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 Juni 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)

Abstract

TODO: english abstract

Zusammenfassung

TODO: german abstract

Contents

List of Figures

List of Tables

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 [18]. 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 [23], and are therefore integral in the functioning of a KB [54]. They provide data schemas, concept vocabularies and rules [37] 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 [54].

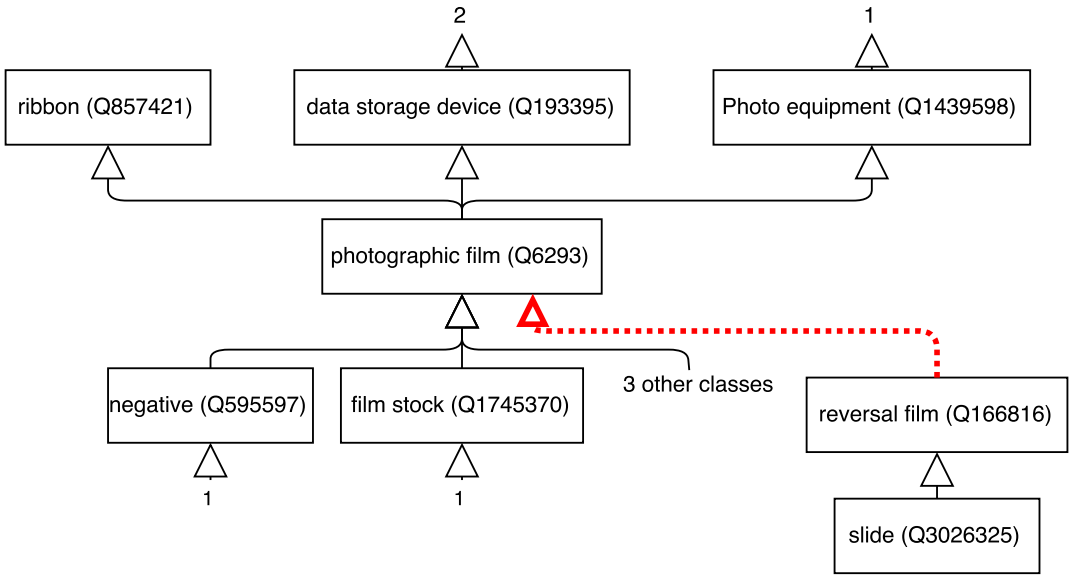

Figure 1: Classification of orphan class in Wikidata's taxonomy

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 [14]. 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. [39]. This is motivated by the effectiveness of neural networks in other context-sensitive tasks like language processing [39] [28] [5], image generation [21] etc. Neural word embeddings are especially interesting, since they have been used to great effect in the construction of taxonomies [17], graph representation [7] and semantic similarity tasks [40] [32]. 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 mostly 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.

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. Figures with long descriptions are framed to clearly separate continuous text and description of the figure. In-line quotes are indicated by quotation marks. The corresponding citation is placed after the quote. Multi-line quotes are indented. The quote is introduced by a short sentence, which contains the citation.

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 in terms of missing facts about different entities [12] 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 [18].

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, an item is 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.

Figure 2 shows an example for a Wikidata item. The item has the **label** "photographic film" and the **unique identifier** "Q6239". Below the label and identifier, the **description** (short sentence describing the item) can be found. The **alias** "film" follows in the next line. An item can have multiple aliases. Statements are grouped into **statement groups** (blue boxes) by their **property**. In this example, two statement groups with properties *topic's main category (P910)* and **subclass of (P279)** are shown. Each **statement** can have a value (orange box), **references** (green box), and **qualifiers**. In this example, the only statement with property *topic's main category (P910)* has the value *Category: Photographic film (Q8363301)* and a reference, which consists of a property *imported from (P143)* and a value *Chinese Wikipedia (Q30239)*. References represent the sources and other meta-data about the source of the statement. In this example, no qualifiers exist. Qualifiers are used to give additional details to statements. A common use for qualifiers is the annotation of statements with time stamps, if the a different value is true for different points in time e.g. population of a country over time.

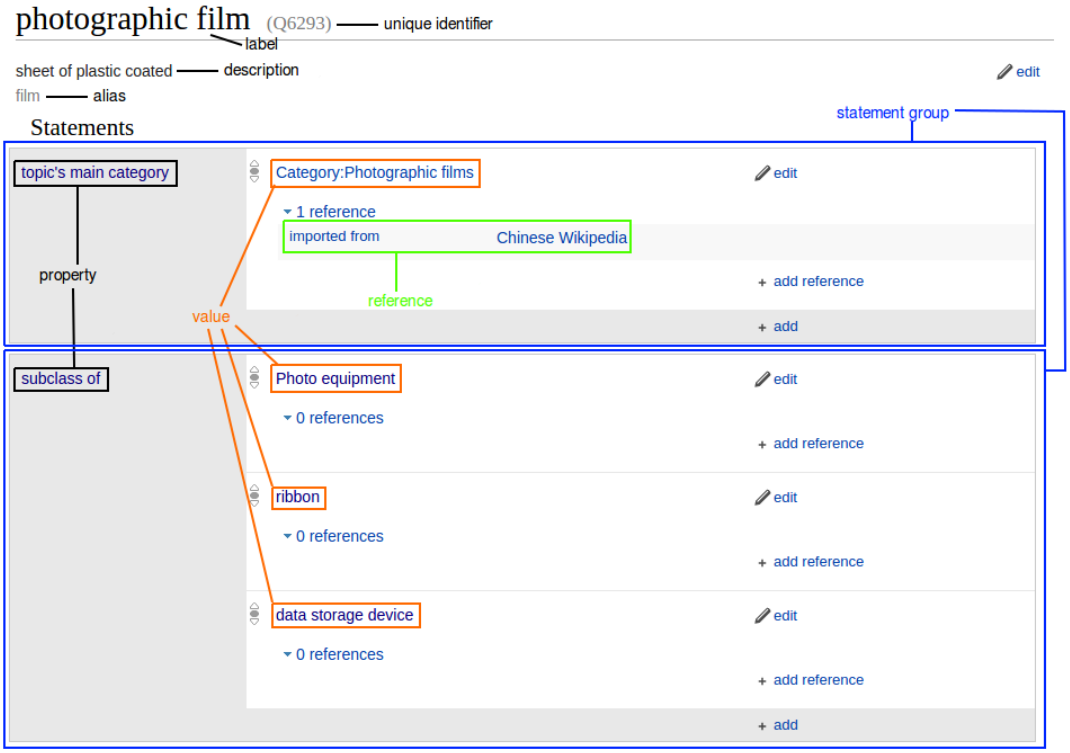

Figure 2: Example excerpt of Wikidata item: photographic film (Q6239). An item consists of a label, aliases, description, and statement group. Each statement group is associated with a specific property such as *subclass of (P270)* and can have multiple statements. The provenance of a statement can be signified by one or multiple references.

Following, the concepts item and statement are formally defined:

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.

For example, the item *photographic film (Q6239)* (Figure 2) can be formally represented, as follows:

(
6239,
"photographic film",
{"film"},
"sheet of plastic coated with light-sensitive chemicals",
{("enwiki", "Photographic film"), ("dewiki", "Fotografischer Film"), . . . }
)

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

- $itemid \in \mathbb{N}$ 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.

