UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Enrichment of ontological taxonomies using a neural network approach

Bachelorarbeit

zur Erlangung des Grades einer Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von
Alex Baier

Erstgutachter: Prof. Dr. Steffen Staab
Institute for Web Science and Technologies

Zweitgutachter: Dr. Mahdi Bohlouli
Institute for Web Science and Technologies

Koblenz, im April 2017

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐
Der Text dieser Arbeit ist unter einer Creative Commons Lizenz verfügbar.	☐	☐
Der Quellcode ist unter einer Creative Commons Lizenz verfügbar.	☐	☐
Die erhobenen Daten sind unter einer Creative Commons Lizenz verfügbar.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Contents

List of Figures

1 Introduction

Knowledge bases (KB) like DBpedia, YAGO and Wikidata store information about every significant real-world entity and the relations between those entities. Using different approaches like human collaboration in Wikidata or information from Wikipedia data boxes in YAGO, these KBs are populated [16]. The issue of incompleteness arises in this context, as it is not realistic to assume that every bit of knowledge can or should be captured in the KBs. **Ontologies** are necessary for modeling, sharing, and exchanging knowledge [19], and are therefore integral in the functioning of a KB [43]. They provide data schemas, concept vocabularies and rules [29] to which a KB has to adhere. Therefore, it should be a priority to ensure the completeness of the ontology in a KB. Types of ontologies range from lightweight ontologies like glossaries and taxonomies (concept hierarchies) to heavyweight ontologies like data models and description logics [43].

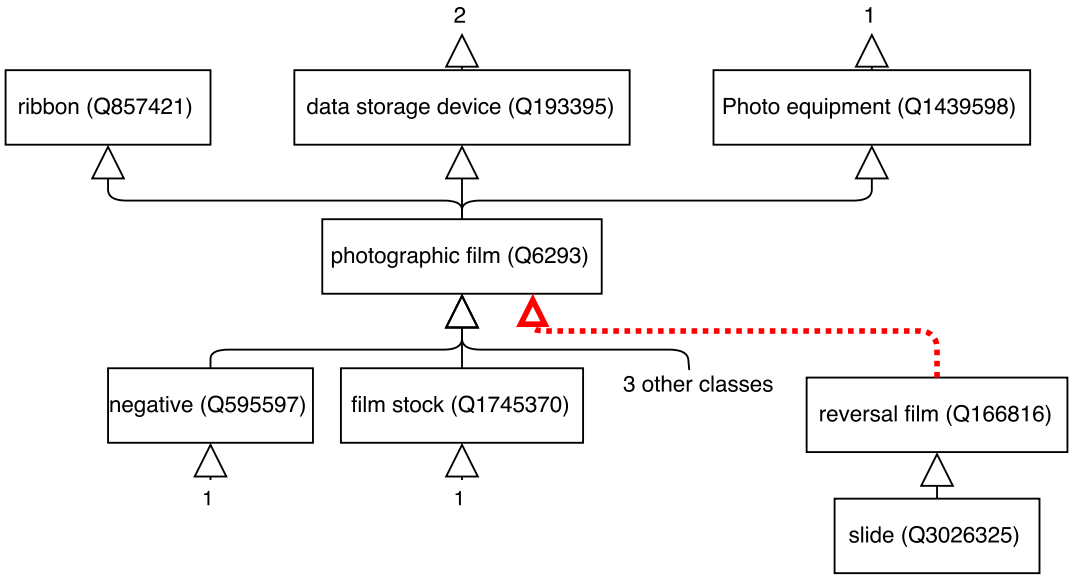

Figure 1: Classification of orphan class in Wikidata's taxonomy

The thesis' goal is the development and evaluation of approaches to increase the completeness of ontological taxonomies. Specifically, the problem of classifying **orphan classes** is solved. Orphan classes are classes in a taxonomy, which have no superclasses (parents) and are not the root of the taxonomy. An example, for such a classification problem is given in Figure 1. An extract of the Wikidata taxonomy is shown. The class *reversal film* is an orphan class in the taxonomy, since it is has no superclass. A possible solution is represented by the dotted arrow, which places *photographic film* as superclass of *reversal film*. It has to be noted that additionally all superclasses of *photographic film* are also valid solutions to the problem for the orphan class *reversal film*, since the subclass-of relationship is transitive. A solution

to the problem should find the most specific superclass for a given orphan class [12]. An algorithm is developed with the task of classifying such orphan classes into a given taxonomy. The algorithm will use neural word embeddings, as developed by Mikolov et al. [30]. This is motivated by the effectiveness of neural networks in other context-sensitive tasks like language processing [30] [22] [5], image generation [18] etc. Neural word embeddings are especially interesting, since they have been used to great effect in the construction of taxonomies [15], graph representation [6] and semantic similarity tasks [31] [25]. Different approaches in using neural word embeddings for classification of orphan classes will be considered and evaluated in this work.

In this work, the taxonomy of Wikidata is considered as case example, as it provides a high number of entities ($24,507,102$ in 2016/11/07 [3]), and a big taxonomy consisting of $1,299,501$ classes [3]. At this time, Wikidata's taxonomy is maintained by human editors. The solution developed by the thesis should be able to support the editors in improving the completeness of the taxonomy by reducing the number of orphan classes and refining the existing taxonomic relations between classes. $16,373$ orphan classes were identified by the author in the dump of 2016/11/07 [3].
TODO: describe this more

The thesis is structured as follows. In Chapter 2 the problem will be formally defined (2.1, 2.2, 2.3, 2.4) and related work in the fields of neural networks (2.6) and ontology learning (2.7) discussed for the use in this work. Section 2.6 introduces the notion of neural networks and discusses work dealing with neural word and graph embeddings. Section 2.7 compares the problem, solved by the thesis, to related work in the field of ontology learning. Subsequently, the problem is classified, solutions to similar problems analyzed and the novelty of the work justified. Chapter 3 describes the Wikidata data set used as case example for the implementation and evaluation of the proposed hybrid algorithm. Chapter 4 describes the components of the developed baseline algorithm and variations, which potentially improve the effectiveness of the method. Chapter 5 describes the evaluation methodology and presents the results. The baseline algorithm and proposed improvements are evaluated.

Formatting of text obeys the following rules. Italic text indicates Wikidata entities like *entity (Q35120)* or *instance of (P31)*. Bold text indicates the first occurrence of a relevant concept, such as **orphan class**. Such concepts are described in the following sentences and, if necessary, formally defined. Citations appear in quotation marks.

2 Background and related work

2.1 Wikidata

Wikidata is an open, free, multilingual and collaborative KB. It is a structured knowledge source for other Wikimedia projects. It tries to model the real world, meaning every concept, object, animal, person, etc. Wikidata is mostly edited and extended by humans, which in general improves the quality of entries compared to fully-automated systems, because different editors can validate and correct occurring errors. However, Wikidata, like most knowledge bases, is incomplete and therefore has to be operated under the **Open World Assumption (OWA)**. OWA states that if a statement is not contained in a knowledge base, it is not necessarily false but rather unknown [16].

In Wikidata **items** and **properties** exist. Items are the aforementioned concepts, objects, etc. Properties are used to make claims about items, e.g. *photographic film (Q6293)* is a *subclass of (P279) data storage device (Q193395)* (see Figure 2). Each item and property has an unique identifier, which starts with the letter Q for items and the letter P for properties and is followed by a numeric code. The identifiers in Wikidata are essential to avoid ambiguity and make items and properties multilingual.

Items consist of labels, aliases and descriptions in different languages. Sitelinks connect items to their corresponding pages of Wikimedia projects like Wikipedia articles. Most importantly item are described by **statements**. Statements are in their simplest form a pair of property and value, assigned to a specific item. A value is either a literal value or another item. It should be noted that an item can have multiple statements with the same property. The set of statements with the same property is called statement group. Statements can be annotated with qualifiers, which specify the context of the statement, e.g. population at a certain point of time. Additionally, references can be used for statements to include its source. See Figure 2 for an example of a Wikidata item.

Following, the terms of item and statement are defined in the context of Wikidata.

Definition 1 (Item)**.** An **item** is a tuple $(id, label, aliases, description, sitelinks)$:

- $id \in \mathbb{N}$ is the numerical item ID;
- $label \in String$ is the English label of the item;
- $aliases \in \mathcal{P}(String)$ is the set of English synonyms for the label;
- $description \in String$ is a short sentence describing the item;
- $sitelinks \in String \times String$ is a set of tuples $(site, title)$, where $site$ refers to a specific site of the Wikimedia projects, e.g. enwiki, and $title$ is the corresponding article title of the item on this site.

Definition 2 (Statement)**.** A **statement** is a tuple $(itemid, pid, value, refs, qualifiers)$:

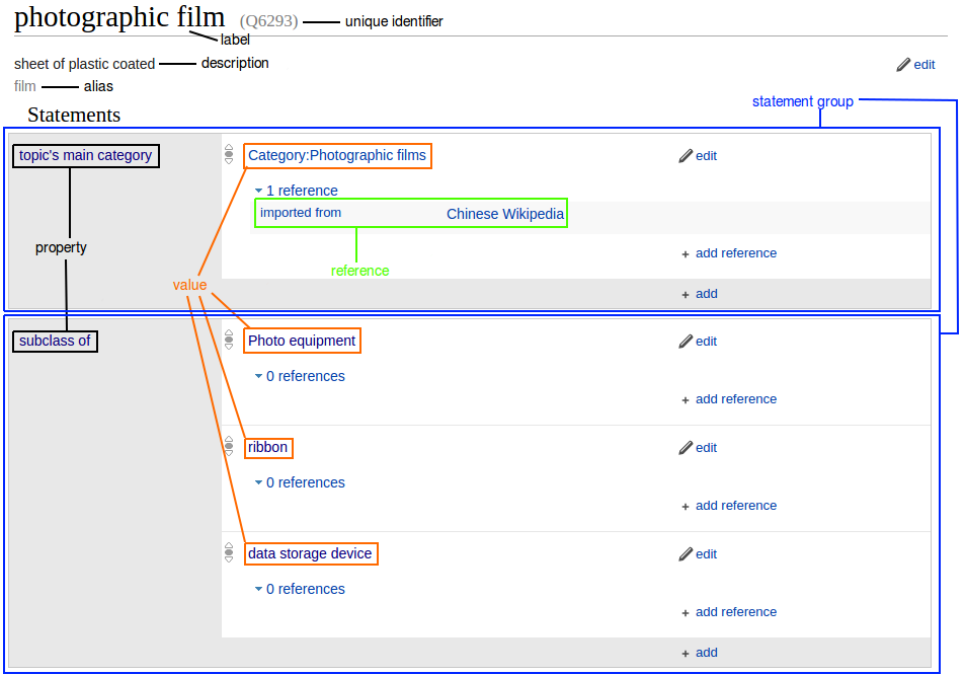

Figure 2: Example of Wikidata class: photographic film (Q6239)

- $itemid \in \mathbb{N}$ is a numerical item ID, to which the statement belongs;
- $pid \in \mathbb{N}$ is a numerical property ID;
- $value$ is either a constant value like string, int, etc., or an item ID;
- $refs$ is a set of references, containing the source of information for the statement;
- $qualifiers$ is a set of qualifiers, which further specifies the statement.

In Wikidata, there is no strict distinction between classes and instances. Both groups are represented as items. This leads to the issue, that recognizing, whether an item is a class or instance is not trivial. Based on which statements connect two items, a distinction can be made. A class is any item, which has instances, subclasses or is the subclass of another class. In Wikidata, the properties *instance of (P31)* and *subclass of (P279)* exist, which describe these relations between items. Therefore to identify whether an item is a class, it needs to be checked, whether the items fulfills any of the three mentioned criteria.

Definition 3 (Class)**.** Given a set of items I and a set of statements R. $c = (classid, \_, \_, \_, \_) \in$

I is a **class**, if at least one of the following assertions are true:

$\exists i = (instanceid, \_, \_, \_, \_) \in I \; \exists s = (itemid, pid, value, \_, \_) \in R :$
$instanceid = itemid \wedge pid = 31 \wedge value = classid$ (has instance)
$\exists s = (itemid, pid, \_, \_, \_) \in I : itemid = classid \wedge pid = 279$ (is subclass)
$\exists i = (subclassid, \_, \_, \_, \_) \in I \; \exists s = (itemid, pid, value, \_, \_) \in R :$
$itemid = subclassid \wedge pid = 279 \wedge value = classid$ (has subclass)

_ is used as an anonymous placeholder, for the purpose of not naming unused elements in tuples. For example, *photographic film (Q6293)* (Figure 2) is a class, because it is the subclass of three other classes.

