[5] is used for the representation of vectors and matrices.

5.1. Dataset

The background data, used in the course of this thesis, is the *Instance Types* dataset of DBpedia from the following release - DBpedia 2016-10 [6]. The RDF triples in the dataset are produced from DBpedia's mapping-based extraction of Wikipedia. A mapping assigns a type from DBpedia's ontology to the entities that are described by the corresponding infobox in Wikipedia [12]. The dataset contains entity-type assignments in the form of facts, namely RDF triples, which have the following structure:

```
<$resource> rdf:type <$dbpedia_ontology_class> .
```

An example of one fact in the dataset is the following triple that states that the entity Bruce Lee is of type Actor.

```
<http://dbpedia.org/resource/Bruce_Lee>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/Actor> .
```

The type of an entity in the *Instance Types* dataset is normally the most specific type. As the types are structured in an hierarchical way, the other types of the entity can be derived by going up the hierarchy type-path, until its root - owl:Thing, is reached. Considering the example with the entity Bruce Lee - it has the `rdf:type dbo:Actor`, which means

[1] https://scikit-learn.org/stable/
[2] https://keras.io/
[3] https://www.tensorflow.org/
[4] https://pandas.pydata.org/
[5] https://numpy.org/
[6] https://wiki.dbpedia.org/downloads-2016-10

that it is assotiated with the type-path: `"/Thing/Agent/Person/Artist/Actor"`, meaning that Bruce Lee belongs to all the classes in the type-path.

Distribution of entities in classes

In order to make statistics about the distribution of the entities in classes and to enable further analysis using the *Instance Types* dataset, the dataset was separated in parts, each part containing entities of only one type. By doing this, it was also found that the entities in the DBpedia's dataset are divided into 416 classes.

Type Hierarchy

The DBpedia ontology is also taken into consideration. Making use of recursive SPARQL queries and the RDF predicate `rdfs:subClassOf`, the structure of the types is found. The type hierarchy of DBpedia has a tree structure of height 7 and contains 760 classes. The difference between the count of the classes in the *Instance Types* dataset (416) and in the DBpedia ontology (760) originates from the fact, that not all types, defined as such in the ontology, are used. For example, `dbo:Capital` is defined as a subclass of `dbo:City` in the DBpedia ontology, but there are no instances of that type. The distribution of the classes in 7 levels of the type hierarchy is as follows:

Level 1: 50 classes
Level 2: 128 classes
Level 3: 210 classes
Level 4: 273 classes
Level 5: 73 classes
Level 6: 23 classes
Level 7: 4 classes.

Selected Classes

Using both the statistics about the distribution of entities in types and the type hierarchy, 59 classes were selected, with which the analysis in the thesis is conducted. A list of those classes is presented in app. A.1. The classes have the following distribution:

- 15 less popular classes (less than 500 entities)
- 20 having between 500 and 1000 entities
- 24 having more than 1000 entities

Entities

For each of the 59 chosen classes, 500 entities were extracted. From the less popular classes all entities were taken, as they are less than 500. For this step, the previously mentioned separation of the dataset in groups, based on their types, was crucial.

RDF2Vec vectors

The vectors generated by the RDF2Vec embedding model (sec. 2.2.1.1) are available for download [7] and the version with 200-dimensional uniform vectors is used in the course of this thesis. The vectors are generated with 4 hops. For each entity in the graph, the dataset contains a row with the entity name and the embedded vector. This file is further manipulated for the needs of the analysis in the thesis, namely for extracting the vector representation of certain entities and types.

5.2. Experimental Setup and Results

Vector Similarity

The vector similarity approach (sec. 4.1) is applied for the selected entities (ca. 500) of all the 59 selected classes. The last step in the vector similarity approach computes the cosine similarity between an entity and the vectors of the subclasses of its top-level class. After finding the top-level class of the selected classes, it was found that all selected classes have one of the following top-level classes: Person, Organisation or Place. This is the reason, the results are presented in those three groups, in order to be more clearly visible.

After finding the cosine similarity values, the hits@1 and hits@3 types were determined and compared to the current types of the entities in the *Instance Types* dataset. The result is presented in bar charts.

Vector Similarity using pre-defined RDF2Vec vectors of the entities, as well as of the classes

The result of the evaluation of this approach is presented in the bar charts in fig. 5.1 (Hits@1 Types) and fig. 5.2 (Hits@3 Types). The vertical axis shows the type, for the entities of which the vector similarity approach was conducted. The blue part of each bar is the part of the entities of the given type, for which a match among the hits@1, respectively hits@3 types was found.