For example, *photographic film (Q6239)* (Figure 2) can be formally represented, as follows:

$$(6239, 910, \text{Q30239}, \{\dots\}, \varnothing)$$

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:

$\exists i = (instanceid, \_, \_, \_, \_) \in I \; \exists s = (itemid, pid, value, \_, \_) \in R :$
$instanceid = itemid \land pid = 31 \land value = classid$ (has instance)

$\exists s = (itemid, pid, \_, \_, \_) \in I : itemid = classid \land pid = 279$ (is subclass)

$\exists i = (subclassid, \_, \_, \_, \_) \in I \; \exists s = (itemid, pid, value, \_, \_) \in R :$
$itemid = subclassid \land pid = 279 \land value = classid$ (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" [37]. 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 [23]. 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 [54]. Different types of ontologies can be grouped by their level of formality and expressiveness. Wong et al. [54] 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].

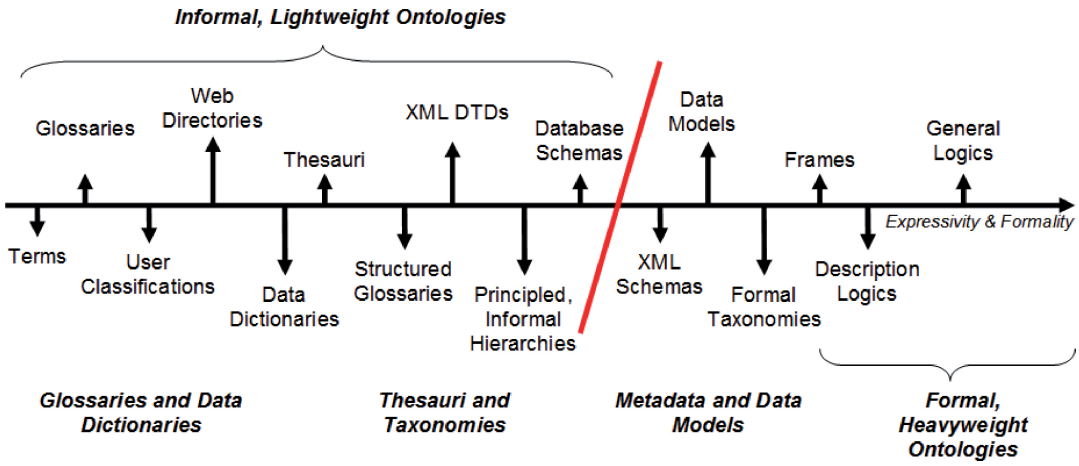

Figure 3: The spectrum of ontology kinds [54]

For the purpose of developing a formal definition of the thesis' problem statement the notion of taxonomy needs to be formalized. Cimiano defines a heavyweight ontology, which includes a taxonomy, as follows [10]:

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,
- 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.

This definition by Cimiano [10] can be mapped to Wikidata. In the Wikidata context, C is the set of classes (see Definition 3 (Class)), R is the set of properties, which target other items, and A is the set of properties, which target literals like strings.

As shown in the preceding definition by Cimiano [10], a taxonomy can be modeled as a semi-upper lattice $\leq_C$. 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 implies that every class has at least one root class, excluding the root class $root_C$. Wikidata's taxonomy does therefore not fulfill the definition, as it is not fully connected.

In the following, definitions will be 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 (DAG)**, 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 DAG, 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 transitive subclass relationship of two classes in T. Given $c_1, c_2 \in C$, $c_1 \lhd_T c_2$ applies, 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, $\forall c_1, c_2, c_3 \in C : c_1 \lhd_T c_2 \wedge c_2 \lhd_T c_3 \implies c_1 \lhd_T c_3$.

If the taxonomy defined by Cimiano [10] 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:

$$|\{c \in C \mid \neg\exists s \in C : c \lhd_T s\}| \geq 1 \tag{2}$$

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 $u \neq root_T \wedge |succ_T(u)| = 0$.

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 $R = \{c \in C | c \lhd_T root_T \vee c = root_T\}$ will be referred to as **root taxonomy** in later sections.

In Figure 4 the defined concepts of taxonomy, subclass-of relation, root class, orphan class and root taxonomy are illustrated. A **taxonomy** $T = (C, S)$ with **root class** $root_T = 1$ can be defined as DAG (see Definition 4). Each node represents a class with its class identifier. The directed edges represent the **subclass-of** relation between the classes. Node 1 is the root of the taxonomy. All gray-colored nodes are connected to $root_T$ and therefore part of the **root taxonomy**. It can be seen that the

white-colored classes 7 and 8 are not connected to $root_T = 1$, therefore they do not belong to the root taxonomy. Class 7 is an **orphan class**, since it is not the root class and has no outgoing edges.

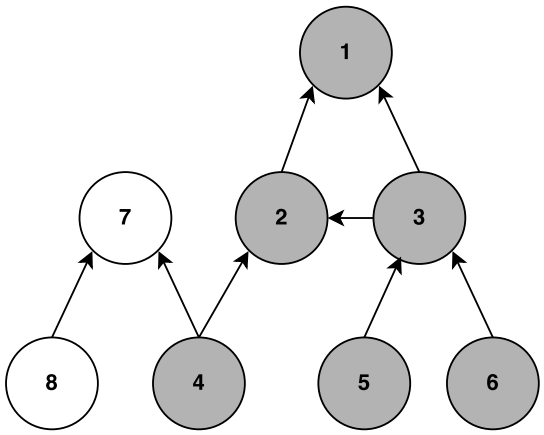

Figure 4: Example: Taxonomy DAG with root class 1 and orphan class 7. The classes 7 and 8 don't belong to the root taxonomy.

2.3 Similarity

For the task of ontology learning [23] as well as classification, e.g. k-nearest-neighbors and SVM [8], the concept of **similarity** is of importance. A **similarity measure** computes the similarity between two objects [53]:

Definition 8 (Similarity measure)**.** $sim : \Omega \times \Omega \mapsto [0, 1]$ is called **similarity measure** for a set of objects Ω. Given two objects $A, B \in \Omega$, $sim(A, B) = 1$ if $A = B$ and $sim(A, B) < 1$ if $A \neq B$.

Lin [35] states three intuitions, which a similarity measure between two objects A and B should fulfill:

- "The similarity between A and B is related to their commonality. The more commonality they share, the more similar they are." [35]
- "The similarity between A and B is related to the differences between them. The more differences they have, the less similar they are." [35]
- "The maximum similarity between A and B is reached when A and B are identical, no matter how much commonality they share." [35]

Therefore a similarity measure $sim(A, B)$ should return 1 only if $A = B$ and 0 if the A and B share no commonalities. A similarity measure should factor in both commonalities and differences to compute the similarity between objects [35].

Many types of information can be encoded as feature vectors in $\mathbb{R}^d$, e.g. words [34] [39], graphs [48] [7], etc., where $d \in \mathbb{N}$ is the dimension of the vector space. The similarity between feature vectors can be measured using a distance measure between two vectors $\vec{p}, \vec{q} \in \mathbb{R}^d$ [53]. Distance measures δ have the signature: $\delta : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}_0^+$. $\delta(\vec{p}, \vec{q}) = 0$ implies $sim(\vec{p}, \vec{q}) = 1$, while $\delta(\vec{p}, \vec{q}) \to \infty$ implies $sim(\vec{p}, \vec{q}) = 0$. Therefore it is necessary to find a mapping of distance to similarity, if a distance measure should be used as similarity measure. A typical distance measure is the L_S**-Norm** [53]:

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

with $S \in \mathbb{R}^+$. 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 [53]:

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]. Aggarwal et al. [4] also show that for increasing dimensions, the distance between the nearest and farthest neighbors converges against ∞ for L_1, against some constant $c \in \mathbb{R}$ for L_2, and for higher L_S-norms the distance converges against 0. This implies that L_S norms with higher S are bad at contrasting between vectors in high-dimensional space [4]. Therefore the distance-based kNN used in the thesis should use either the L_1- or the L_2-norm as distance measure to avoid the low contrast associated with the other L_S measures.