2.2 Taxonomy

"Ontologies are (meta)data schemas, providing a controlled vocabulary of concepts, each with an explicitly defined and machine processable semantics" [29]. Additionally it is possible for ontologies to contain axioms used for validation and constraint enforcement. Ontologies enable the modeling and sharing of knowledge in a specific domain and support the knowledge exchange via web by extending syntactic to semantic interoperability [19]. In comparison, a KB like Wikidata can be seen as an instantiation of such an ontology, since every KB has to be conceptualized by an ontology [43]. Different types of ontologies can grouped by their level of formality and expressiveness. Wong et al. [43] differentiates ontologies as lightweight and heavyweight ontologies (see Figure 3). **Taxonomies** are concept or class hierarchies. They typically represent a parent-child structure, which can be formalized with a single relationship called for example *subclass-of* in the case of Wikidata. The observed taxonomy in Wikidata belongs to the category of lightweight ontologies, specifically principled, informal hierarchies, as the only enforced rule for the subclass-of relation is that it should connect two entities [1].

For the purpose of developing a formal definition of the thesis' problem statement the notion of taxonomy needs to be formalized. Cimiano [9] defines a heavyweight ontology, which includes a taxonomy, as follows:
"

Definition (Ontology)**.** An **ontology** is a structure

$$\mathcal{O} := (C, \leq_C, R, \sigma_R, \leq_R, \mathcal{A}, \sigma_\mathcal{A}, \mathcal{T})$$

consisting of

- four disjoint sets C, R, $\mathcal{A}$, and $\mathcal{T}$ whose elements are called concept identifiers, relation identifiers, attribute identifiers and data types, respectively,
- a semi-upper latice $\leq_C$ on C with top element $root_C$, called concept hierarchy or taxonomy,

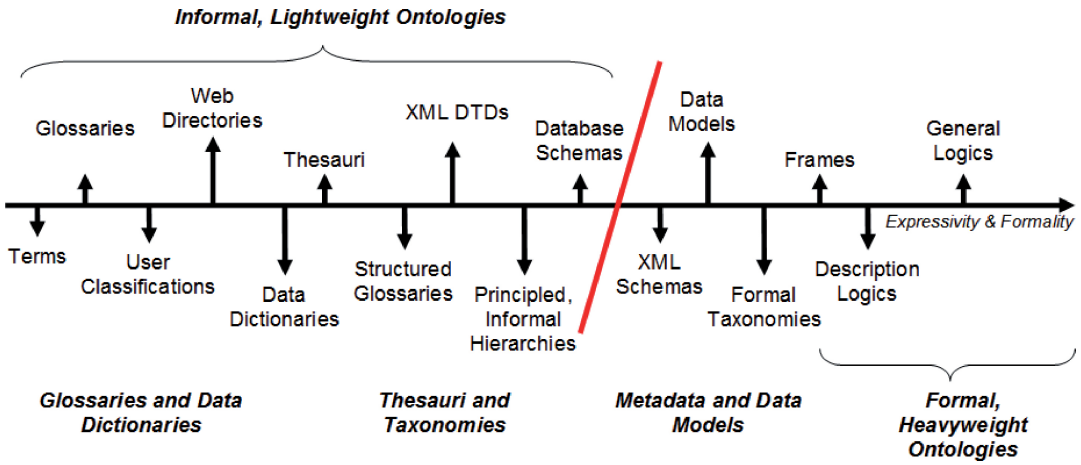

Figure 3: The spectrum of ontology kinds. [43]

- a function $\sigma_R : R \to C^+$ called relation signature,
- a partial order $\leq_R$ on R, called relation hierarchy, where $r_1 \leq_R r_2$ implies $|\sigma_R(r_1)| = |\sigma_R(r_2)|$ and $\pi_i(\sigma_R(r_1)) \leq_C \pi_i(\sigma_R(r_2))$, for each $1 \leq i \leq |\sigma_R(r_1)|$, and
- a function $\sigma_{\mathcal{A}} : \mathcal{A} \to C \times \mathcal{T}$, called attribute signature,
- a set $\mathcal{T}$ of datatypes such as strings, integers, etc.

Hereby, $\pi_i(t)$ is the i-th component of tuple t. [...] Further, a semi-upper lattice $\leq$ fulfills the following conditions:

$\forall x(x \leq x)$ (reflexive)
$\forall x \forall y(x \leq y \wedge y \leq x \implies x = y)$ (anti-symmetric)
$\forall x \forall y \forall z(x \leq y \wedge y \leq z \implies x \leq z)$ (transitive)
$\forall x x \leq top$ (top element)
$\forall x \forall y \exists z(z \geq \wedge z \geq y \wedge \forall w(w \geq x \wedge w \geq y \implies w \geq z))$ (supremum)

So every two elements have a unique most specific supremum. ”

A taxonomy can be modeled as a semi-upper lattice. This induces two important assumptions about the structure and to some degree completeness of the observed taxonomies. First, there is only one *root class*, top element of the lattice, of which every other class is (transitively) a subclass. Second, because of the supremum property, the taxonomy is fully connected, which means each class, but the root class, has a superclass. Wikidata's taxonomy does therefore not fulfill the definition by Cimiano [9], as it is not fully connected.

In the following, definitions wil bel presented, which attempt to model an incomplete taxonomy based on the already presented data model and structure of Wikidata. Refer to Appendix A for the necessary definitions on graphs.

In Wikidata, a class can have multiple superclasses, therefore a tree structure is not sufficient to model the taxonomy. However a directed acyclic graph, can model the taxonomy. The acyclic constraint is necessary to ensure that no class is transitively a subclass of itself.

Definition 4 (Taxonomy)**.** A **taxonomy** $T = (C, S)$ is a directed acyclic graph, where C is a set of class identifiers, and S is the set of edges, which describe the subclass-of relation between two classes. such that c_1 is the subclass of c_2, if $(c_1, c_2) \in S$.

Definition 5 (Subclass-of relation)**.** The transitive binary relation $\lhd_T$ on the taxonomy $T = (C, S)$ represents the subclass relationship of two classes in T. Given $c_1, c_2 \in C$, $c_1 \lhd_T c_2$, if there is a walk $W = (c_1, \ldots, c_2)$ with length $n \geq 1$, which connects c_1 and c_2. $\lhd_T$ is transitive, $\forall c_1, c_2, c_3 \in C : c_1 \lhd_T c_2 \land c_2 \lhd_T c_3 \implies c_1 \lhd_T c_3$.

If the taxonomy defined by Cimiano [9] is mapped on this graph-based taxonomy model, the following assumption is true, for $T = (C, S)$:

$$|\{c \in C \mid \neg\exists s \in C : c \lhd_T s\}| = 1 \tag{1}$$

Only one class in this taxonomy has no superclasses. This class is called **root class**. However in the case of Wikidata, this assumption does not hold true. The following state is the case:

$$|\{c \in C \mid \neg\exists s \in C : c \lhd_T s\}| > \tag{2}$$

There are classes other than the root class, which also have no superclasses. These classes will be called orphan classes.

Definition 6 (Root class)**.** Given a taxonomy $T = (C, S)$, the **root class** $root_T$ is a specific, predefined class with no superclasses in T. For $root_T$, $|succ_T(root_T)| = 0$ applies.

Definition 7 (Orphan class)**.** Given a taxonomy $T = (C, S)$ with a root class $root_T$, a class $u \in C$ is called **orphan class**, if $u \neq root_T \land |succ_T(u)| = 0$.

In Wikidata, the root class is *entity (Q35120)* [2]. All other classes, which are not subclasses of *entity (Q35120)*, are therefore either orphan classes, or subclasses of orphan classes. In Chapter 3, it is shown that 97% of all classes are subclasses of the root class *entity (Q35120)*. This set $R = \{c \in C | c \lhd_T root_T \lor c = root_T\}$ will be referred to as **root taxonomy** in later sections.

2.3 Similarity

For the task of ontology learning [19] as well as classification, e.g. k-nearest-neighbors [7], the concept of **similarity** is of importance. A basic intuition of similarity is for example given by Lin [28]. Similarity is related to the commonalities and differences between two objects. More commonalities implies higher similarity. Vice versa, more differences implies lower similarity. Two identical objects should have the maximum similarity. In addition, only identical objects should be able to achieve maximum similarity. A **similarity measure** computes the similarity between two objects [42]:

Definition 8 (Similarity measure)**.** $sim : \Omega \times \Omega \mapsto [0, 1]$ is called **similarity measure** for a set of objects Ω.

A similarity of 1 implies identical objects and a similarity of 0 implies that there are no commonalities between the input objects. The information-theoretic definition of similarity states that a similarity measure should factor in both commonalities as well as differences to compute the similarity between objects [28].

Many types of information can be encoded as feature vectors in $\mathbb{R}^n$, e.g. words [27] [30], graphs [38] [6], etc. The similarity between feature vectors can be measured using the distance $\delta_S(\vec{p}, \vec{q})$ between two vectors $\vec{p}$ and $\vec{q}$. The distance represents the differences between two feature vectors. A typical distance measure is the L_S**-Norm** [42]:

$$L_s : \delta_S(\vec{p}, \vec{q}) = \sqrt[S]{\sum_{i=0}^{n-1} |\vec{p}_i - \vec{q}_i|^S} \tag{3}$$

For example, the L_2-Norm is the euclidean distance:

$$L_2 : \delta_2(\vec{p}, \vec{q}) = \sqrt{\sum_{i=0}^{n-1} |\vec{p}_i - \vec{q}_i|^2} \tag{4}$$

Similarity between vectors can now be defined by mapping the distances to similarity values using a correspondence function [42]:

Definition 9 (Correspondence function)**.** $h : \mathbb{R}^+ \mapsto [0, 1]$ is a **correspondence function**, if it fulfills the following properties:

1. $h(0), h(\infty) = 0$,
2. $\forall x, y : x > y \implies h(x) \leq h(y)$

Aggarwal et al. [4] show that L_S distance measures especially suffer the curse of dimensionality for increasing values of S. The curse of dimensionality implies that "the ratio of the distances of the nearest and farthest neighbors to a given target

in high dimensional space is almost 1 for a wide variety of data distributions and distance functions" [4]. **Cosine similarity** can offset this problem to some degree by including the direction of the vectors into the calculation of the similarity [20]. This is done by computing the dot product between two vector. It is defined as follows:

$$sim_{cos}(\vec{p}, \vec{q}) = \frac{\vec{p}^T \cdot \vec{q}}{||\vec{p}||_2 \cdot ||\vec{q}||_2} \tag{5}$$

Cosine similarity maps to $[-1, 1]$. It returns 1 if the vectors are identical, 0 if the vectors are orthogonal, and -1 if the angle between the vectors is π and therefore the vectors are opposites. However, if the feature vector space Ω contains only positive vectors, $\forall \vec{p} \in \Omega : \vec{p_i} \geq 0 \forall i = 0, \ldots, n-1$, then the cosine similarity maps to $[0, 1]$ and fulfills the definition of a similarity measure. For example, feature vectors representing word frequencies in documents are positive, as a word can not occur less than 0 times.

In ontology learning, **semantic similarity** is used to great effect, e.g for clustering objects to create hierarchies[19] [43] or mapping between different ontologies [12] [39]. Semantic similarity compares the semantic content of objects or documents . This can be achieved by comparing which features can be found in both objects (commonalities) and which features are unique to the compared objects (differences). Rodríguez and Egenhofer [39] develops a semantic similarity measure for comparing entity classes in ontologies. Given two objects $a, b \in \Omega$. A and B are their corresponding descriptions sets, e.g. for Wikidata the aliases and statements. α is a function, which defines the importance of differences between a and b. $A \cap B$ is the set of commonalities, and A/B the set of differences between the a and b. The defined similarity function is not symmetric, $sim(a, b) \neq sim(a, b)$.

$$sim(a, b) = \frac{|A \cap B|}{|A \cap B| + \alpha(a, b)|A/B| + (1 - \alpha(a, b))|B/A|} \tag{6}$$

for $0 \leq \alpha \leq 1$.

Calculating the similarity between vectors is an easier task than calculating the similarity between objects in an ontology. Using neural word or graph embeddings, which are presented in Sections 2.6.2 and 2.6.3, enables the representation of classes and instances in ontologies as feature vectors. The mentioned curse of dimensionality is a non-issue as it applies, if the the number of irrelevant of features is high. This is typically solved by reducing the dimension of the feature vectors to include only relevant features [13]. Neural embeddings however seem to implicitly capture only relevant features [30].

2.4 Problem statement

The task of this thesis is the classification of orphan classes in Wikidata. In other words a function is needed, which given an orphan class u of a taxonomy $T = (C, S)$ with a root class $root_T$, find an appropriate superclass for u.