An interesting point is in what degree the first and third cosine similarity value found are related to each other. The degree of dependence between the two values is captured by the the statistical measure correlation. The correlation for each class is given in sec. A.2. Further, scatterplots are created to represent the correlation between the two variables in the case of the following classes - Actor, AdultActor, Airline, Artist, Athlete (sec. A.2).

Vector Similarity using pre-defined RDF2Vec vectors of the entities, but vectors of the classes - generated using the Set Theory

[7] https://zenodo.org/record/1320211#.Xbnwf25FydI

The outcome of this approach was evaluated in the same way as described above. The result is presented in the bar charts in fig. 5.3 (Hits@1 Types) and fig. 5.4 (Hits@3 Types). The blue part of each bar is again the part of the entities of the given type, for which a match among the hits@1, respectively hits@3 types was found.

It can be seen, that using the vectors generated by taking the average of the RDF2Vec vectors of the elements of the given class (set theory) produced much better results than using the pre-trained RDF2Vec vectors of the classes. It can be claimed that the approach using the generated vectors succeeded producing the correct hits@1 class in almost all the cases.

As observed from the experiments, the RDF2Vec pre-trained class vectors do not reflect the characteristics of the entities of the class. This is due to the fact that RDF2Vec is path dependent and considers only the outgoing edges in the RDF graph (which is a DAG), meaning it takes into account only paths in direction from an entity vertex to a type vertex. In contrast, the class vectors generated by computing the average of the vectors of the elements of a given class are able of reflecting the characteristics of a class, as they are regarded as the mean of the members of a set (sec. 2.4).

CNN Model

As described in 4.2, first one CNN model was created, using two 1D CNN layers. It was trained with 120 entities (which were correctly typed by the vector similarity approach using only RDF2Vec vectors) and was tested on 30 other entities (80% training dataset/20% testing dataset). This experiment was done using 20 classes (the once that had most correctly typed entities).

Since the number of entities per class was small, the model overfits with two layers of 1D convolutional neural network yielding 93.5% accuracy. There are two basic ways of reducing overfitting in neural network models: by training the network on more data or by changing the complexity of the network.

In this work, both approaches have been tried. In the second experiment, 59 classes with 500 entities from each class have been considered, out of which 80% are used for training the model and 20% for testing, which results in 81.78% accuracy. Also, we tried to reduce the complexity of the model to only one 1D convolutional layer which results in 42.61% accuracy for 59 classes (table 5.1) . It has been observed from the result that for this task, reducing the complexity of the model works better for the smaller datasets.

CNN Model	Less classes and entities	All classes and entities
One 1D layer	59.83%	42.61%
Two 1D layers	93.5%	81.78%

Table 5.1.: Accuracy of 1D CNN models.