In ontology learning, **semantic similarity** is used to great effect, e.g for clustering objects to create hierarchies[23] [54] or mapping between different ontologies [14] [49]. 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 [49] 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, $sim(a, b) \neq sim(b, a)$.

$$sim(a, b) = \frac{|A \cap B|}{|A \cap B| + \alpha(a, b)|A/B| + (1 - \alpha(a, b))|B/A|} \tag{5}$$

for $0 \leq \alpha \leq 1$ [49].

Calculating the similarity between vectors is advantageous to calculating the similarity between objects in an ontology, because vectors abstract from the specific objects and their relations to other objects in the 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 [15]. Neural embeddings however seem to implicitly capture only relevant features [39], therefore no dimensionality reduction is necessary, if neural embeddings are used.

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 $root_T$, finds an appropriate superclass for u.

An optimal similarity function sim_{opt} is assumed for the definition of the problem statement. Let $s \in C$ be the most specific, appropiate superclass of u. The following property should be fulfilled by sim_{opt}:

$$\forall c \in C \setminus \{s\} 1 = sim_{opt}(u, u) > sim_{opt}(u, s) > u, c \tag{6}$$

This definition implies that the most similar class to u in the taxonomy T, other than u itself, is the most appropiate superclass s of u.

The thesis' 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. **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.

2. **Representation of items**. Items in Wikidata are structured information, similar to nodes in RDF graphs and Wikidata can be represented as such [48] [16]. 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 [55] [8]. For the considered use case of classification in Wikidata, these are 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 [8].

Given a set of classified objects $\Phi \subset \mathbb{R}^n$ with the set of classes C, where the function $h : \Phi \mapsto C$ returns the class for a given object. Given an unclassified object $\vec{u} \in \mathbb{R}^n$, the parameter k defines the number of nearest neighbors $\{\vec{n_i} \in \Phi | k = 1, \ldots, k\}$ of $\vec{u}$, which are considered for the classification decision. The unknown is classified by a vote. Each nearest neighbor n_i votes for its own class $c_i = h(n_i)$. The class, which has the majority vote, is then chosen as the class of $\vec{u}$.

The votes of each neighbor can be weighted based on different criteria. Uniform weights give each vote the same weight, therefore the distance between the neighbors to the unknown is not relevant for this weighting scheme.

Results produced by uniform weighted kNN are not always desirable. Assume the case, in which the parameter k was chosen too high for a given unknown, which leads to the problem that very dissimilar objects could be part of the k nearest neighbors. Using uniform weighting would be problematic, since the dissimilar objects would negatively influence the majority vote. Votes of distant objects are not as relevant to the classification as close objects [8]. This leads to the criteria of **affinity**, which states that similar neighbor should be given larger weights [8]. A typical weighting approach to achieve affinity is distance-based weighting. The distance

between each neighbor and the unknown is computed. For example in $\mathbb{R}^n$, the euclidean norm δ_2 could be used. Each neighbor $\vec{n_i}$ of $\vec{u}$ is assigned the inverse distance as weight $w_i = \delta_2(\vec{u}, \vec{n_i})^{-1}$. Therefore the weight of each neighbor is directly proportional to its similarity to $\vec{u}$.

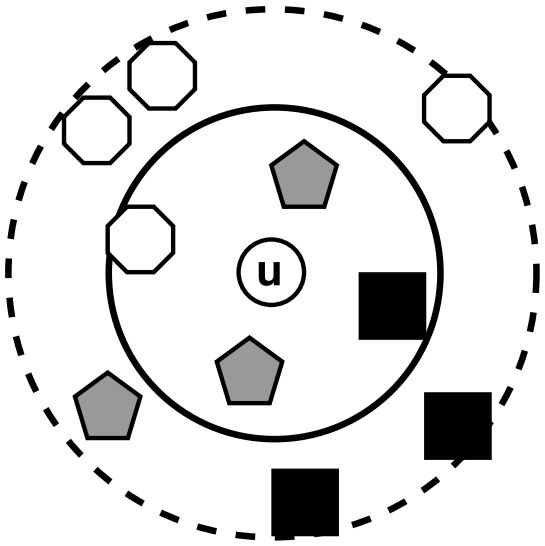

Figure 5: Example for kNN with uniform weights.

Figure 5 describes a classification using kNN with uniform weights. kNN considers only the closest k neighbors in its classification decision. In this example, the circle with name u is the unknown class, which has to be classified. 10 other objects, where each belongs to one of the 3 classes (square, pentagon, octagon), are present. Different weighting schemes are used by kNN. Each neighbor votes on the class of u. The weight controls to what degree a single neighbor can influence the class of u. The simplest approach is uniform weighting, which is used in this example. Uniform weighting assigns each object the same weight. Therefore, the distance of the neighbors to u does not matter in the classification decision.

If $k = 4$ then all objects in the inner circle are considered. 2 objects have the class pentagon, while the classes square and octagon are represented with 1 object each in the 4 neighbors. Using uniform weighting, the class pentagon gets 2 votes and therefore u would be classified as pentagon.

Changing the parameter k has influence on the results produced by kNN. If k would be set as 10 instead of 4, all object in the outer dashed circle would be considered. The classes square and pentagon have each 3 objects in this set of neighbors, while the class octagon has 4 objects in the set. Using uniform weights would therefore classify u as octagon.

It could be argued that the classification result using $k = 10$ may be inaccurate, since all objects of class octagon have a high distance to u in comparison to the objects in class pentagon. Therefore the decision made by $k = 4$ may be more

accurate. This problem can be solved by applying other weighting schemes than uniform weights to kNN. A typical approach, which was described in this section, is distance-based weights. This approach assigns higher weights to closer neighbors and could therefore solve the previously mentioned inaccuracy.

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. [39], Ristoski and Paulheim [48], Cao et al. [7] 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 $(i,j) \in C$ [30]. Each neuron has an activation function f_{act}, which commonly is the Fermi function

$$f_{act} = \frac{1}{1+e^{-x}} \tag{9}$$

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 [30]. 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:

$$o_i = f_{act}(\sum_{k=1}^{n} w_{ki} o_i) \tag{10}$$

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. Figure 6 shows a feedforward neural network with input dimension $s_i = 2$, output dimension $s_o = 1$, depth $d = 1$, and hidden layers $h = 2$. The connection weights are already initialized.

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

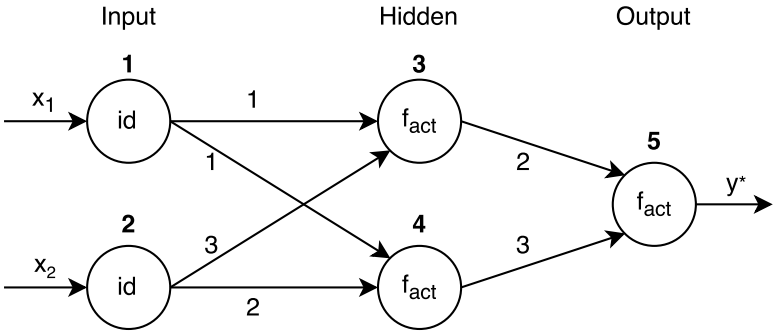

Figure 6: Example for feedforward neural network

the hidden layer and outputs of the hidden layer are computed, which results in a vector $\vec{h}$ in the hidden layer:

$$\vec{h} = f_{act}(\begin{bmatrix} w_{1,3} & w_{2,3} \\ w_{1,4} & w_{2,4} \end{bmatrix} \vec{x}) \tag{11}$$

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

$$y^* = f_{act}(\begin{bmatrix} w_{3,5} & w_{4,5} \end{bmatrix} \vec{h}) \tag{12}$$

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

$$E = \frac{1}{2}(y - y^*)^2 \tag{13}$$

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 [41]:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i \text{ with} \tag{14}$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \tag{15}$$

Subsequently, the weights can be updated by adding $\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$ to each corresponding weight [41]. 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 [39], the connection weights between the input and hidden layer. More specifically, if the weights are represented as a matrix (e.g. see Equation 11), each row is an embedding for the corresponding input neuron.