Doan et al. [12] suggests that for the task of placing a class into an appropriate position in T, either finding the most similar class, most specific superclass, or most general subclasses of u, are sensible approaches. Therefore a "most-specific-parent" similarity measure $sim_{msp}(a, b)$ is assumed, which defines similarity between a subclass a and a superclass b. $sim_{msp}(a, b)$ is 0, if b is not a superclass of a.

The problem is defined as follows:

Definition 10 (Problem definition)**.** Given a taxonomy $T = (C, S)$ with root class $root_T$ and a similarity function sim over T, find a function $f : C \mapsto C$, which, given an orphan class $u \in C$, returns a class $s = f(u)$, fulfilling the following criteria:

$$\forall p \in P : \neg(p \lhd_T u) \text{ no children} \tag{7}$$

$$s = \max_{s \in C}(sim_{msp}(u, c)) \text{ most specific parent} \tag{8}$$

The stated problem induces several challenges, which will be listed here, but addressed in later sections.

1. **Multi-label classification**. Algorithms for classification typically map entered objects to one label. In Wikidata and other ontologies, it is possible for one class to have multiple superclasses. It has to be decided whether the solution will be simplified to a single-label classification, or attempt multi-label classification.

2. **High number of labels**. An orphan class has to be assigned to a class in the root taxonomy. The only restricting condition is that the chosen class cannot be a subclass of the orphan class. As shown in Section 3 97% of 12999501 classes are part of the root taxonomy. Classification methods, like SVM or neural networks, usually classify with a small number of labels. A classification method, which is able to handle over a million labels, is required.

3. **Representation of items**. Items in Wikidata are structured information, similar to nodes in RDF graphs and Wikidata can be represented as such [38] [14]. As motivated in Section 1, the solution should exploit the apparent power of neural networks. Neural networks, which are introduced in Section 2.6, require input to be represented as vectors. Therefore it will be necessary to map items to vectors.

2.5 k-nearest-neighbors classification

Based on the characteristics of the classification problem, described by the problem statement, and the challenges attached to it, the **k-nearest-neighbor algorithm (kNN)** seems like an appropriate tool for solving the task. Nearest-neighbors classification is a lazy method, as it does not require training before testing. This is useful for applications with high amounts of data, large numbers of classes, and changing data [44] [7]. For the considered use case of classification in Wikidata, these are

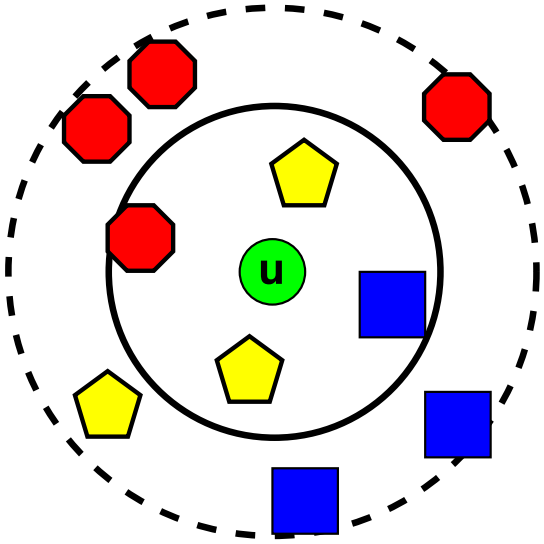

Figure 4: Example for k-nearest neighbors for 3 classes with k=4 and k=10.

very important strengths, as the number of classes in the taxonomy is very high and Wikidata is being constantly edited.

kNNs can be defined as a similarity-based classification method. kNN uses a pairwise similarity or distance measure (see Section 2.3). Access to the features of the classified objects is therefore not required [7].

The basic notion of kNN is presented in Figure 4. Given a set of points in $\mathbb{R}^2$ with classes blue, red and yellow. To classify an unknown class u, the closest k classes are selected. In the example, the solid circle indicates $k = 4$ and the dashed circle indicates $k = 10$. The k selected neighbors vote for their own class with a given weight, which is typically dependent on its similarity to u. In this simple example, it is assumed that all points have a uniform weight. Following, for $k = 4$ u is classified as yellow and for $k = 10$ as red.

In comparison to the example, the weights assigned to each neighbor are not uniform. Weights are assigned, so that similar neighbors are given larger weights (affinity). Because in practical applications many neighbors may be very similar to each other, which can skew the classification results, Chen et al. [7] propose to additionally down-weight very similar neighbors (diversity). This can be accomplished by using **kernel ridge interpolation (KRI)** [7]. The following quadratic programming has to be solved to calculate the weights:

$$\begin{aligned} &\min_{w} \frac{1}{2} w^T S w - s^T w + \frac{\lambda}{2} w^T w \\ &\text{subject to } \sum_{i=1}^{k} w_i = 1, w_i \geq 0, i = 1, \ldots, k, \end{aligned} \tag{9}$$

where $w \in \mathbb{R}^{k \times 1}$ the weights of the nearest neighbors, $S \in [0, 1]^{k \times k}$ is the similarity

matrix between the nearest neighbors of u, $s \in [0,1]^{k\times 1}$ is the similarity between u and its nearest neighbors, $\lambda > 0$ is a regularization parameter. It is common to assume that weights are positive and the sum of weights is 1. Minimizing $-s^T w$ alone would return all weight on the nearest neighbor with $w_1 = 1$. The ridge regularization term $\frac{\lambda}{2} w^T w$, which controls the variance of the weights in w, counteracts this by pushing the weights closer to uniform weights. Increasing λ increases the uniformity of the weights. Together these terms fulfill the affinity requirement. The term $\frac{1}{2} w^T S w$ represents the diversity requirement, as the weights of very similar neighbors are down-weighted to minimize the term. Solving this problem therefore returns weights, which fulfill the requirements of affinity and diversity. Experiments show that KRI-kNN on average generates better results than uniformly or distance-weighted kNN [7].

2.6 Neural networks

The notion of neural networks is introduced with the example for feedforward networks with backpropagation. The task of representing semantic information encoded in text and graphs as feature vectors using neural embeddings is then discussed. Solutions for generating word and graph embeddings by Mikolov et al. [30], Ristoski and Paulheim [38], Cao et al. [6] are presented.

2.6.1 Introduction to neural networks

A **neural network** is a triple (N, C, W), where N are **neurons**, $C = \{(i,j)|i,j \in N\}$ is a set of **connections** between neurons, and W are the weights corresponding to the connections, where W_{ij} is the weight for the connection $(i,j) \in C$ [24]. Each neuron has an activation function f_{act}, which commonly is the Fermi function

$$f_{act} = \frac{1}{1+e^{-x}} \tag{10}$$

mapping values to the interval of $(0,1)$. This function is non-linear and therefore allows the neural network to solve non-linear problems. Input neurons typically use the identity function id as activation function [24]. The output of a neuron is computed by summing up the products of weights and inputs of the neuron, which is called net input, and applying the activation function on this value:

$$o_i = f_{act}(\sum_{k=1}^{n} w_{ki} o_i) \tag{11}$$

The input for input neurons (neurons on the first layer) are the inputs of the network.

A **feedforward neural network** consists of an input layer with s_i input neurons, an output layer with s_o output neurons and d hidden layers with h neurons per layer. A neural network with many hidden layers is called deep neural network. The connections in a feedforward network only connect the neurons of a layer i with the next layer $i+1$. This means that a neuron can never influence itself. Networks, which allow this, are called recurrent neural networks. See in Figure 5 a feedforward neural network with $s_i = 2, s_o = 1, d = 1, h = 2$ with already initialized weights.

Forward propagation pushes an input through all layers of the network in a step-wise manner (layer per layer). In the first step, the input vector $\vec{x}$ is pushed to the hidden layer and outputs of the hidden layer are computed, which results in a vector $\vec{h}$ in the hidden layer:

$$\vec{h} = f_{act}(\begin{bmatrix} w_{1,3} & w_{2,3} \\ w_{1,4} & w_{2,4} \end{bmatrix} \vec{x}) \tag{12}$$

In the second step, the output of the hidden layer is pushed to the output layer resulting in the output y^*:

$$y^* = f_{act}(\begin{bmatrix} w_{3,5} & w_{4,5} \end{bmatrix} \vec{h}) \tag{13}$$

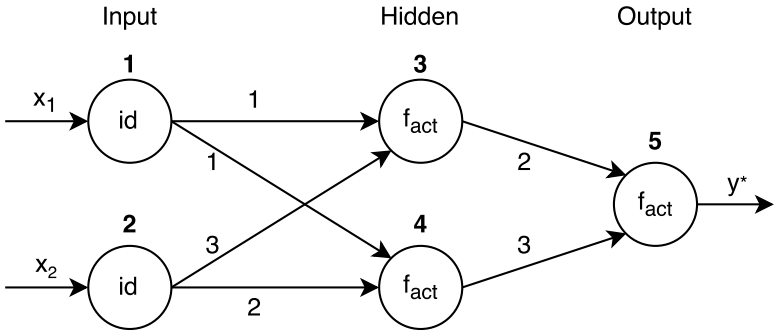

Figure 5: Example for feedforward neural network

Assuming the expected output for an input $\vec{x}$ is y, the error function is defined as

$$E = \frac{1}{2}(y - y^*)^2 \tag{14}$$

Training objective is to minimize the error function E. This can be accomplished by adjusting the weights $w_{ij} \in W$. Rather than brute-forcing the optimal weights for a network, **backpropagation** using **gradient descent** is applied in training. The idea of gradient descent is to look at the slope of the error function in respect to the current weights and adjust the weights into the descending direction. To do so the partial gradient in respect to each weight w_ij has to be derived from the error function [32]:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i \text{ with} \tag{15}$$

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} \tag{16}$$

Subsequently, the weights can be updated by adding $\Delta w_{ij} = -\alpha\frac{\partial E}{\partial w_{ij}}$ to each corresponding weight [32]. Repeating the backpropagation for a set of training samples will train the neural network to approximate a specific function.

In the following subsections, the concept of embeddings will be used to generate word and graph vector representations. By learning a certain task like predicting the probability for a certain word to appear in a context, a neural network implicitly learns hidden embeddings for the inputs. Embeddings are, in the case of Word2Vec [30], the connection weights between the input and hidden layer. More specifically, if the weights are represented as a matrix (e.g. see Equation 12), each row is an embedding for the corresponding input neuron.

2.6.2 Word embeddings

Mikolov et al. [30] introduce two neural network language models, **Continuous Bag-of-Words (CBOW)** and **Skip-gram (SG)**, which have proven to be very effective at creating word embeddings. The generated word embeddings of both models

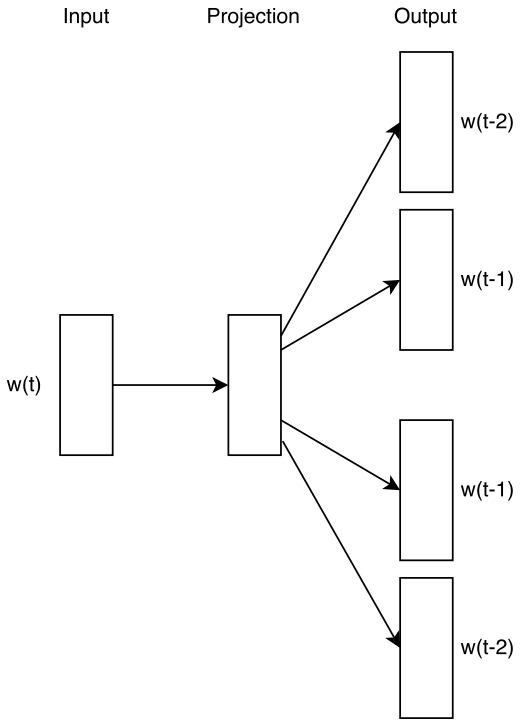

Figure 6: Skip-gram model

encode the semantics and linguistic regularities of words. Words, which are semantically close, are also close in the word embedding vector space. Calculating the offset between words makes it possible to answer more complex semantic questions. It is possible to answer more complex questions than similarity about the relationship between words. For example, the question "What is the word that is similar to *small* in the same sense as *biggest* is similar to *big*?" can be answered by calculating the offset between *big* and *biggest* and adding the vector of *small* to it: $vector(\text{"biggest"}) - vector(\text{"big"}) + vector(\text{"small"})$ [30].

Further research in this topic has shown that **SG with negative sampling (SGNS)** is generally more effective in generating high-quality word embeddings than CBOW [30] [31] [27], as it is better in representing rare words. Therefore only SGSN will be introduced in this thesis. The following description is based on work by Mikolov et al. [30] [31], Levy and Goldberg [25] [26] [17], Rong [40], and Levy et al. [27].