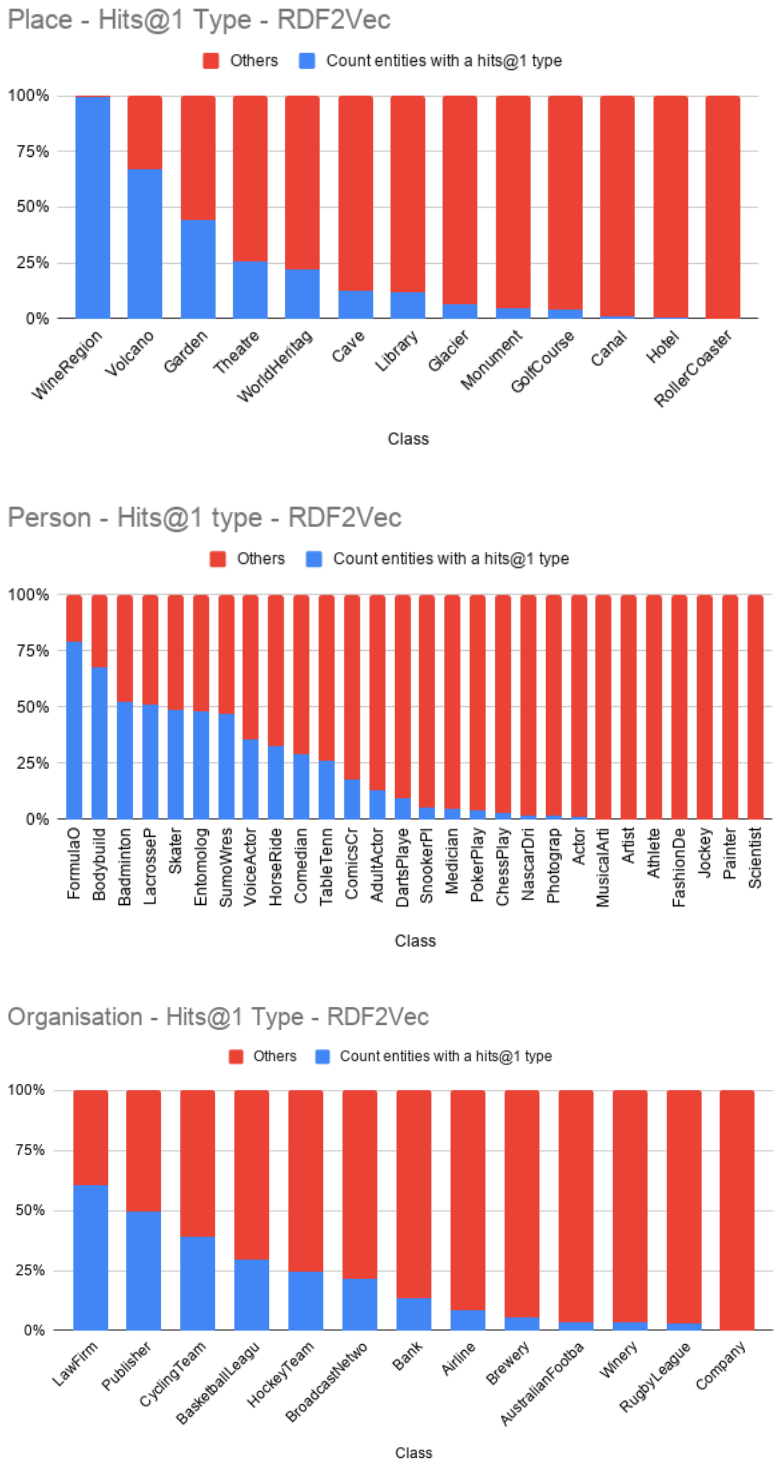


Figure 5.1.: Hits@1 - RDF2Vec

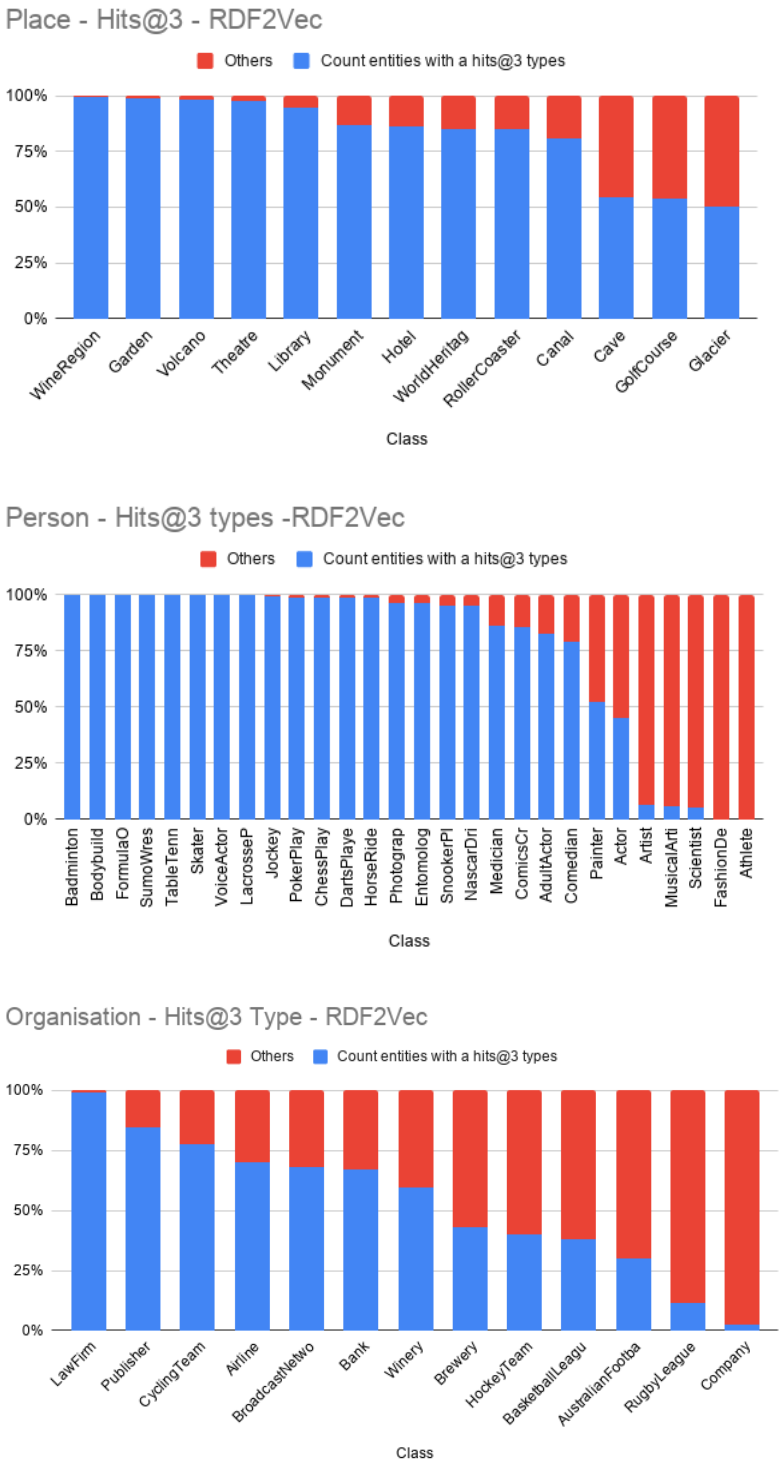


Figure 5.2.: Hits@3 - RDF2Vec

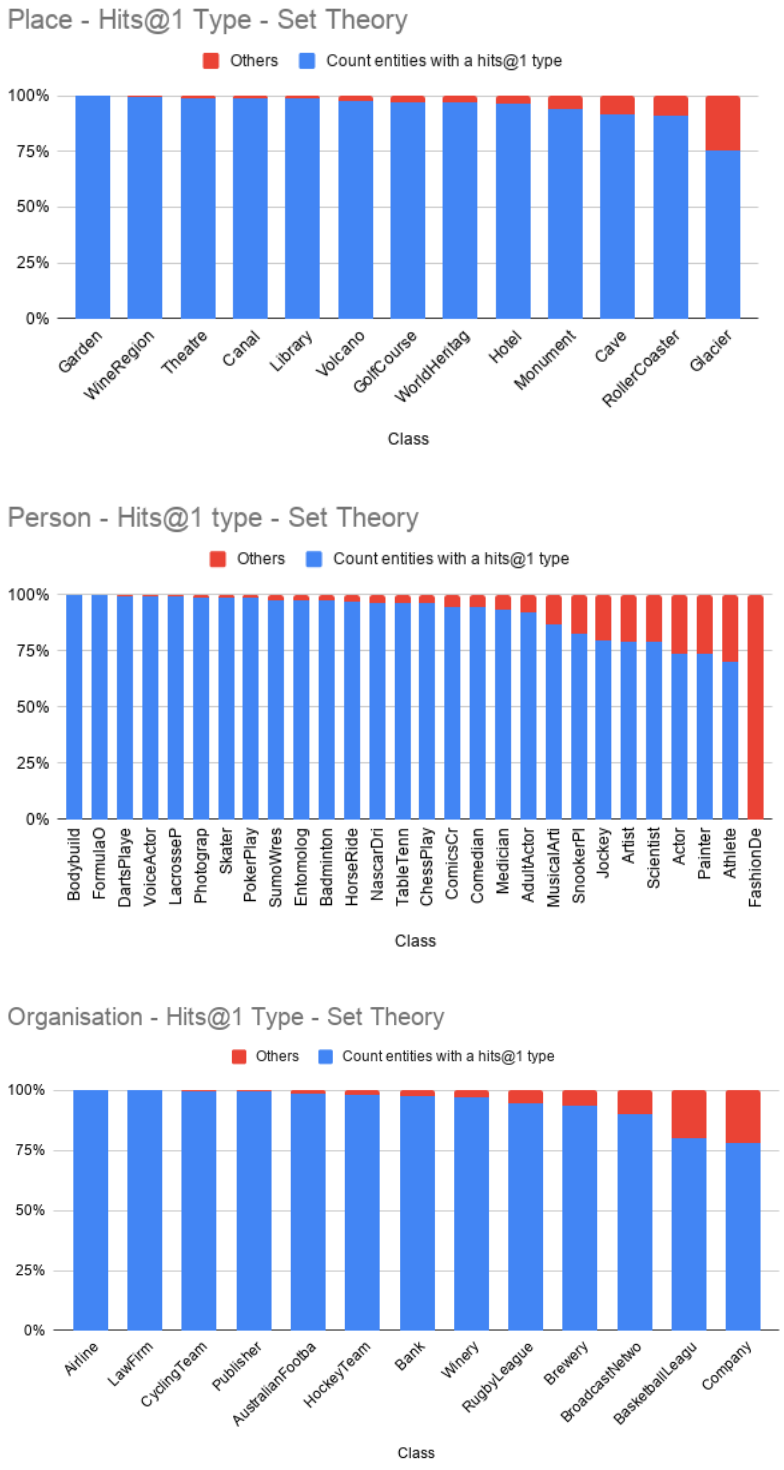


Figure 5.3.: Hits@1 - SetTheory