2.6.2 Word embeddings

Mikolov et al. [39] 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 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: $vector(\text{"biggest"}) - vector(\text{"big"}) + vector(\text{"small"})$ [39].

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 [39] [40] [34], 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. [39] [40], Levy and Goldberg [32] [33] [20], Rong [50], and Levy et al. [34].

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 7) 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 [50].

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 [32]:

$$p(D = 1|w, c) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}}} \tag{16}$$

which leads to the maximization objective [33]:

$$\max_{\vec{w},\vec{c}} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-\vec{w}\cdot\vec{c}}} \tag{17}$$

This problem has a trivial solution with $\vec{c} = \vec{w}$ and $\vec{c} \cdot \vec{w} = K$, for a large enough K [33] [32]. 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 [33]:

$$\max_{\vec{w},\vec{c}}(\sum_{(w,c)\in D} \log \sigma(\vec{c} \cdot \vec{w}) + \sum_{(w,c)\in D'} \log \sigma(-\vec{c} \cdot \vec{w})) \tag{18}$$

, 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 17.

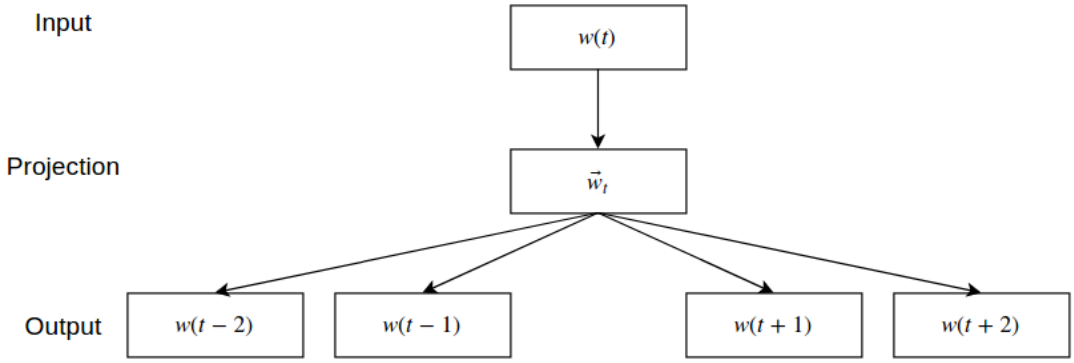

Figure 7: Skip-gram model. Given a context window of size 2 and a word sequence $(w(t+i)|i = -2, \cdots, 2)$. The skipgram model is trained on the task of predicting the skipgram $(w(t-2), w(t-1), w(t+1), w(t+2))$ given the word $w(t)$. An by-product of this training are word embeddings, e.g. $\vec{w}_t$ is the embedding for the word $w(t)$.

Additionally Mikolov et al. [39] 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 [32], which improves the quality of word embeddings [40].

Research has shown that the effectiveness of word embeddings is highly dependent on the choice of hyperparameters [34] [40]. Experiments by Levy et al. [34]

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 [40].

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

$$O = E \times T \times C \times (D + D \times \log_2(V)) \tag{19}$$

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 [39].

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 [7].

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 [48] 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. [44] 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 [44]. Evaluation results show that this approach outperforms standard feature generation approaches [48].

Cao et al. [7] 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 [7]:

$$p_k = \alpha \cdot p_{k-1} A + (1 - \alpha) p_0 \tag{20}$$

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 [7]:

$$r = \sum_{k=1}^{K} w(k) \cdot p_k^* \tag{21}$$

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 [7]. The neural network encodes and decodes the inputted vectors, which performs a meaningful dimension reduction by removing redundant information and noise [7]. Evaluation with comparison to other word embedding approaches like SGNS [39], DeepWalk [44], 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 [39]. 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 [39] [7]. 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 [7].

2.7 Ontology learning

Manually building ontologies is an expensive, tedious and error-prone process [23]. Maedche and Staab [37] 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* [37]. 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. [9].

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. [9], Maedche and Staab [37], and Hazman et al. [23]. 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 [37] 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. [9] 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 [52] 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. [9], use text as input.

Lexico-syntactic patterns are word 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 [54].

Clustering uses some measure of similarity to organize objects into groups. This can be achieved by representing the words or terms as vectors [11], 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 [43] 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. [17] develop a method to construct a semantic hierarchy (taxonomy) given a hyponym (subclass) and a list of corresponding hypernyms (superclasses), as shown in Figure 8. In this example, the input is the subclass *computer (Q68)* and its superclasses *tool (Q39546)* etc. Fu et al. [17]'s solution uses neural word embeddings, as discussed in Section 2.6.2, to construct a more specific and precise concept hierarchy. For this example, the direct superclasses of *computer (Q68)* are *tool (Q39546)* and *electrical apparatus (Q2425052)*, while the other superclasses are transitive superclasses to *computer (Q68)*.

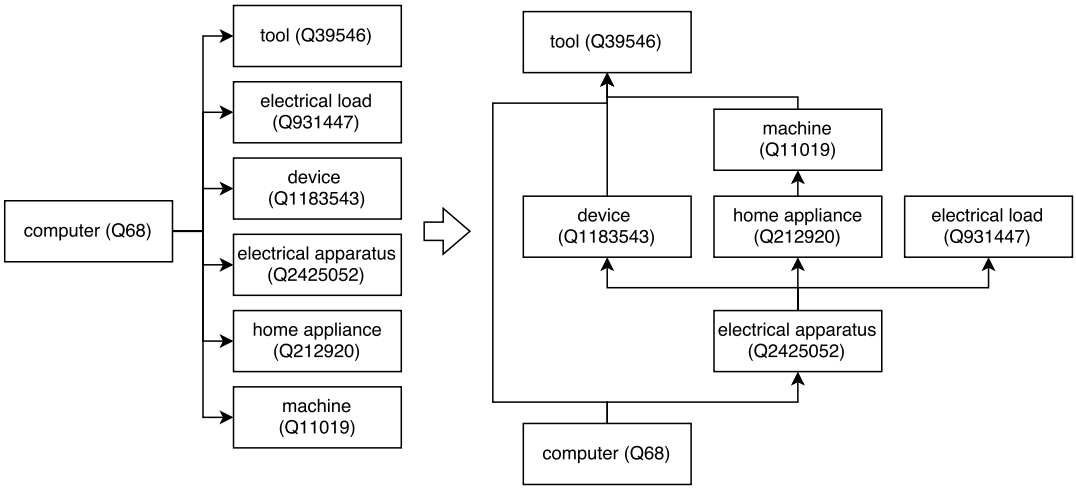

Figure 8: Construction of semantic hierarchy in Fu et al. [17].

The approach proposed by Fu et al. [17] exploits the linguistic regularities of neural word embeddings. It is assumed that there is a subclass vector offset between the subclass and its superclass embeddings, which behaves similar to the prominent example by Mikolov et al. [39]: Queen $=$ King $-$ Man $+$ Woman. Computing such subclass offsets for random subclass-superclass pairs for Fu et al.'s dataset as well as the Wikidata dataset [3] analyzed in this thesis (see Section 4.3.2) shows that clusters of very similar subclass offsets exist. Analysis by Fu et al. reveals that the offsets in one cluster refer to similar topic, e.g. single cluster concerned about people's occupations [17]. The subclass offsets in the Wikidata dataset reveal similar characteristic (see Section 4.3.2).