SGNS uses a word corpus $w \in V_W$ and the corresponding context corpus $c \in W_C$. V_W and V_C are vocabularies. Given a sequence of words $w_1, \ldots, w_n$, e.g. a text corpus like a Wikipedia text dump, the context to a word w_i are the $2 * L$ words around it: $w_{i-L}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+L}$. $L \in \mathbb{N}$ is the size of the context window. The network (see Figure 6) is trained for the task of predicting the context of an input word. Each word $w \in V_W$ is mapped to an embedding vector $\vec{W} \in \mathbb{R}^d$, and each context $c \in V_C$ is mapped to an embedding vector $\vec{c} \in \mathbb{R}^d$, where d is the embedding

size. Both embeddings are parameters, which are learned by the network. The embeddings can be represented as matrices $W \in \mathbb{R}^{|V_W| \times d}$ and $C \in \mathbb{R}^{|V_C| \times d}$, where W maps the word input to the projection layer and C maps the projection to the output layer, and therefore the context of the entered word [40].

Given a word w and a context c the model wants to maximize the probability $p(D = 1|w, c)$, that (w, c) is in the data D. The probability distribution is modeled as [25]:

$$p(D = 1|w, c) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}}} \tag{17}$$

, which leads to the maximization objective [26]:

$$\max_{\vec{w},\vec{c}} \sum_{(w,c) \in D} \log \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}}} \tag{18}$$

This problem has a trivial solution with $\vec{c} = \vec{w}$ and $\vec{c} \cdot \vec{w} = K$, for a large enough K [26] [25]. Using negative sampling solves the problem of having a trivial solution and also benefits the quality of word embeddings, as it increases the distance between word-context pairs, which do not occur in the data. Negative sampling is represented with the probability $p(D = 0|w, c) = 1 - p(D = 1|w, c)$, that a pair (w, c) does not occur in the data D. The negative sampling training objective can be written as [26]:

$$\max_{\vec{w},\vec{c}} (\sum_{(w,c) \in D} \log \sigma(\vec{c} \cdot \vec{w}) + \sum_{(w,c) \in D'} \log \sigma(-\vec{c} \cdot \vec{w})) \tag{19}$$

, where D' is a set of negative training samples, which are not in D, and $\sigma(x) = \frac{1}{1+e^x}$. For this objective $p(D = 1|w, c)$ needs to produce small values for $(w, c) \in D'$ and high values for $(w, c) \in D$, which counteracts the trivial solution possible for objective 18.

Additionally Mikolov et al. [30] introduces parameters to further influence the quality of word embeddings. Rare words that occur less than a certain threshold are not considered as words or context. Additionally very frequent words are downsampled (occur less often). This is done before context generation and increases the effective size of context windows [25], which improves the quality of word embeddings [31].

Research has shown that the effectiveness of word embeddings is highly dependent on the choice of hyperparameters [27] [31]. Experiments by Levy et al. [27] indicate that SGNS prefers a high number of negative samples. Additionally subsampling of very frequent words (e.g. "in", "a") benefits the embeddings, since these words provide less information than less frequent words [31].

SGNS has a log-linear computational complexity O in respect to the vocabulary size V

$$O = E \times T \times C \times (D + D \times \log_2(V)) \tag{20}$$

where E is the number of training epochs, T is the number of words in the training set, C is the context window, and D is the word embedding size [30].

2.6.3 Graph embeddings

Similar to generating vector embeddings for words, it is beneficial to create graph embeddings. As it allows the extraction of useful information about vertices in its graph context [6].

Where sentences are directly representable as linear sequences and can therefore be directly used in SGNS, this is not the case for graph structures like RDF graphs or protein networks. Ristoski and Paulheim [38] proposes the use of **graph walks** to generate linear sequence samples. Given an directed, weighted graph $G = (V, E)$, graph walk will generate all graph walks P_v of depth d starting in vertex v, for all vertices $v \in V$. Breadth-first search is used for generating the graph walks. This results in a set of sequences of the format $v_i \rightarrow e_{ij} \rightarrow v_j \rightarrow \ldots$, where $v_i, v_j \in V$ and $e_{ij} \in E$. The number of generated walks increases exponentially with depth d. Therefore, instead of generating all graph walks for each vertice, a random walk approach as developed by Perozzi et al. [35] is used, where the number of walks per vertice is limited. A **random walk** W_v rooted at vertice v consists of random variables $W_v^1, W_v^2, \ldots, W_v^k$. W_v^{i+1} is a vertice, which is chosen randomly from the neighbors of W_v^i. Random walking allows an easier parallelization, since multiple workers can simultaneously generate walks in different parts of the graph [35]. Evaluation results show that this approach outperforms standard feature generation approaches [38].

Cao et al. [6] develop a deep neural network for graph representation (DNGR). Instead of generating graph walks, a **random surfing** model similar to Google's PageRank is used to generate a probabilistic co-occurrence matrix, which indicates the probability of reaching a vertice j after a number of steps k from a starting vertice v_i. Similar to PageRank a teleportation probaility α is used, which indicates the chance whether the random surfing continues or is reset to the starting vertice. A row p_k of the co-occurrence matrix is therefore defined as follows [6]:

$$p_k = \alpha \cdot p_{k-1} A + (1 - \alpha) p_0 \tag{21}$$

with $p_{0_i} = 1, p_{0_j} = 0, j \neq i$. Based on the co-occurrence relation, a vector representation r can be defined. It can be assumed that vertices, which are close to the original vertice should have higher weight than distant vertices. This leads to the vector representation r for the starting vertice v_i [6]:

$$r = \sum_{k=1}^{K} w(k) \cdot p_k^* \tag{22}$$

with $p_k^* = p_{k-1}^* A = p_0 A^k$ being the probabilities of arriving in exactly k steps, if no random restart occurs, and w being a decreasing weight function. Finally a stacked

denoising autoencoder is used to produce a non-linear mapping from the representations to low dimensional vectors. Stacked implies that the autoencoder has multiple hidden layers (deep neural network), which allows the learning better embeddings with each layer [6]. The neural network encodes and decodes the inputted vectors, which performs a meaningful dimension reduction by removing redundant information and noise [6]. Evaluation with comparison to other word embedding approaches like SGNS [30], DeepWalk [35], etc. show that the combination of random surfing and deep neural networks is effective, as the approach performs better than the other baselines. However, the approach has a linear complexity in regards to the number of vertices in the graph, while SGNS has a log-linear computational complexity [30]. This is problematic in the thesis' use case, since Wikidata contains over 20 million vertices.

Random surfing has multiple advantages to sampling approaches like graph walking. Linear sequences have finite lengths and can therefore fail to capture relevant contextual information. Using random surfing overcomes this problem as it is able to consider walks of every length. Additionally, a desired property of embedding approaches is the ability to weight context based on its distance to original word or vertice [30] [6]. Random surfing allows, similarly to Word2Vec (see Section 2.6.2, the weighting of words based on its distance, which is important to create good word representations [6].

2.7 Ontology learning

Manually building ontologies is an expensive, tedious and error-prone process [19]. Maedche and Staab [29] recognize that the manual construction of ontologies results in a **knowledge acquisition bottleneck**, which motivates the research into the field of **ontology learning (OL)**. The field of OL supports ontology engineers in the construction and maintenance of ontologies by providing OL techniques in the form of tools like *OntoEdit* [29]. The process of OL can be divided into different subtasks. OL tools consist of different components, which automatically or semi-automatically support this process. The field of ontology learning exploits different research fields like natural language processing, machine learning, ontology engineering, etc. [8].

The process of OL and the components of a generic OL architecture are summarized and the task of the thesis is categorized. Following, basic algorithms for learning taxonomic relations are summarized. Both subsections are based on work by Cimiano et al. [8], Maedche and Staab [29], and Hazman et al. [19]. Finally, related work, which exploits neural networks, is analyzed and compared to the thesis' task. The novelty and additional benefits of this work are justified.

2.7.1 Process and architecture for ontology learning

The process of OLg can be divided into subtasks. Maedche and Staab [29] define a ontology learning cycle, consisting of the following steps:

1. **Import/Reuse** is the merging of existing structures and mapping between the structures and the target ontology.

2. **Extraction** defines the modeling of the target ontology by feeding from web documents.

3. **Prune** takes the generated model and adjusts the ontology to its purpose.

4. **Refine** completes the ontology at a fine granularity.

5. **Apply** applies the resulting ontology on its target application. This serves as a measure of validation.

These 5 steps can be repeated as often as necessary to include additional domains and updating it with new content. The thesis' task can be categorized to the *Refine* step, as its goal is to improve the completeness of an existing ontology.

Cimiano et al. [8] introduces a generic OL architecture and its major components. The described tool is semi-automatic, meaning that it support an ontology engineer rather than fully automatizing the process of ontology construction and maintenance. The architecture consists of the following components:

1. **Ontology management component** provides an interface between the ontology and learning algorithms. Learned concepts, relations and axioms should

be added to the ontology using this component. It is also used for manipulating the ontology. More specifically, for the importing, browsing, modification, versioning and evolution of ontologies.

2. **Coordination component** is the user interface, which should allow the ontology engineer to choose input data, learning and resource processing methods.

3. **Resource processing component** allows the discovery, import, analysis and transformation of unstructured, semi-structured and structured input. For this purpose the component needs different natural language processing components to parse and analyze input on different levels, word to sentence-level.

4. **Algorithm library component** contains the algorithms, which are applied for the purpose of ontology learning. These algorithms are generic standard machine learning methods, as well as specialized OL methods. Additionally, different similarity and collocation measures should be available.

In the context of Wikidata, there exists no single comprehensive OL tool. However, in the Wikimedia technology space different tools exist, which mainly support the task of refining and maintaining the current ontology. For example, Stratan [41] develops a taxonomy browser for Wikidata, which is able to evaluate the quality of the taxonomy by detecting different types of cycles, redundancies, errors and unlinked classes. This tool is an ontology learning component, as it provides the ability to browse and evaluate the ontology of Wikidata.

2.7.2 Approaches for learning taxonomic relations

A subgroup of algorithms for OL is concerned with learning taxonomic relations. The following approaches, categorized by Cimiano et al. [8], use text as input.

Lexico-syntactic patterns are word patterns in patterns, which are used to identify hypernym-hyponym pairs (superclass-subclass pairs) in natural text. For example, such a pattern is

$$NP_{hyper} \text{ such as } \{NP_{hypo}, \}^* \{(\text{and} \mid \text{or})\} \, NP_{hypo}$$

, where NP stands for noun phrase, and NP_{hyper} is a hypernym or superclass, while NP_{hypo} are hyponyms or subclasses. These patterns provide reasonable results, but the manual creation of patterns is involve high cost and time investments [43].

Clustering uses some measure of similarity to organize objects into groups. This can be achieved by representing the words or terms as vectors [10], on which different distance or similarity measures can be applied. Clustering methods can be categorized to three different types. Agglomerative clustering initializes each term as its own cluster and merges in each step the most similar terms into one cluster. Divisive clustering approaches the problem the opposite way by starting with a single cluster, containing all words, and then dividing them into smaller groups. Both

approaches generate hierarchies. Agglomerative clustering is doing so bottom-up and divisive clustering top-down.

Phrase analysis analyses noun phrases directly. It is assumed that nouns, which have additional modifiers are subclasses of the noun without modifiers. For example, this could be applied on the labeled unlinked classes of Wikidata. For example the classes *Men's Junior European Volleyball Championship (Q169359)* and *Women's Junior European Volleyball Championship (Q169956)* could be subclasses of *European Volleyball Championship (Q6834)*. In this case, phrase analysis interprets *Men's Junior* and *Women's Junior* as modifiers, which denote these classes as specialization to *Q6834*.

Classification-based approaches can be used, when a taxonomy is already present. In this case, classification can be used to add unclassified concepts to the existing hierarchy. Challenging with this task is that a taxonomy typically contains a large amount of classes and therefore classification methods like SVM are not suited for the task. Specific algorithms, which only consider, a subset of relevant classes are necessary to carry out an efficient classification. For example, Pekar and Staab [34] solves this problem by exploiting the taxonomy's tree structure using tree-ascending or tree-descending algorithms.

The algorithm developed by this thesis uses a classification-based approach, since a large taxonomy is already given. Instead of exploiting the hierarchic structure of the taxonomy, a variant of k-nearest-neighbors is used to address the "high number of labels" problem (mentioned in Section 2.4).

2.7.3 Neural networks in ontology learning

Fu et al. [15] develop a method to construct semantic hierarchies given a hyponym (subclass) and a list of corresponding superclasses. The solution uses neural word embeddings generated with the Skip-gram model [30], which was presented in Section 2.6.2. Using the linguistic regularities of these word embeddings, Fu et al. train a linear projection on existing sets of hypernym-hyponym pairs. A linear projection is used instead of an offset, because it was observed that the subclass-of relation is too complex to be represented as a simple offset. The embeddings are trained on a 30 million Chinese sentence corpus. And the projection is trained on 15247 hypernym-hyponym word pairs. The handpicked testing data consists of 418 entities and their hypernyms. The method achieves an F-score of 73.74% on a manually labeled dataset, and outperforms existing state-of-the-art methods.