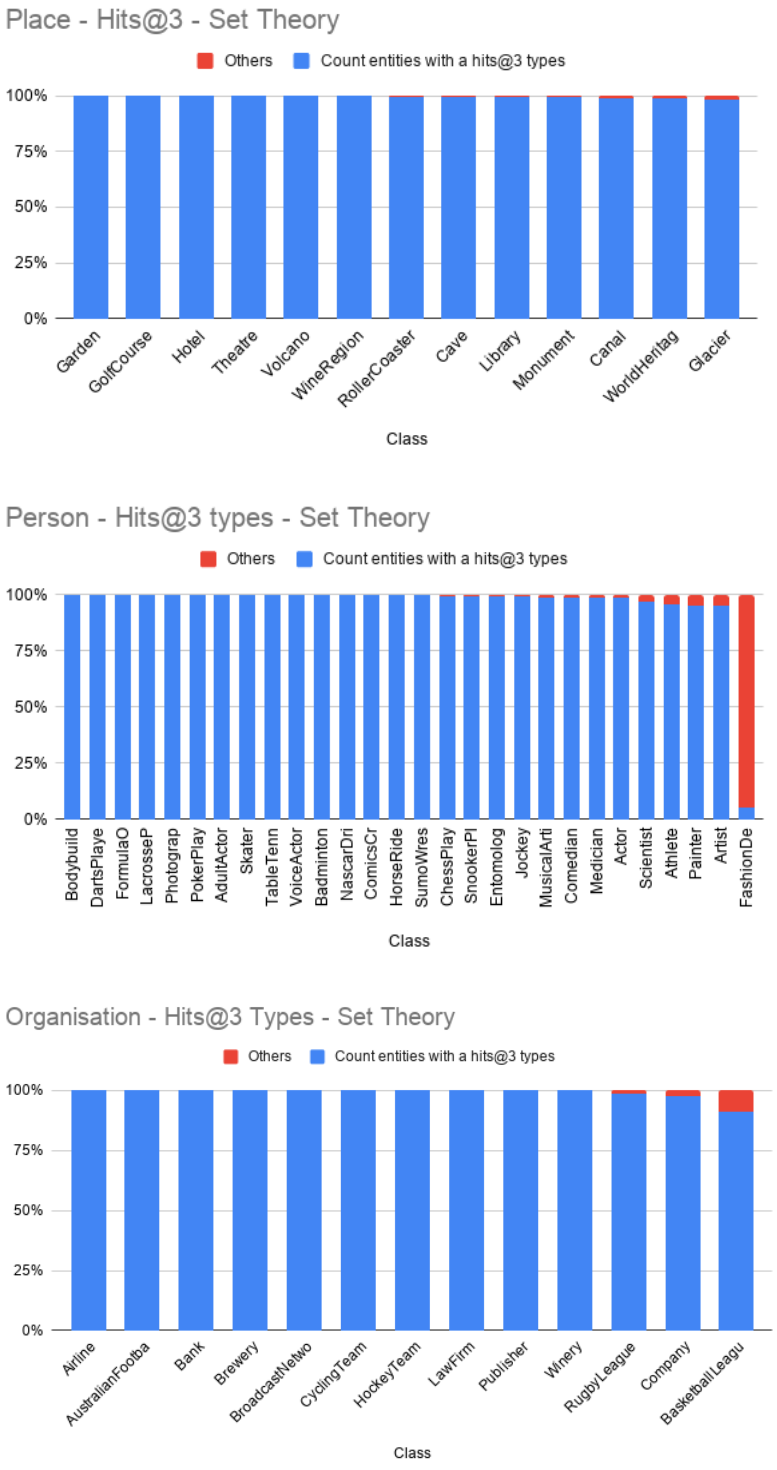


Figure 5.4.: Hits@3 - SetTheory

6. Conclusion

6.1. Summary

In this work, the problem of predicting type information of entities in a knowledge base has been addressed, particularly DBpedia. The importance of knowledge bases in representing information in a structured manner was introduces. Furthermore, some related methods were described and investigated.