Fu et al. [17] conclude that the subclass-of relation is too complex to be described by a translation. Instead a linear projection is trained by minimizing the

mean squared error between projection of the subclass $\vec{x} \in \mathbb{R}^d$ and the actual superclass $\vec{y} \in \mathbb{R}^d$, where d is the embedding size. The following minimization problem is thereby solved:

$$\Phi^* = \min_{\Phi} \frac{1}{N} \sum_{(\vec{x},\vec{y})} \|\Phi\vec{x} - \vec{y}\|^2 \tag{22}$$

where N is the number of $(\vec{x}, \vec{y})$ subclass-superclass pairs in the training data [17]. The problem described by this minimization problem is a multivariate linear regression problem [17].

Fu et al. argues that because of the varied types of subclass-of relations, a single matrix is not adequate at fitting all subclass-superclass pairs. Therefore piecewise linear projection is implemented. The subclass-superclass pairs are clustered by their subclass-of offsets and a projection for each cluster is trained. Piecewise linear projection learns a projection matrix Φ_k for each identified cluster in the data set [17]:

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

where N_k is the amount of word pairs in the k^{th} cluster C_k [17]. The number of clusters is fine-tuned based on the data set [17].

To determine whether a given pair of concepts $\vec{x}, \vec{y} \in \mathbb{R}^d$ have a subclass-superclass relation, the cluster k, which has the closest cluster center to the subclass offset $\vec{y} - \vec{x}$, is chosen and the projection $\Phi_k\vec{x}$ computed [17]. The pair $(\vec{x}, \vec{y})$ is a subclass-superclass pair, if either the distance between the projection $\Phi_k\vec{x}$ and $\vec{y}$ falls below a certain threshold δ or if there exist another concept $\vec{z}$, which is superclass of $\vec{x}$ and subclass of $\vec{y}$, and therefore allows the use of the transitivity of the subclass-of relation [17]. This approach allows one subclass to have multiple superclasses and is also able to identify the case, in which no valid subclass-of relation exists.

Fu et al. evaluated the piecewise linear projection on a test set of 418 subclasses and their superclasses. The embeddings are trained on a 30 million Chinese sentence corpus and the projection is trained on 15247 hypernym-hyponym word pairs. The approach achieves an F-score of 73.74% on a manually labeled dataset, and outperforms existing state-of-the-art methods.

As argued in Chapter 2.6, the use of neural word embeddings seems promising, which is supported by the good results of Fu et al.'s work. A similar approach, which is presented in Chapter 4, can be applied in the given task. The main difference in the problems is the expected input, which is one of the reasons why Fu et al. [17]'s approach works so well. Fu et al. assumes that a subclass and its superclasses are given as input. This is advantageous for multiple reasons. Only a small set of classes has to be considered for the classification decision, which are all related to subclass and are therefore related to similar topics. Additionally, the input already

provides a set of correct subclass-of relations. In the thesis' case, the only given input is the subclass. Therefore every class in the taxonomy has to be assumed as a possible superclass rather than a small subset. The problem statement also assumes that for each class a superclass has to be found. Consequently the identification of subclass-of relations by Fu et al. is not applicable anymore and another approach has to be implemented. Although piecewise linear projection is still a promising approach to use neural word embeddings for the subclass classification, the difficulty of the given problem implies worse results than in the presented work by Fu et al. [17].

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 common characteristics among classes.

The taxonomy contained in the Wikidata dump of 2016/11/07 was analyzed. It contains a total of $24,507,102$ items, of which $1,299,501$ 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 9). $1,260,842$ 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.

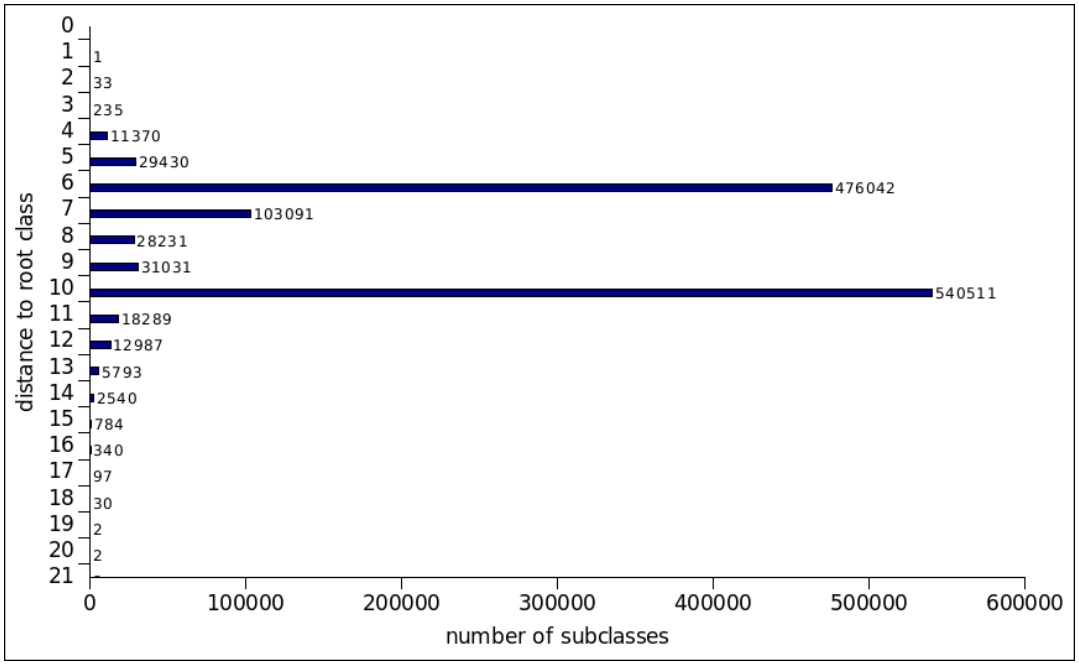

Figure 9: Distance of subclasses to root class *entity (Q35120)*. Wikidata (2016/11/07)

In Figure 9, 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. On the opposite, placing classes into distant levels would indicate incorrectness of the algorithm, since only a small subset of classes occurs in these levels.

3.2 All classes

All $1,299,501$ classes were analyzed. 98.2% of all classes have an English label. In Figure 10, the percentage of classes with a specific amount of instances is shown. It can be seen that a majority of classes have no instances. The median of instances per class is 0, while the average is ≈ 15. The high average in comparison to the low median implies that there are some classes with a very high amount of instances. Figure 11 shows the percentage of classes with a specific amount of subclasses. The median of subclasses per class 0, while the average is ≈ 1.3. Most classes do not have subclasses or instances. Subsequently, this implies that almost all classes should have the *subclass of (P279)*, since the other two criteria used to identify classes do not apply. In Figure 12, the percentage of classes with a certain amount of properties is shown. The median of properties is 7, while the average is ≈ 6.65. The similar median and average implies that few outliers exist. Figure 13 supports this conclusion that the majority of classes have the *subclass of (P279)* property. Other very common properties like *found in taxon (P703)*, *Entrez Gene ID* and *UniProt ID (P637)* indicate that at least $\approx 80\%$ of all classes are related to genes and proteins. The set of classes identified with these properties is mostly entered by bots, which map information from other knowledge bases like NCBI Entrez to Wikidata [45]. The characteristics of these classes is dissimilar to the human curated part of Wikidata, which is shown in the following Section 3.3. Additionally, the method developed by this thesis should find practical application by supporting humans in improving Wikidata's taxonomy. Therefore, the set of classes related to proteins and genes is not of relevance to the problem, and should not be considered in the development of an solution. Following, the set of relevant classes, which are of interest to human editors, is analyzed.