Petrucci et al. [36] develop a recurrent neural network for ontology learning. The purpose of the work is a first step to verify whether neural networks are able to support the ontology learning process. The task solved by the specific network is the translation of encyclopedic text to logical formulas. This task is divided into the task of sentence transduction, identify logical structure and generate corresponding formula template, and sentence tagging, identifying the roles of words in the input sentence. The tagged sentence can then be mapped unto the generated formula template. This is achieved by the use of a recurrent neural network for sentence tagging

and a recurrent encoder-decoder for sentence transduction. Both are supported by gated recursive units, which provide a short-term-memory effect. The developed network is, at this time, under evaluation. Therefore, no statement about the effectiveness of recursive neural networks for the task of ontology learning can be made based on their work.

In regards to this work, the use of word embeddings for enriching the taxonomy seems promising, as the approach by Fu et al. [15] generated very good results. Additionally, research in neural word embeddings has shown a high effectivness in similarity-based tasks **citations**, which is relevant for the described problem. This work will in comparison to Fu et al. [15] attempt the task of adding taxonomic relations to an existing rather than constructing a taxonomy if given a subclass and a set of superclasses. Multiple approaches using neural word embeddings will be evaluated on the full Wikidata dataset, which has a greater size than for example the Chinese word corpus used in the previously mentioned related work. Therefore the novelty of this work, at the time of writing, is assumed.

3 Analysis of the Wikidata taxonomy

For the task of developing an algorithm, which takes orphan classes as input, it is necessary to know, what information the classes carry and if there are certain patterns among the classes. For this purpose an analysis of the taxonomy needs to be carried out, which may answer questions.

The taxonomy contained in the Wikidata dump of 2016/11/07 was analyzed. It contains a total of 24507102 items, of which 1299501 are classes. Classes were recognized as defined in Section 2.1.

3.1 Root taxonomy

The state of the taxonomy was captured in regards to the root class *entity (Q35120)* (see Figure 7). 1260842 classes are currently subclasses of *entity (Q35120)*. 97% of all classes are therefore nodes in the root taxonomy. This implies a high agreement in the Wikidata community on which class is considered root, and thereby also supports the modeling decision made in Section 2.2, which assumes that a taxonomy only has one root, and this root is *entity (Q35120)* in Wikidata.

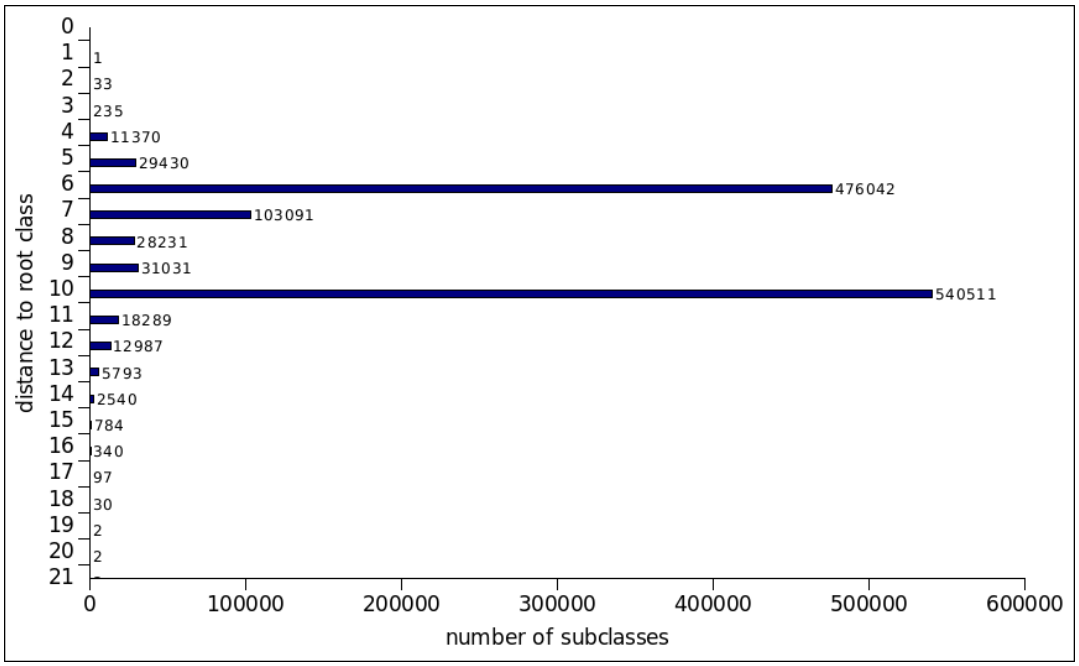

Figure 7: Distance of subclasses to root class *entity (Q35120)*. Wikidata (2016/11/07)

In Figure 7, the levels of the root taxonomy tree are represented. Most classes have a shortest distance between 6 and 10 to the root. While the highest distance to the root is 20. The most distant classes generally belong to concepts of biological taxonomies. The most distant classes are subclasses of the genus *Homo (Q171283)*.

The developed algorithm will likely place orphan classes into the same levels of the taxonomy, in which most classes already reside. If the algorithm places classes in levels close to the root, it would indicate that the classification method is not specific enough but not necessarily. On the opposite, placing classes into distant levels would indicate incorrectness of the algorithm, since only a very specific subset of classes occur there.

3.2 All classes

3.3 Relevant classes

3.4 Orphan classes

Orphan classes were identified by checking whether a class does not have the *subclass of (P279)* property.

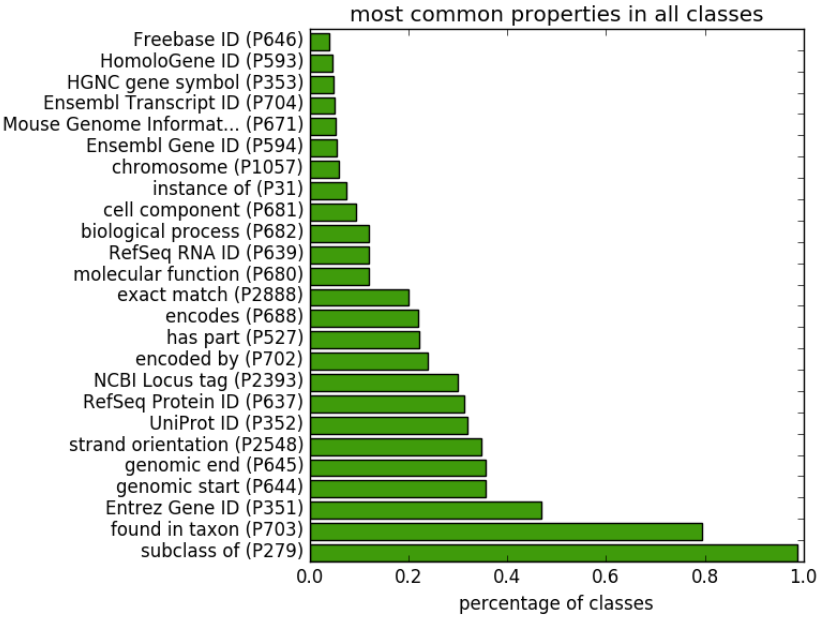

Figure 8: Frequency of properties in all classes. Wikidata (2016/11/07)

Figure 9: Percentage of all classes with a specific amount of unique properties. Wikidata (2016/11/07)

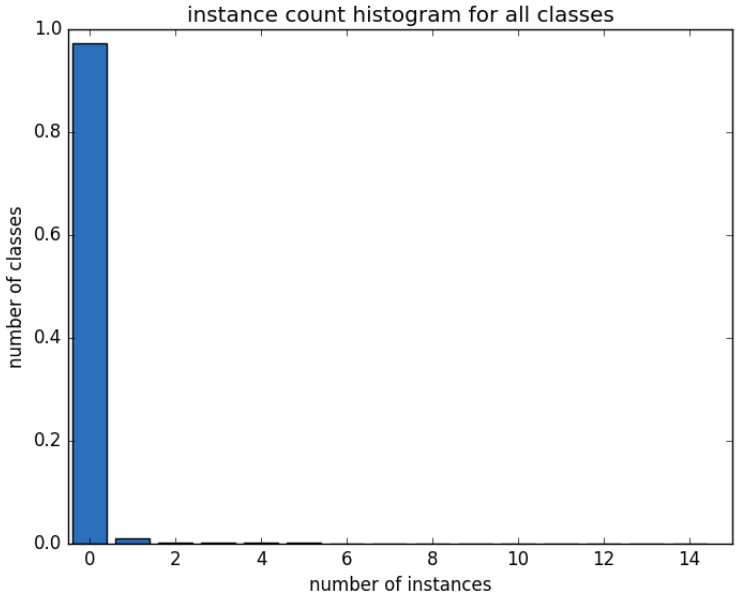

Figure 10: Percentage of all classes with a specific amount of instances. Wikidata (2016/11/07)

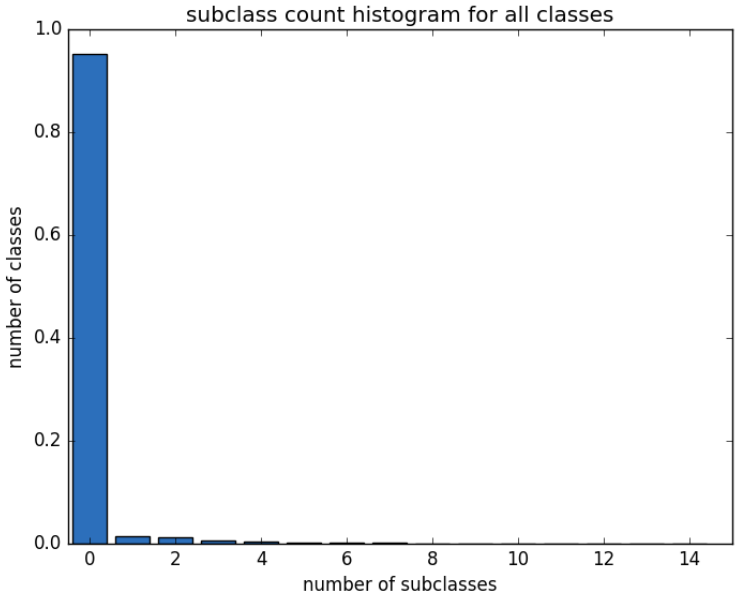

Figure 11: Percentage of all classes with a specific amount of subclasses. Wikidata (2016/11/07)

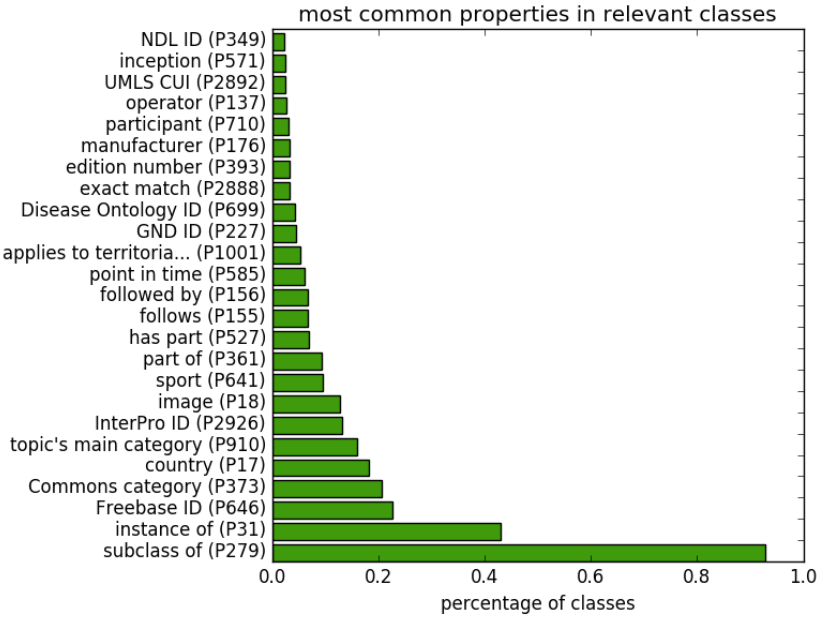

Figure 12: Frequency of properties in relevant classes. Wikidata (2016/11/07)