In order to predict entity types, first the unsupervised approach vector similarity was explored. This method takes advantage of the RDF2Vec embedding model for the vector representation of the entities and the classes. The evaluated results show that the use of pre-defined RDF2Vec vectors for the classes is not beneficial in predicting the entity types. Therefore, new class vectors were generated, following the example of the set theory. In this way, almost all entities were assigned with the correct type. Moreover, a classification task has been performed using one supervised approach, namely a 1D convolutional neural network model.

6.2. Future Work

There are plenty of future directions of the current work. Some of those include:

- Performing 1D CNN for all the classes in DBpedia with more number of entities per class.
- Make use of the vectors from the graph kernel method of RDF2Vec for both cosine similarity and CNN.
- As observed from the experiments, the class vectors from RDF2Vec do not reflect the characteristics of the entities of the class because RDF2Vec generates paths and considers only the outgoing edges from the vertices of the entities. Retraining the RDF2Vec model by applying inverse relation from the classes to the entities in the class and performing the classification task, in order to find if this strategy would improve the results, is an interesting aspect which could be further researched.

Bibliography

[1] Remzi Celebi Ammar Ammar. In: *Proceedings of the Cyber-Physical Systems PhD Workshop 2019*. URL: http://ceur-ws.org/Vol-2456/paper33.pdf.

[2] Robert Andre. *Axioms and Set Theory*. 2014. ISBN: 978-0-9938485-0-6.

[3] Krisztian Balog. *Entity-Oriented Search*. Vol. 39. The Information Retrieval Series. Springer, 2018. ISBN: 978-3-319-93933-9. DOI: 10.1007/978-3-319-93935-3. URL: https://eos-book.org.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". In: *Scientific American* 284.5 (May 2001), pp. 34–43. URL: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[5] Russa Biswas, Maria Koutraki, and Harald Sack. "Exploiting Equivalence to Infer Type Subsumption in Linked Graphs." In: *ESWC (Satellite Events)*. Ed. by Aldo Gangemi et al. Vol. 11155. Lecture Notes in Computer Science. Springer, 2018, pp. 72–76. ISBN: 978-3-319-98192-5. URL: http://dblp.uni-trier.de/db/conf/esws/eswc2018s.html#BiswasKS18.

[6] Michael Cochez et al. "Biased Graph Walks for RDF Graph Embeddings". In: *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*. WIMS '17. Amantea, Italy: ACM, 2017, 21:1–21:12. ISBN: 978-1-4503-5225-3. DOI: 10.1145/3102254.3102279. URL: http://doi.acm.org/10.1145/3102254.3102279.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[8] Uday Kamath. *Deep Learning for NLP and Speech Recognition*. Ed. by John Liu and James Whitaker. Cham, 2019. URL: https://doi.org/10.1007/978-3-030-14596-5.

[9] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 2014, pp. 1746–1751. URL: http://aclweb.org/anthology/D/D14/D14-1181.pdf.

[10] Tomás Kliegr and Ondrej Sváb-Zamazal. "LHD 2.0: A text mining approach to typing entities in knowledge graphs". In: *J. Web Semant.* 39 (2016), pp. 47–61.

[11] Yann LeCun and Yoshua Bengio. "The Handbook of Brain Theory and Neural Networks". In: ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. ISBN: 0-262-51102-9. URL: http://dl.acm.org/citation.cfm?id=303568.303704.

[12] Jens Lehmann et al. “DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia”. In: *Semantic Web Journal* 6.2 (2015), pp. 167–195. URL: http://jens-lehmann.org/files/2015/swj_dbpedia.pdf.

[13] Pablo N. Mendes, Max Jakob, and Christian Bizer. “DBpedia for NLP: A Multilingual Cross-domain Knowledge Base”. In: *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC’12)*. Ed. by Nicoletta Calzolari (Conference Chair) et al. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012. ISBN: 978-2-9517408-7-7.

[14] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119. URL: http://dl.acm.org/citation.cfm?id=2999792.2999959.

[15] Jojo Moolayil. *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*. 1st. Berkely, CA, USA: Apress, 2018.

[16] Heiko Paulheim and Christian Bizer. “Type Inference on Noisy RDF Data”. In: *The Semantic Web – ISWC 2013*. Ed. by Harith Alani et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 510–525. ISBN: 978-3-642-41335-3.