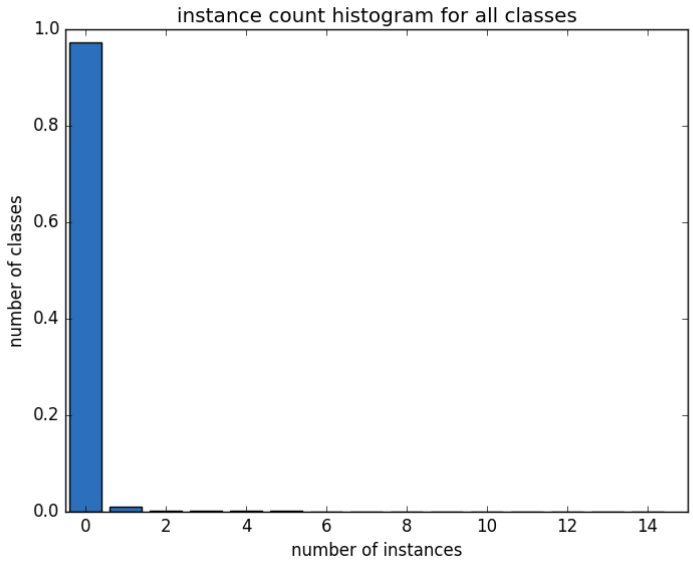

Figure 10: Percentage of all classes with specific amount of instances (2016/11/07).

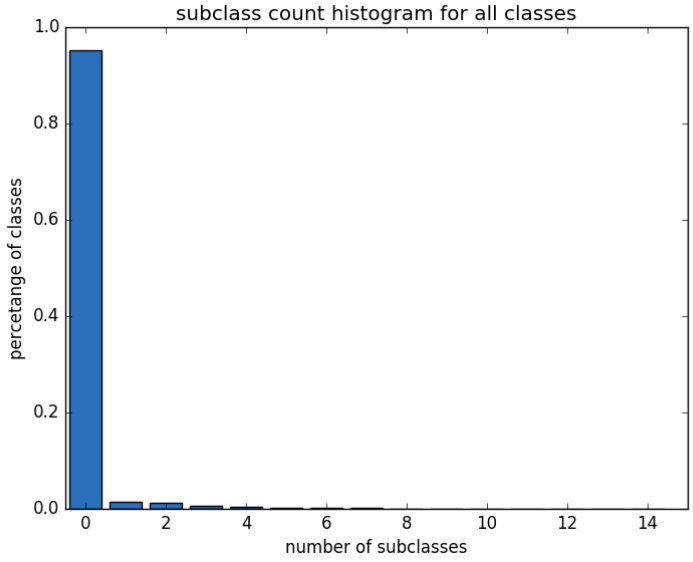

Figure 11: Percentage of all classes with specific amount of subclasses (2016/11/07).

Figure 12: Percentage of all classes with specific amount of unique properties (2016/11/07).

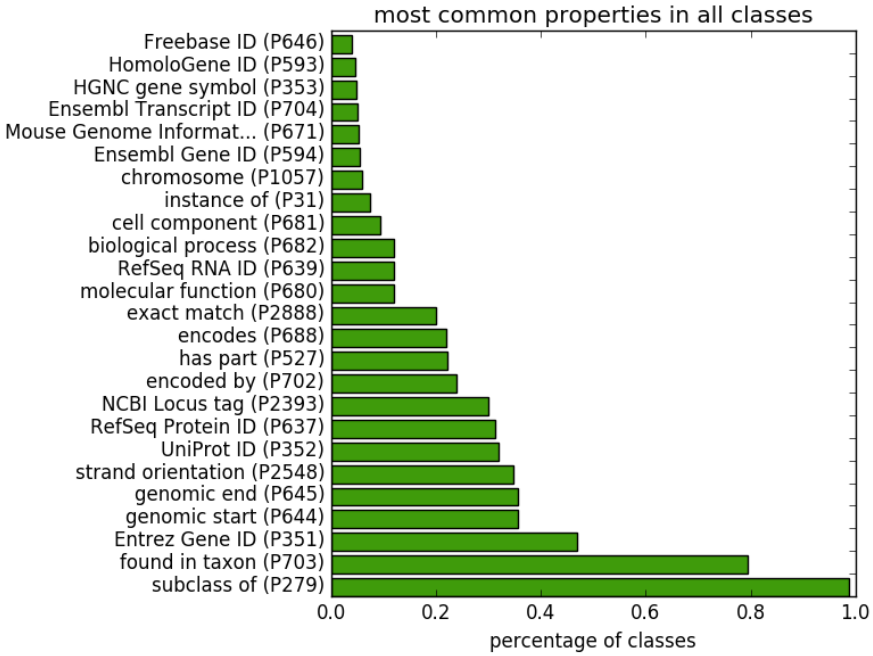

Figure 13: Frequency of properties in all classes (2016/11/07).

3.3 Relevant classes

Following the observations in the previous section, the set of relevant classes was analyzed. A class is considered relevant, if it does not have an irrelevant property. The properties considered irrelevant identify classes representing genes and proteins. The following properties are irrelevant:

- *Entrez Gene ID (P351)*
- *genomic start (P644)*
- *genomic end (P645)*
- *strand orientation (P2548)*
- *UniProt ID (P352)*
- *NCBI Locus tag (P2393)*
- *encoded by (P702)*
- *encodes (P688)*
- *molecular function (P680)*
- *RefSeq RNA ID (P639)*
- *Mouse Genome Informatics ID (P671)*
- *Ensembl Transcript ID (P704)*
- *HGNC gene symbol (P353)*
- *HomoloGene ID (P593)*
- *Gene Ontology ID (P686)*
- *InterPro ID (P2926)*

These properties were handpicked mostly based on the property frequency in Figure 13.

A total of $195,124$ classes are relevant classes, which $\approx 15.01\%$ of all classes. Of all relevant classes 87.92% have an English label, which is $\approx 10\%$ percent less than the ratio of all classes. The distribution of instances and subclasses is similar to the set of all classes (Figures 14 and 15). Both the percentage of classes with 0 instances, as well as the classes with 0 subclasses, are about 15% lower than all classes. Accordingly the instance average increases from ≈ 15 to ≈ 99.5, and the subclass average increases from ≈ 1.3 to ≈ 7.02. The median remains unchanged at 0 for both instances and subclasses. Comparing the distribution of properties of all classes (Figure 12) to relevant classes (Figure 16) shows a shift to the left, which implies a decrease of properties per class in the set of relevant classes. The average decreased from ≈ 6.65 to 4.82 and the median decreased from 7 to 4. Genes have simultaneously occurring properties, e.g. *genomic end (P645)*, *genomic start (P644)*, *strand orientation (P2548)*, and *Entrez Gene ID (P351)*, which results in a high percentage of classes having a similar number of properties, which is 7 for the set of all classes. Removing these classes consequently lowers the property median. The property frequency (see Figure 17) shows a range of different properties, which are not concerned with the same subject, as it was the case for the set of all classes. The property frequency shows that classes concerned with countries (*country (P17)*) and sports (*sport (P641)*) are relatively common. But even the most common topic related properties occur with a frequency of less than 30%, therefore it is concluded that the data is not biased to a certain topic, as it was for the set of all classes. As a result,

the hybrid algorithm will be developed and evaluated in regards to relevant classes rather than all classes.