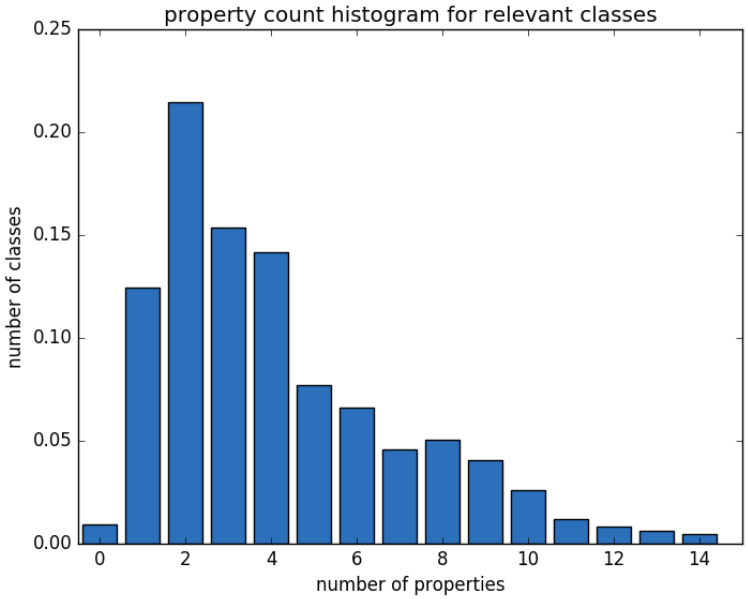

Figure 13: Percentage of relevant classes with a specific amount of unique properties. Wikidata (2016/11/07)

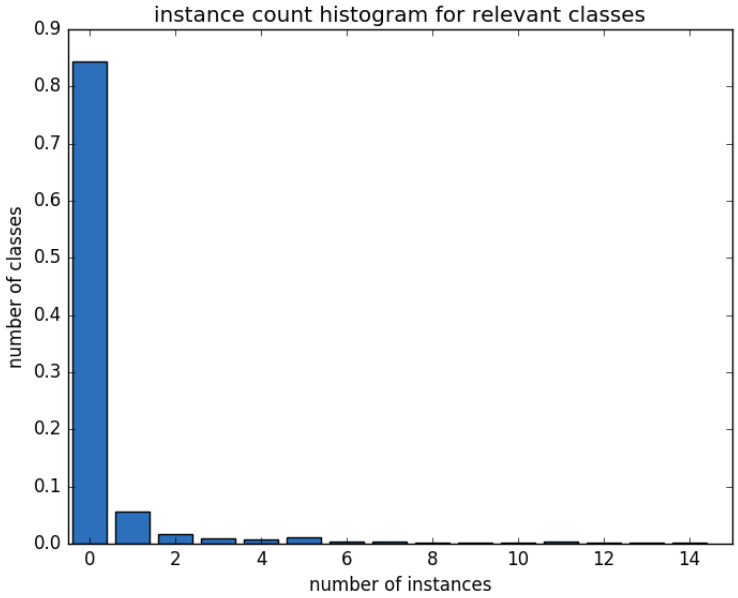

Figure 14: Percentage of relevant classes with a specific amount of instances. Wikidata (2016/11/07)

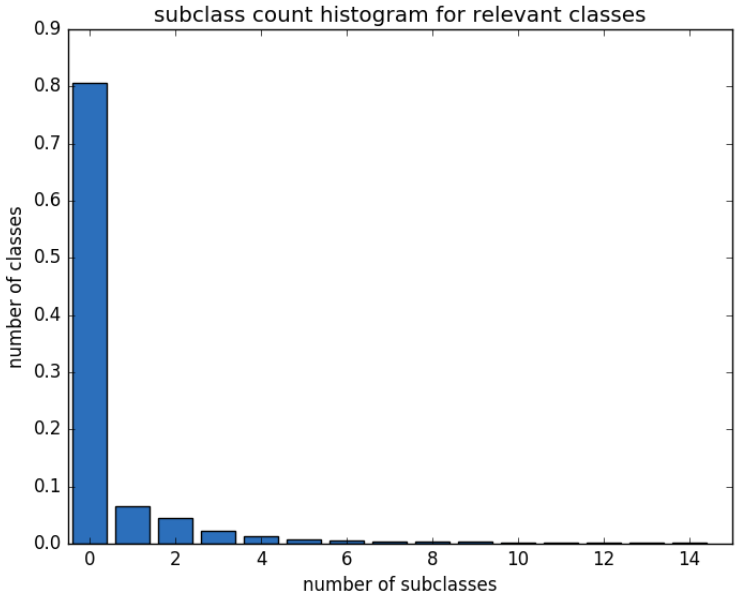

Figure 15: Percentage of relevant classes with a specific amount of subclasses. Wikidata (2016/11/07)

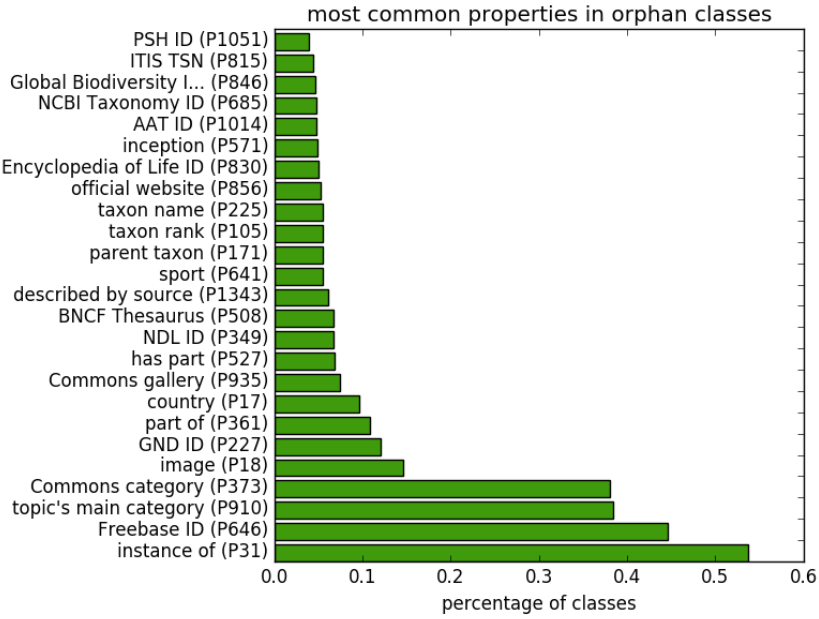

Figure 16: Frequency of properties in orphan classes. Wikidata (2016/11/07)

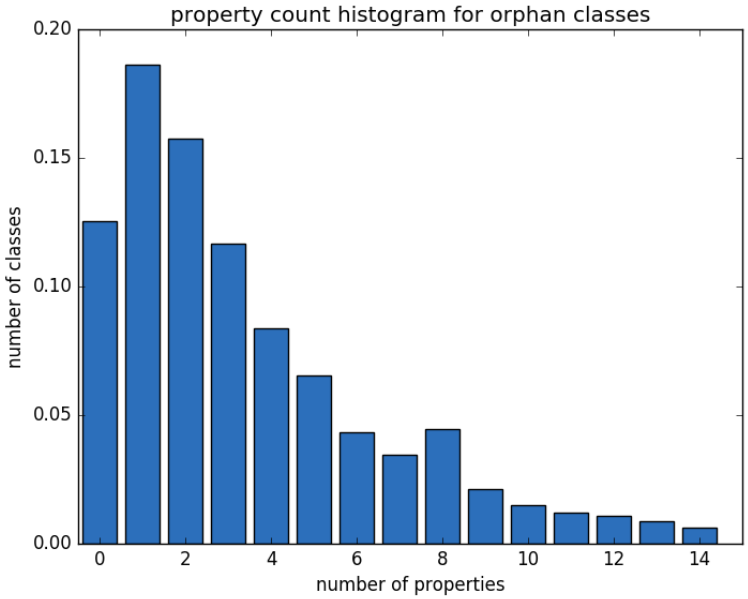

Figure 17: Percentage of orphan classes with a specific amount of unique properties. Wikidata (2016/11/07)

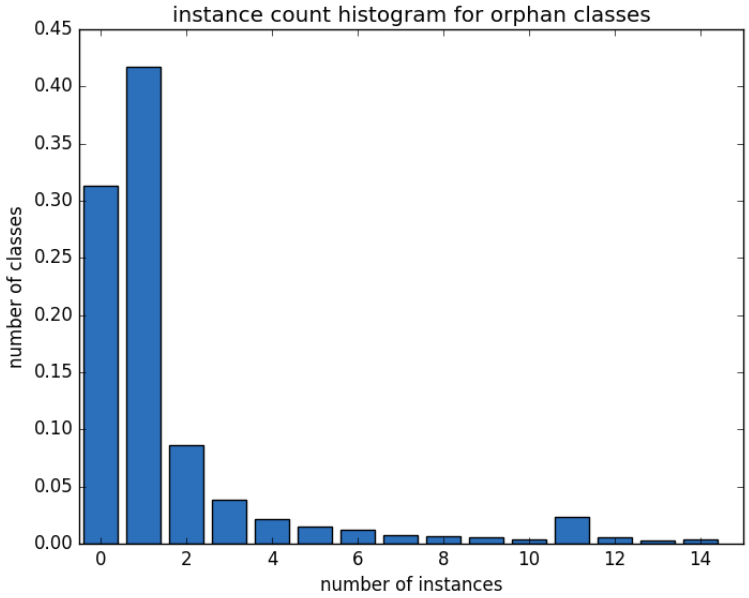

Figure 18: Percentage of orphan classes with a specific amount of instances. Wikidata (2016/11/07)

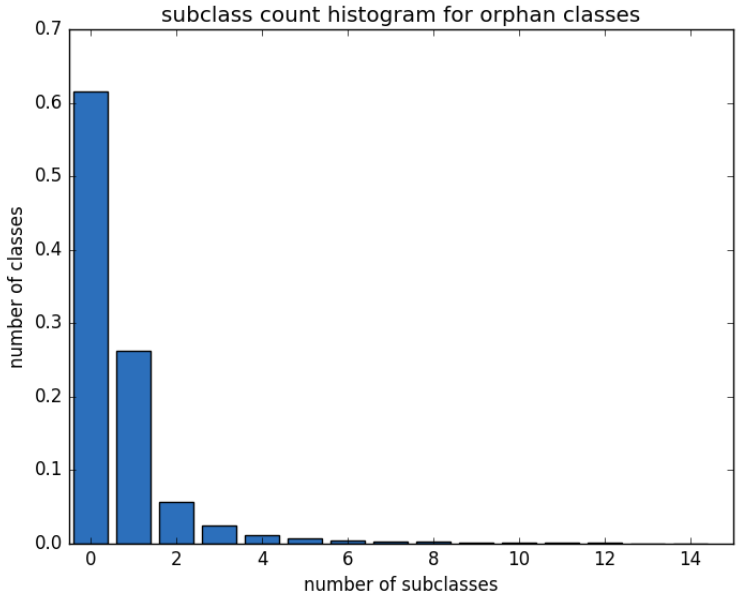

Figure 19: Percentage of orphan classes with a specific amount of subclasses. Wikidata (2016/11/07)

4 Hybrid algorithm

As motivated in the previous chapters, a hybrid algorithm using neural word embeddings is implemented to solve the defined problem. In this chapter, the required components for the algorithm are presented. For each component, possible implementations are proposed and implemented.

4.1 Components

The hybrid algorithm exploits neural word embeddings to solve the classification task, which is motivated by the power of word embeddings in similarity tasks [30] and in taxonomy construction [15].

For the task of computing word embeddings, the SGNS model will be used, which has shown to generate effective word embeddings relatively fast [30] [27]. It has been proven through theoretic analysis as well as experiments that SGNS creates very effective word embeddings, if its hyperparameters are chosen well [30] [27].

Based on Levy et al. [27], Mikolov et al. [30] and Mikolov et al. [31] the following hyperparameters are used:

- Embedding size: 300
- Subsampling frequency: 10^{-5}
- Context window: 2
- Negative samples: 15

The original paper by Mikolov et al. [30] uses context windows of size 2. Bigger context windows show slight improvements in the quality of word embeddings [27]. However triple sentences (see Section 4.2.1) only have a maximum context of 2, e.g. $w_1\, w_2\, w_3$ the maximum distance between words in the same sentence is 2. Therefore triple sentences would not benefit from bigger context sizes. To preserve comparability between variations of the algorithm, the same context size is also used for all types of SequenceGen components. The implementation of SGNS in the gensim library by Řehůřek and Sojka [37] is used.

Two additional components can be identified for the hybrid classification algorithm using SGNS. The data flow and IO of the components is visualized in Figure 20.