[17] Petar Ristoski and Heiko Paulheim. “RDF2Vec: RDF Graph Embeddings for Data Mining”. In: *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*. 2016, pp. 498–514. DOI: 10.1007/978-3-319-46523-4_30. URL: https://doi.org/10.1007/978-3-319-46523-4%5C_30.

[18] Amit Singhal. “Modern Information Retrieval: A Brief Overview”. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 24 (2001), pp. 35–43.

[19] Liyang Yu. *A Developer’s Guide to the Semantic Web*. SpringerLink. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. ISBN: 9783642159701.

A. Appendix

A.1. List of 59 classes used in the scope of the thesis

1. Artist
    a) Comedian
    b) Actor
    c) Fashion Designer
    d) Photographer
    e) Painter
    f) Comics Creator
    g) Adult Actor
    h) Voice Actor
    i) Musical Artist
2. Athlete
    a) Snooker Player
    b) Body Builder
    c) Sumo Wrestler
    d) Lacrosse Player
    e) Table Tennis Player
    f) Skater
    g) Jockey
    h) Darts Player
    i) Poker Player
    j) Formula One Racer
    k) Horse Rider
    l) Nascar driver
    m) Badminton Player
    n) Chess Player

3. Scientist
    a) Medician
    b) Entomologist
4. Organisation
    a) Cycling Team
    b) Australian Football team
    c) Law firm
    d) Winery
    e) Rugby League
    f) Basketball league
    g) Broadcast network
    h) Library
    i) Publisher
    j) Hockey Team
5. Company
    a) Airline
    b) Brewery
    c) Bank
6. Place
    a) Garden
    b) Wine Region
    c) Monument
    d) Canal
    e) Glacier
    f) Volcano
    g) Golf Course
    h) Theatre
    i) Roller Coaster
    j) Cave
    k) World Heritage Site
    l) Hotel
7. Language
    a) Programming Language
8. Genre
    a) Musical Genre

A.2. Correlation

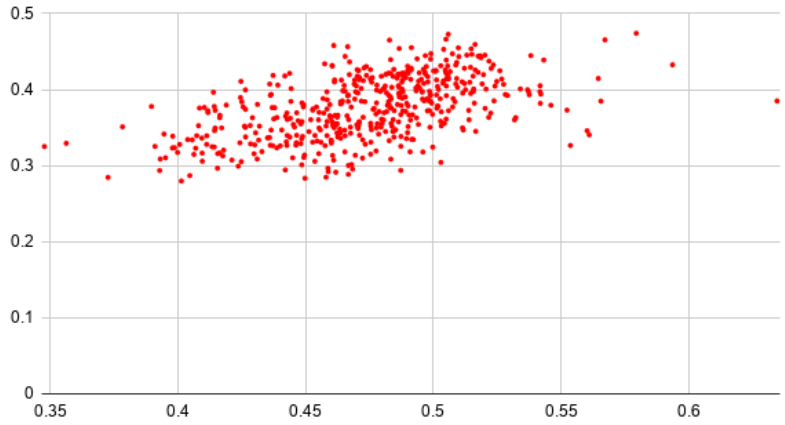


Figure A.1.: Correlation - Actor.

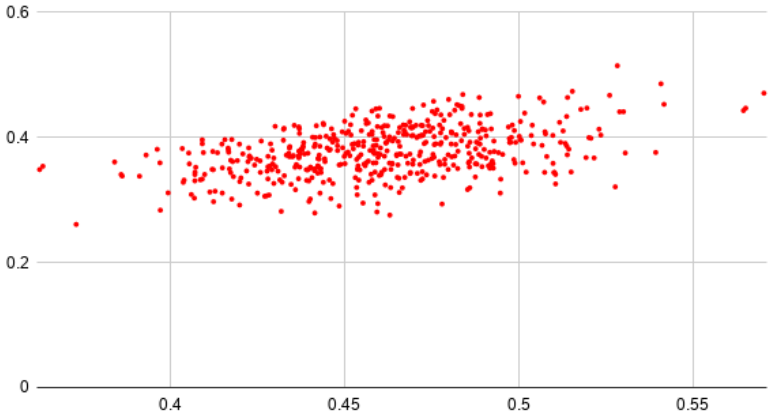


Figure A.2.: Correlation - AdultActor.