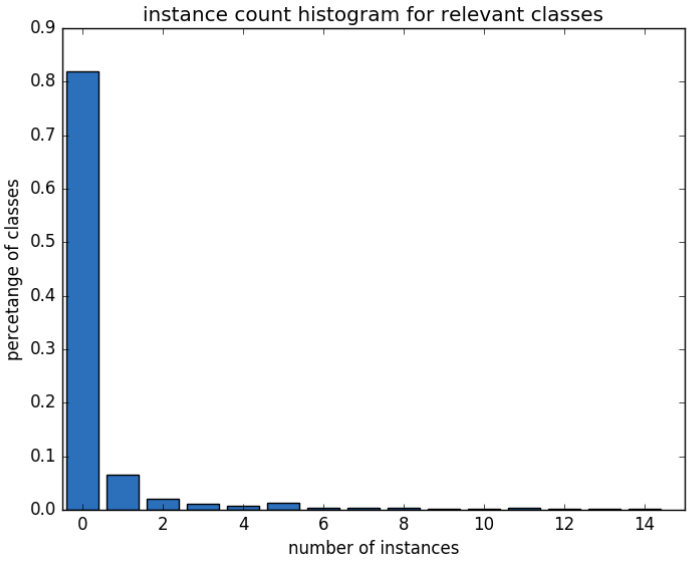

Figure 14: Percentage of relevant classes with specific amount of instances (2016/11/07).

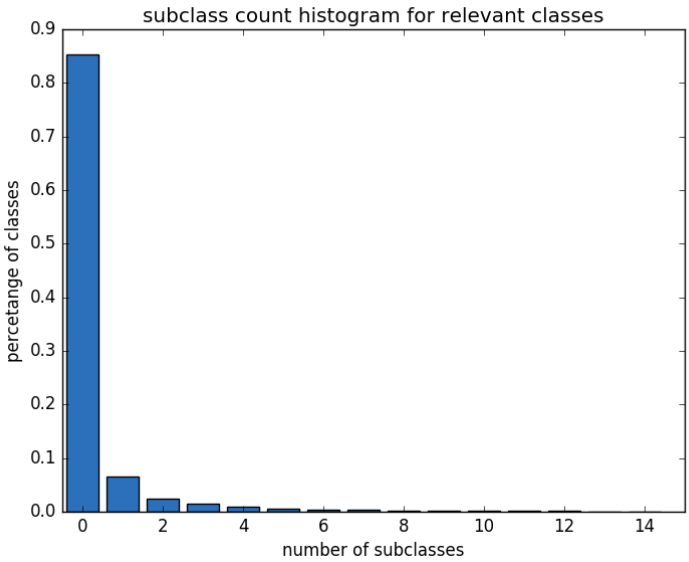

Figure 15: Percentage of relevant classes with specific amount of subclasses (2016/11/07).

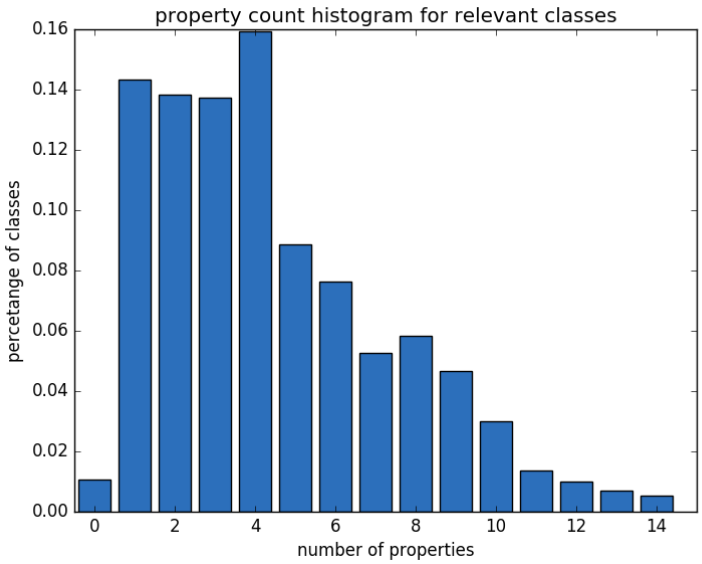

Figure 16: Percentage of relevant classes with specific amount of unique properties (2016/11/07).

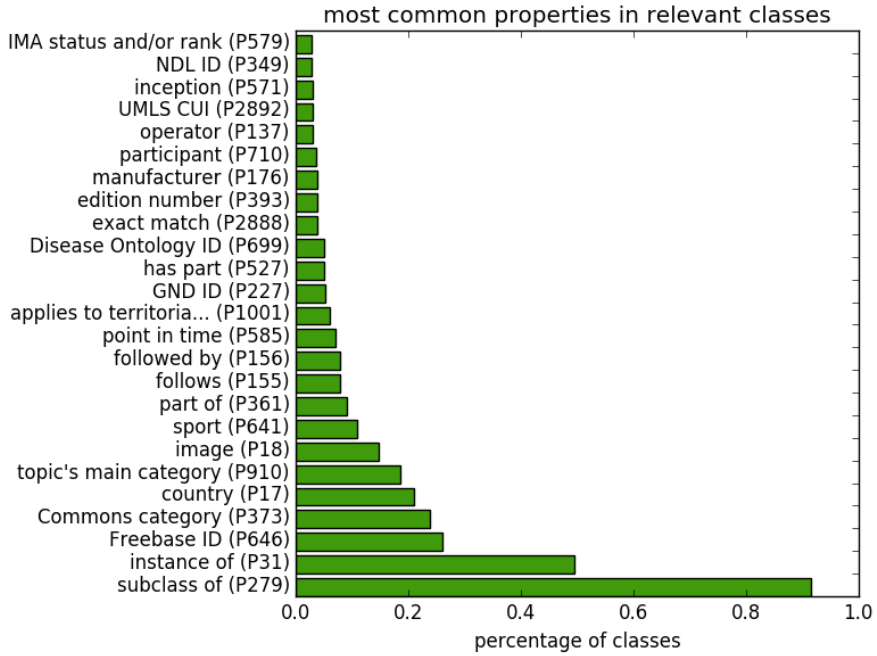

Figure 17: Frequency of properties in relevant classes (2016/11/07).

3.4 Orphan classes

Orphan classes were identified by checking whether a class does not have the *subclass of (P279)* property. The final goal of the thesis is to develop a method, which is able to classify orphan classes with a good accuracy. However, orphan classes can not be used for testing the method, because a gold standard for orphan classes is missing. Instead a gold standard can be extracted from the set of relevant classes. To ensure that the gold standard evaluation is relevant to the given problem, the set of orphan classes has to be analyzed and compared to the set of relevant classes. If the characteristics of both sets are similar, the gold standard evaluation is directly relevant to the performance in classifying orphan classes.