1. **SequenceGen**. SGNS requires a linear sequence of words as input (see Section 2.6.2). The SequenceGen component transforms any possible kind of data into a set of sentences, which can be used for training the SGNS. Possible data sets include Wikidata and Wikipedia. Wikidata's highly interlinked structure is more alike to an RDF graph [14]. Using Wikidata therefore requires an approach, which maps the graph to linear sequences. If the encyclopedic text provided by Wikipedia is used, a different challenge has to be solved. Since

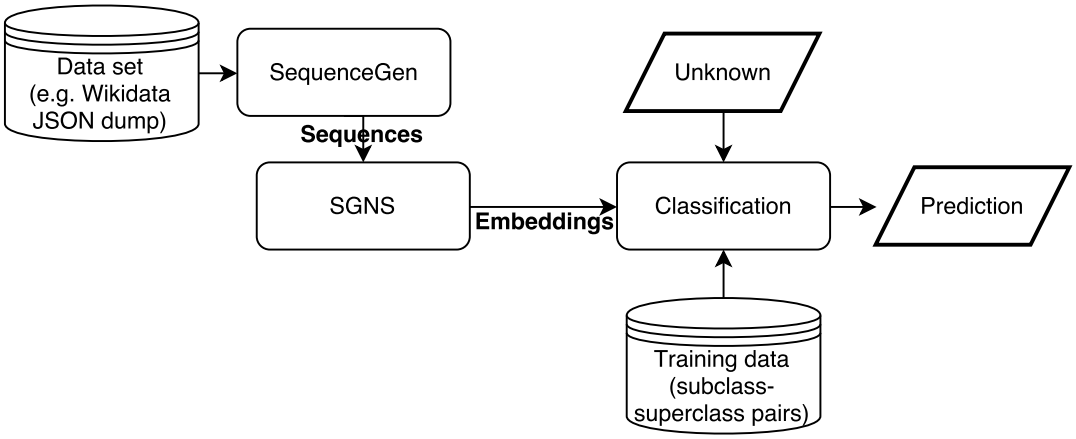

Figure 20: Data flow between components

the gold standard consists of pairs of subclass-superclass pairs, which were acquired from Wikidata [3], a mapping from words to Wikidata IDs would have to be implemented.

2. **Classification**. The Classification component is trained on a set of gold standard subclass-superclass pairs. It uses the word embeddings generated by SGNS to make a classification decision. The word embeddings produced by SGNS group similar words close to each other and preserve linguistic regularities. These properties can be exploited by different classifiers. kNN can exploit the first characteristic using a similarity or distance measure. The second characteristic can be exploited by computing the vector offset or learning a linear projection representing the subclass-of relationship between two classes [15]. The classification component should implement a single-label multiclass classification. This will simplify the evaluation and comparibility of the different algorithms.

4.2 SequenceGen

4.2.1 Triple sentences

Triple sentences uses Wikidata as input data set. Statements in Wikidata can be represented as triples of $(source, property, target)$, where $source$ is a Wikidata item ID, $property$ is a Wikidata property ID, and $target$ is either a Wikidata item ID or a literal. These triples represents simple 3-word sentences, which are sufficient as training input for Word2Vec. Appereances of literals are removed from the generated sentences to reduce the amount of noise in the data. In comparison to natural text, the triple sentence contain a very low amount of noise, which may improve the quality of embeddings in comparison to using for example Wikipedia as data set.

4.2.2 Graph walk sentences

A graph walk, developed by Ristoski and Paulheim [38] and explained in Section **??** can be used for generating longer sentences instead of triple sentences. The longer sequences are able to better capture contextual information than triples. This is achieved by creating walks of a depth d, starting from each class. Evaluation of RDF2Vec shows that Skip-gram and CBOW both produce better word embeddings using graph walk [38]. Therefore it is expected that graph walk will also improve the performance of the hybrid algorithm. Computing all walks for each vertex is however not feasible, since the number of walks for each vertex is potentially exponential [38]. The exponential runtime can be counteracted by computing a limited number of random walks instead of computing all walks [35]. Additionally, the starting vertices for each walk will be sampled from the set of classes rather than the set of all items. The quality of word embeddings in regards to classes is a major concern, while the word embeddings for other items are only optional for the task of classification. The proposed measure will ensure that each class occurs at least once in the graph walk sequences and the frequency of classes in the generated set will be higher than by random sampling from all items.

The set of random walk sequences will be combined with the set of triple sentences, which are used in the baseline, to constitute a new training data set for the SGNS.

The following parameters are used for the graph walk :

- Depth of walks: 4
- Maximum number of walks per vertice: 10

Ristoski and Paulheim [38] applies 200 maximum walks per vertice for the Wikidata dataset. Because of time and hardware constraints only 10 walks per vertice are computed, which should allow a "good" coverage of classes in the generated sequences.

4.2.3 Variation: Wikipedia

4.3 Classification

4.3.1 k-nearest-neighbors

Similar classes are grouped close to each other in the embedding vector space. Intuitively a kNN classifier (see Section 2.5) should be able to use this property to great effect.

The KRI-kNN classifier by [7] is implemented as classification component. The KRI-kNN classifier should be superior to a uniform or distance-based kNN classifier, since it not only includes the similarity of neighbors to the unclassified object into the classification decision, but also the similarity between the neighbors. Additionally, a distance-based kNN classifier using euclidian distance implemented,

which can be compared to KRI-kNN. Depending on the distribution of embeddings in the vector space, the benefits of KRI-kNN could be minimal. If this is the case, distance-based kNN should be preferred as it has a better runtime.

In the implementation of KRI-kNN, the scipy.optimize library [21] is used to compute the KRI weights. It is possible, that the weights do not converge for a set of nearest neighbors. If this exception occurs, the similarity-based weights will be used. Distance-based kNN is implemented by the scikit-learn library [33].

4.3.2 Vector offset

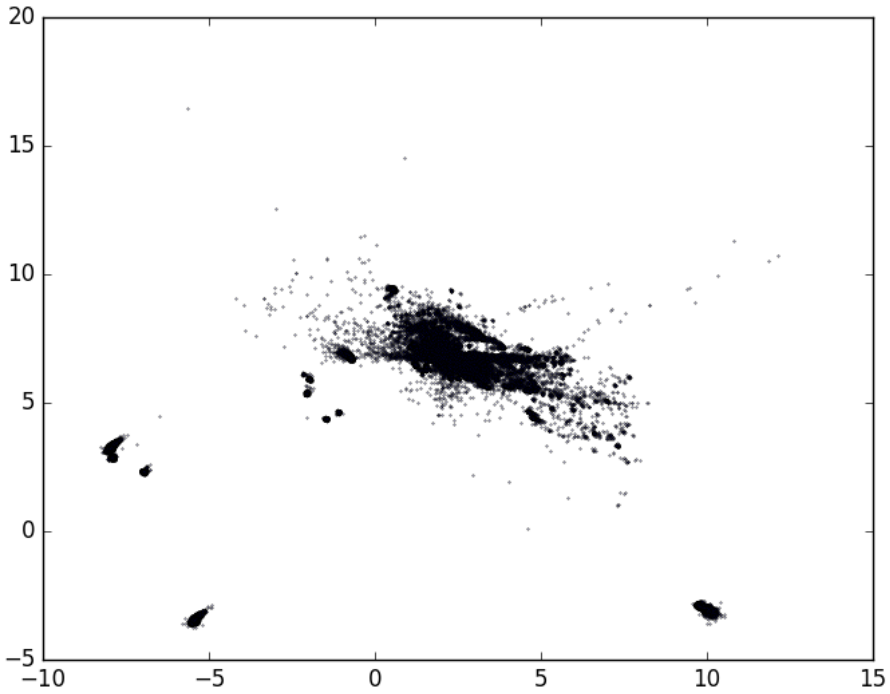

Figure 21: Subclass-of offsets of 800000 randomly chosen subclass-superclass pairs

A possible approach for classification is to exploit the linear regularities encoded in the word embeddings. As shown in Section 2.6.2, it is possible to answer semantic questions by applying algebraic operations on word embeddings [30]. Given a subclass-superclass pair with word embeddings $\vec{c}, \vec{p}$ and the word embedding $\vec{o}$ of an orphan class, the superclass with word embedding $\vec{r}$ of $\vec{o}$ could be calculated, as follows

$$\vec{r} = \vec{p} - \vec{c} + \vec{o}$$

The subclass-of relationship is represented by the offset $\vec{p} - \vec{c}$. The classification task could be simplified to the task of adding such an offset to a given word embedding. This is however only possible, if the subclass-of offset is similar for all existing subclass-superclass pairs [15].

For the use case of Wikidata, it can be shown that the offsets do not fulfill this criteria. It can rather be seen that different clusters of similar subclass-of offsets exist. Using the word embeddings created by SGNS trained on triple sentences. the subclass-of offsets for 800000 randomly chosen subclass-superclass pairs were computed. Using Principal Component Analysis (PCA), the 300-dimensional vectors were reduced to 2-dimensional vectors, which are shown in Figure 21.

It can be seen, that clusters of offsets exist, which have completely different orientations in respect to the origin $(0, 0)$. Therefore directly using vector offsets would not be applicable, as offsets can not represent the complex subclass-of relation. Fu et al. [15] reached a similar conclusion and proposes the use of linear projections as means to represent the subclass-of relation. In the following section, the approach by Fu et al. is described and implemented as a variation of the classification component.

4.3.3 Linear projection

It is assumed that a class $\vec{x}$ can be projected to its superclass $\vec{y}$ using a matrix Φ, so that $\vec{y} = \Phi\vec{x}$ applies. Fu et al. [15] propose two variants of linear projection for classification. A uniform linear projection, which trains a single matrix Φ on all subclass-superclass pairs, and a piecewise linear projection, which trains k matrices Φ_k for k clusters of subclass-superclass offsets.

Uniform linear projection tries to minimize the mean squared error of the projected $\Phi\vec{x}$ and the actual result $\vec{y}$, as follows

$$\Phi^* = \min_{\Phi} \frac{1}{N} \sum_{(\vec{x},\vec{y})} \|\Phi\vec{x} - \vec{y}\|^2 \tag{23}$$

where N is the number of $(\vec{x}, \vec{y})$ subclass-superclass pairs in the training data [15].

Piecewise linear projection learns a projection matrix Φ_k for each identified cluster in the data set. For this to work, training data needs to be selected in such a way that each cluster is represented in the training data. Otherwise the training objective is the same to the uniform linear projection:

$$\Phi^* = \min_{\Phi_k} \frac{1}{N_k} \sum_{(\vec{x},\vec{y}) \in C_k} \|\Phi_k\vec{x} - \vec{y}\|^2 \tag{24}$$

where N_k is the amount of word pairs in the k^{th} cluster C_k [15].

Both training objectives describe multivariate linear regression tasks. Fu et al. [15] uses stochastic gradient descent (SGD) to solve the objective. In the thesis' implementation SciPy's SGD regressor [21] is used to train the linear projection matrices. The thesis' evaluation will show whether linear projection is superior to the

KRI-kNN in the given task, and to what degree it benefits from word embeddings using graph walk sentences instead of triple sentences.

5 Evaluation

5.1 Method

The hybrid algorithms are evaluated using a gold standard. The gold standard is fetched from classes in the 2016/11/07 Wikidata dump [3]. $1,283,128$ classes with superclasses, where retrieved from the dump. As test data $200,000$ classes were randomly chosen. The remaining $1,083,128$ classes are used as training data. The retrieved subclass-superclass relations should be sufficient as gold standard, because they are curated by experts of the Wikidata community.

Precision, recall and F1-score cannot effectively be used to evaluate classification problems with high amounts of classes in a taxonomy [23]. These measures are typically used for flat classification problems with small amounts of classes, which are not related. It has to be recognized that a misclassification in the thesis' use case is not a binary problem. Assuming a pair of prediction and gold standard, the predicted class could be very similar or very different to the gold standard. The evaluation should consider the similarity between prediction and gold standard, rather than only differentiating between correct and incorrect classification. Good evaluation measures for ontologies should additionally evaluate the results in different dimensions [11]. Relevant dimensions, which can be used to evaluate the classification task, include the similarity of word embeddings and the distance in the taxonomy. The similarity of word embeddings can be computed using cosine similarity and represents a semantic similarity measure, because the embedding vectors encode the semantic information [30] and therefore similar vectors equates to similar classes. Distance of prediction and gold standard also describes a type of semantic similarity, since classes describing similar concepts should be grouped close in the taxonomy. Additionally, the taxonomic distance allows to evaluate other characteristics of the classification algorithm, which are described in the following paragraph.