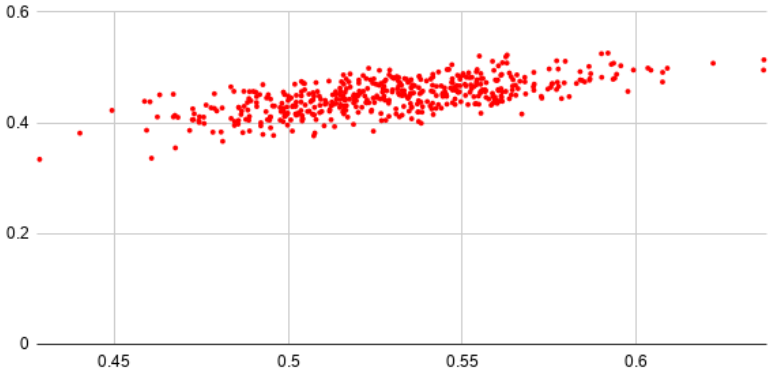


Figure A.3.: Correlation - Airline.

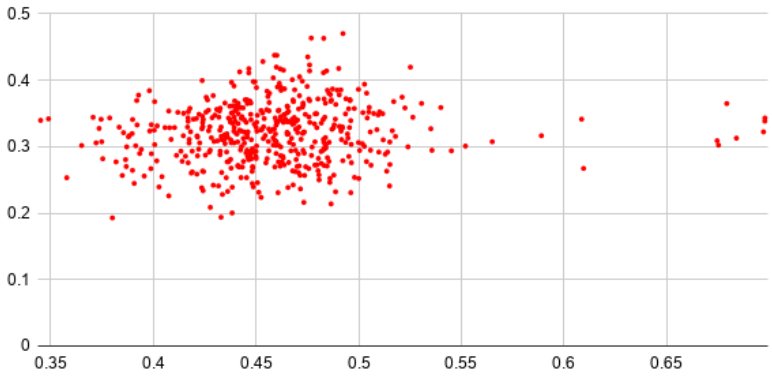


Figure A.4.: Correlation - Artist.

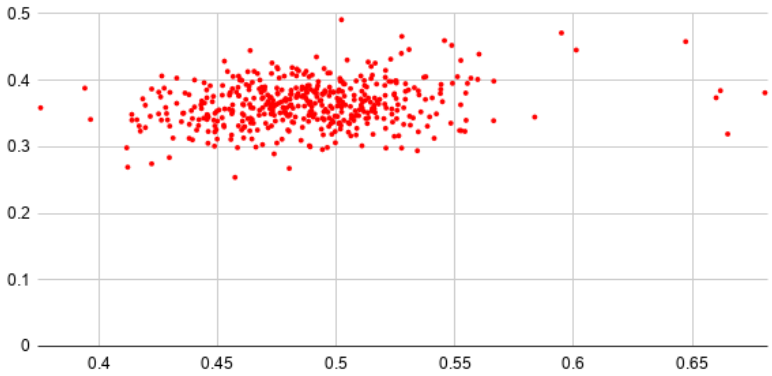


Figure A.5.: Correlation - Athlete.

Class	Correlation
Actor	0.529456
AdultActor	0.442402
Airline	0.663913
Artist	0.10312
Athlete	0.222505
AustralianFootballTeam	0.648803
BadmintonPlayer	0.549596
Bank	0.630473
BasketballLeague	0.596523
Bodybuilder	0.737294
Brewery	0.626912
BroadcastNetwork	0.644686
Canal	0.703279
Cave	0.734999
ChessPlayer	0.324538
Comedian	0.363572
ComicsCreator	0.537698
Company	0.831974
CyclingTeam	0.150057
DartsPlayer	0.336342
Entomologist	0.517296
FashionDesigner	0.529456
FormulaOneRacer	0.651314
Garden	0.448983
Glacier	0.757058
GolfCourse	0.682011
HockeyTeam	0.752062
HorseRider	0.642217
Hotel	0.473888
Jockey	0.389885

Table A.1.: Correlations - Part 1.

Class	Correlation
LacrossePlayer	0.526022
Language	-
LawFirm	0.397312
Library	0.489951
Medician	0.333219
Monument	0.49175
MusicalArtist	0.550251
MusicGenre	0.382212
NascarDriver	0.15976
Organisation	0.715338
Painter	0.400472
Photographer	0.612491
Place	0.721288
PokerPlayer	0.353083
ProgrammingLanguage	-
Publisher	0.28252
RollerCoaster	0.818314
RugbyLeague	0.772606
Scientist	0.145496
Skater	0.494113
SnookerPlayer	0.617371
SumoWrestler	0.640482
TableTennisPlayer	0.593807
Theatre	0.372384
VoiceActor	0.642889
Volcano	0.526229
WineRegion	0.617912
Winery	0.69275
WorldHeritageSite	0.416674

Table A.2.: Correlations - Part 2.