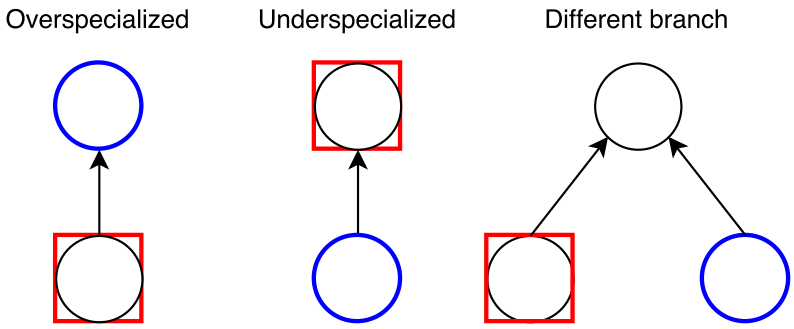

Figure 22: Possible taxonomic relations between prediction and gold standard

Legend: red box = gold standard, blue circle = prediction.

Different taxonomic relations between the prediction and gold standard are possible in a hierarchical classification task [23]. The desired case is a correct classifica-

tion, such that the prediction is equal to the gold standard. In the case of misclassification, three cases of taxonomic relations are possible. These are represented in Figure 22. Evaluation should give an insight whether the evaluated algorithm tends to over- or underspecialization or if it tends to miss the appropriate branch **Note: maybe find a better word than branch**. Additionally, the distance in the taxonomy between the prediction and gold standard should be included in this analysis, since the taxonomic distance can serve as a measure of similarity and therefore is therefore able to more accurately show the performance of the evaluated algorithm. **consider case where distance between gold and prediction is too high**

A taxonomy is modeled as a directed acyclic graph. Therefore it is possible for each class to have multiple superclasses, subsequently for each input class more than one class could be considered as gold standard. It has to be defined, how the predicted class is paired to the multiple gold standard classes. Following the decision by Kosmopoulos et al. [23], the predicted class is paired with the closest gold standard class for each measure. This achieves a minimization of the classification error, which is a sensible approach.

Scores:

- **True positive ratio** TP.
- **Mean squared error** MSE. For an algorithm alg, the set R_{alg} with $|R_{alg}| = n$ consisting of prediction-gold standard pairs. The corresponding gold standard is chosen, as previously stated, to be the closest given option to the prediction. Based on this set, the classification error for a given algorithm can be computed using the mean squared error, as follows

$$MSE = \frac{1}{n} \sum_{(\vec{p},\vec{g}) \in R_{alg}} (1 - sim_{cos}(\vec{p}, \vec{g}))^2 \tag{25}$$

  In comparison to the true positive ratio, the classification error MSE also considers the severity of misclassifications. The MSE signifies how far a prediction of the given algorithm is from the gold standard. Therefore a lower value is preferred since this induces that the classifier predicts on average very close to the gold standard. A low MSE does not necessarly also imply a high TP. A classifier could for example have a relatively high TP, because it predicts well for a specific subset of unknowns, which occurs frequently in the test samples, but otherwise predicts very badly. Using the MSE, it is possible to identify this case.

- weighted F_1-score

Plots:

- Density function over prediction to gold standard similarities.
- Distribution function $P[x \geq X]$ over similarities.
- Taxonomic distances for misclassification for 3 cases described in Figure 22

5.2 About the dataset

Triple sentences generated $112,904,736$ sentences, of which $66,258,922$ are triples.

5.3 Results

Different hybrid algorithms were executed based on the components proposed in Chapter 4. In Table 1, the evaluated hybrid algorithms and their components are listed.

name	Wikidata2Sequence	Classification
baseline	triple sentences	KRI-kNN
distknn	triple sentences	distance-based kNN
linproj	triple sentences	linear projection
pwlinproj	triple sentences	piecewise linear projection
gw	graph walk sentences	KRI-kNN
gw+pwlinproj	graph walk sentences	piecewise linear projection

Table 1: Evaluated hybrid algorithms

distknn uses the euclidian distance as weights. It is compared to baseline. It is expected that KRI-kNN provides better results than a distance-based kNN, since experiments by Chen et al. [7] has shown generally better classification results for KRI-kNN. The comparison between these classifiers will check whether this is the case for the applied usecase.

6 Conclusion and future work

References

[1] Property talk: P279. https://www.wikidata.org/wiki/Property_talk:P279, . Accessed: 2017-01-29.

[2] Talk: Q35120. https://www.wikidata.org/wiki/Talk:Q35120, . Accessed: 2017-01-29.

[3] Wikidata entity dump. https://dumps.wikimedia.org/wikidatawiki/entities/, 11 2016. Accessed: 2016-11-20.

[4] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Database Theory – ICDT 2001*, pages 420–434, 2001. ISSN 0956-7925. doi: 10.1007/3-540-44503-X_27. URL http://link.springer.com/10.1007/3-540-44503-X{_}27.

[5] Ebru Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, WLM '12, pages 20–28, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=2390940.2390943.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In Dale Schuurmans and Michael P. Wellman, editors, *AAAI*, pages 1145–1152. AAAI Press, 2016. URL http://dblp.uni-trier.de/db/conf/aaai/aaai2016.html#CaoLX16.

[7] Yihua Chen, Eric K. Garcia, Maya R. Gupta, Ali Rahimi, and Luca Cazzanti. Similarity-based classification: Concepts and algorithms. *J. Mach. Learn. Res.*, 10:747–776, June 2009. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1577069.1577096.

[8] P. Cimiano, A. Mädche, S. Staab, and J. Völker. Ontology learning. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 245–267. Springer, 2nd revised edition edition, 2009. URL http://www.uni-koblenz.de/~staab/Research/Publications/2009/handbookEdition2/ontology-learning-handbook2.pdf.

[9] Philipp Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387306323.

[10] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning Concept Hierarchies from Text Corpora using Formal Concept Analysis. *Journal of Artificial Intelligence Research*, 24:305–339, 2005. ISSN 10769757. doi: 10.1.1.60.228.

[11] Klaas Dellschaft and Steffen Staab. On how to perform a gold standard based evaluation of ontology learning. In *Proceedings of the 5th International Conference on The Semantic Web*, ISWC'06, pages 228–241, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-49029-9, 978-3-540-49029-6. doi: 10.1007/11926078_17. URL http://dx.doi.org/10.1007/11926078_17.

[12] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 662–673, New York, NY, USA, 2002. ACM. ISBN 1-58113-449-5. doi: 10.1145/511446.511532. URL http://doi.acm.org/10.1145/511446.511532.

[13] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012. ISSN 0001-0782. doi: 10.1145/2347736.2347755. URL http://doi.acm.org/10.1145/2347736.2347755.

[14] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing wikidata to the linked data web. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8796, pages 50–65, 2014. ISBN 9783319119632. doi: 10.1007/978-3-319-11964-9.

[15] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning Semantic Hierarchies via Word Embeddings. *Acl*, pages 1199–1209, 2014.

[16] Luis Galárraga. *Rule Mining in Knowledge Bases*. PhD thesis, Telecom ParisTech, 2016.

[17] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014. URL http://arxiv.org/abs/1402.3722.

[18] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015. URL http://arxiv.org/abs/1502.04623.

[19] Maryam Hazman, Samhaa R El-Beltagy, and Ahmed Rafea. A Survey of Ontology Learning Approaches. *International Journal of Computer Applications*, 22 (9):975–8887, 2011.

[20] Michael E. Houle, Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. *Can Shared-Neighbor Distances Defeat the Curse of Dimensionality?*, pages 482–500. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13818-8. doi: 10.1007/978-3-642-13818-8_34. URL http://dx.doi.org/10.1007/978-3-642-13818-8_34.

[21] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed 2017-03-16].

[22] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014. URL http://arxiv.org/abs/1404.2188.

[23] Aris Kosmopoulos, Ioannis Partalas, Éric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery, Springer (accepted for publication)*, june 2014. URL http://arxiv.org/abs/1306.6802.

[24] David Kriesel. *A Brief Introduction to Neural Networks*. 2005. URL http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf.

[25] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 2177–2185, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969033.2969070.

[26] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-2050.

[27] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/570.

[28] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8. URL http://dl.acm.org/citation.cfm?id=645527.657297.

[29] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2):72–79, March 2001. ISSN 1541-1672. doi: 10.1109/5254.920602. URL http://dx.doi.org/10.1109/5254.920602.

[30] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL http://arxiv.org/abs/1310.4546.

[32] Michael Nielsen. *Neural Networks and Deep Learning*. 2017. URL http://neuralnetworksanddeeplearning.com/.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Viktor Pekar and Steffen Staab. Taxonomy learning - factoring the structure of a taxonomy into a semantic classification decision. In *Proceedings of the 19th Conference on Computational Linguistics, COLING-2002, August 24 - September 1, 2002 , Taipei, Taiwan, 2002*, 2002.

[35] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014. URL http://arxiv.org/abs/1403.6652.

[36] Giulio Petrucci, Chiara Ghidini, and Marco Rospocher. Using recurrent neural network for learning expressive ontologies. *CoRR*, abs/1607.04110, 2016. URL http://arxiv.org/abs/1607.04110.

[37] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[38] Petar Ristoski and Heiko Paulheim. *RDF2Vec: RDF Graph Embeddings for Data Mining*, pages 498–514. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46523-4. doi: 10.1007/978-3-319-46523-4_30. URL http://dx.doi.org/10.1007/978-3-319-46523-4_30.

[39] M. Andrea Rodríguez and Max J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Trans. on Knowl. and Data Eng.*, 15(2):442–456, February 2003. ISSN 1041-4347. doi: 10.1109/TKDE.2003.1185844. URL http://dx.doi.org/10.1109/TKDE.2003.1185844.

[40] Xin Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014. URL http://arxiv.org/abs/1411.2738.

[41] Serghei Stratan. *Software Implementation for Taxonomy Browsing and Ontology Evaluation for the case of Wikidata*. Master thesis, Technische Universität Dresden, 2016.

[42] Roger Weber. *Similarity Search in High-Dimensional Vector Spaces*. PhD thesis, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, 2000.

[43] Wilson Wong, Wei Liu, and Mohammed Bennamoun. Ontology learning from text: A look back and into the future. *ACM Comput. Surv.*, 44(4):20:1–20:36, September 2012. ISSN 0360-0300. doi: 10.1145/2333112.2333115. URL http://doi.acm.org/10.1145/2333112.2333115.

[44] Min-Ling Zhang and Zhi-Hua Zhou. A k-Nearest Neighbor Based Algorithm for Multi-label Classification. volume 2, pages 718–721 Vol. 2. The IEEE Computational Intelligence Society, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1547385.

Appendices

A Appendix: Graphs

Definition 11 (Directed graph)**.** A **directed graph** G is an ordered pair $G = (V, E)$, where V is a set of vertices, and $E = \{(v_1, v_2) \mid v_1, v_2 \in V\}$ is a set of ordered pairs called directed edges, connecting the the vertices.

Definition 12 (Subgraph)**.** Let $G = (V, E)$ and $H = (W, F)$ be directed graphs. H is called **subgraph** of G, if $W \subseteq V$ and $F \subseteq E$.

Definition 13 (Predecessor)**.** Let $G = (V, E)$ be a directed graph. $v_1 \in V$ is a **predecessor** of $v_2 \in V$, if there exists an edge so that $(v_1, v_2) \in E$. Let $v \in V$ be a vertice of G, then $pred_G(v) = \{w \mid (w, v) \in E\}$ is the set of predecessors of v.

Definition 14 (Successor)**.** $v_1 \in V$ is a **successor** of $v_2 \in V$, if there exists an edge so that $(v_2, v_1) \in E$. Let $v \in V$ be a vertice of G, then $succ_G(v) = \{w \mid (v, w) \in E\}$ is the set of successors of v.

Definition 15 (Walk)**.** Let $G = (V, E)$ be a directed graph. A **walk** W of length $n \in \mathbb{N}$ is a sequence of vertices $W = (v_1, \ldots, v_n)$ with $v_1, \ldots, v_n \in V$, so that $(v_i, v_{i+1}) \in E \; \forall i = 1, \ldots, n-1$.

Definition 16 (Connected (Vertice))**.** Let $G = (V, E)$ be a directed graph. Vertices $u, v \in V$ are **connected**, if $u = v$, or $u \neq v$ and there is walk between u and v or v and u.

Definition 17 (Connected (Directed graph))**.** Let G be a directed graph. G is **connected**, if every pair of vertices in G are connected.

Definition 18 (Cycle)**.** A walk $W = (v_1, \ldots, v_n)$ of length n is called a **cycle**, if $v_1 = v_n$.

Definition 19 (Directed acyclic graph)**.** A directed graph G is called **directed acyclic graph**, if there are no cycles in G.

Definition 20 (Directed weighted graph)**.** A **directed weighted graph** G is an ordered pair $G = (V, E)$, where V is a set of vertices, and $E = \{(v_1, v_2, e_{v_1 v_2}) \mid v_1, v_2 \in V\}$, where $e_{v_1 v_2}$ is the edge weight between the vertices v_1 and v_2.