$16,373$ classes were identified as orphan classes, which is $\approx 1.26\%$ of all classes. $\approx 84.33\%$ of all orphan classes have an English label, which is only $\approx 5\%$ lower than the relevant classes. The distribution of instances for orphan classes (Figure 18 is very different to the relevant classes (Figure **??**). The majority of orphan classes have at least 1 instance rather than 0 instances. Consequentially, the median of instances per class is 1 for orphan classes and the average is ≈ 4.66, which is much lower than the average ≈ 86.34 of relevant classes. This, in combination with the higher median for orphan classes, implies that orphan classes have none to very few outliers in regards to many instances. Additionally, it can be hypothesized that orphan classes are often created for the purpose of classifying single instances by human editors, as a mean to provide a better description for the instance, but the orphan class itself is not further described. The subclass distribution (Figure 19) shows that $\approx 25\%$ of orphan classes have at least 1 subclass, which is $\approx 20\%$ higher than the relevant classes. However, the median of subclasses per class remains 0, while the average is ≈ 0.852 and therefore lower than the average 6.5 of relevant classes, which indicates very low or not existing amount of outliers in the set of orphan classes. The same behavior was already observed for the instance distribution. The distribution of properties per class (Figure 20) is very similar to the relevant classes (Figure 16). The only difference is the percentage of classes with 0 properties, which is $\approx 12.5\%$ for orphan classes and lower than 1% for relevant classes. This observation supports the previously made hypothesis, which claims that orphan classes are mainly created for the purpose of classifying instances, since 12.5% of orphan classes is not described at all. The corresponding property median remains at 3 and the average is ≈ 4.78, which is almost identical to the relevant classes. The most frequent properties for orphan classes, as shown in Figure 21, shows very similar properties to the relevant classes (Figure 17). This, in addition to the similar property median, is a good indicator that orphan and relevant classes have common characteristics, since their descriptions are similar. Therefore, it can be concluded that the set of relevant classes should be adequate at representing the set of orphan classes in the implementation and evaluation of the thesis' hybrid algorithm.

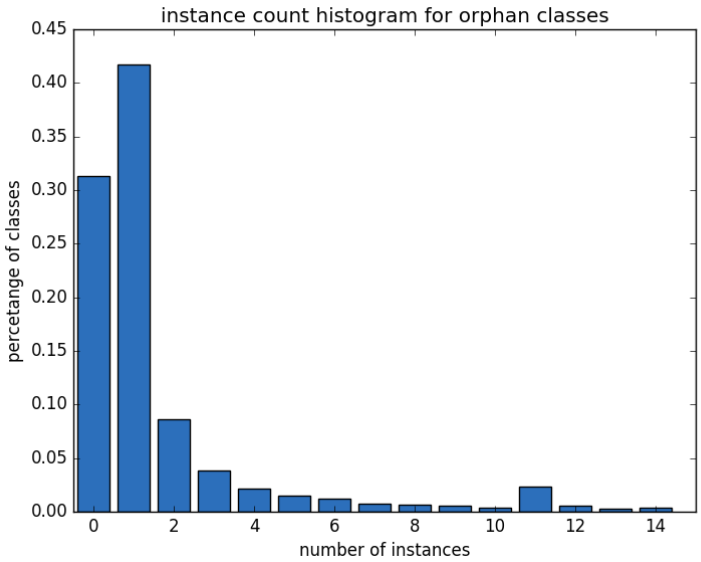

Figure 18: Percentage of orphan classes with a specific amount of instances (2016/11/07).

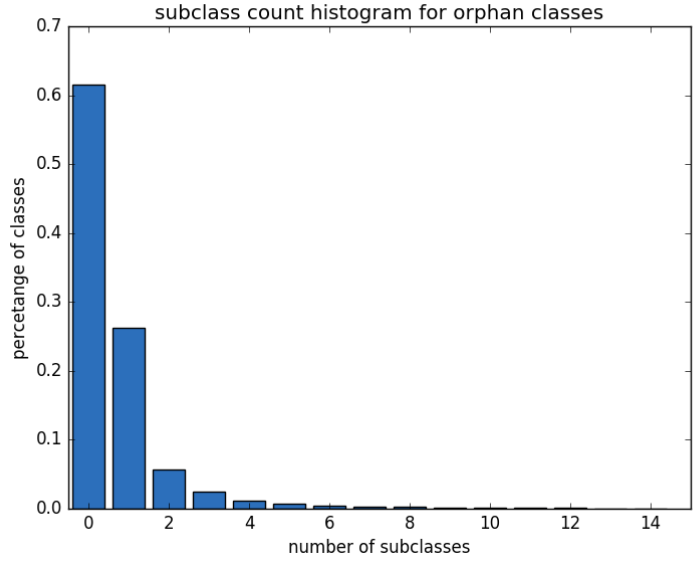

Figure 19: Percentage of orphan classes with a specific amount of subclasses (2016/11/07).

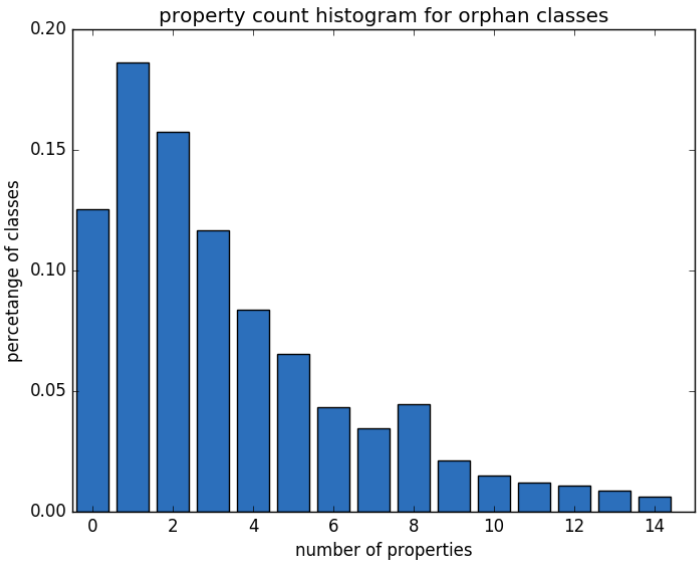

Figure 20: Percentage of orphan classes with a specific amount of unique properties (2016/11/07).

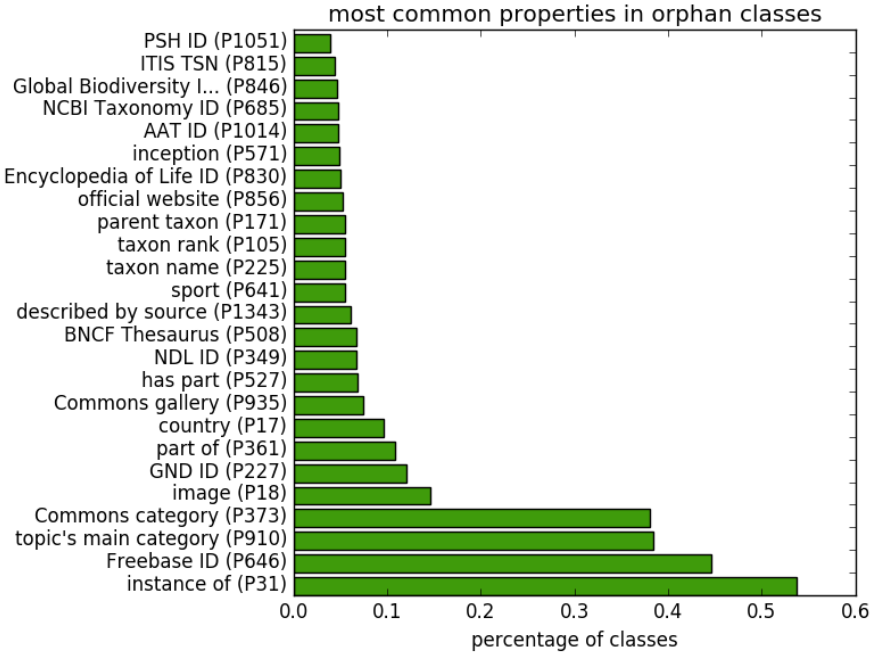

Figure 21: Frequency of properties in orphan classes (2016/11/07).

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. Additionally, only relevant classes, as described in Section 3.3, are considered in this chapter. Therefore training of the algorithm uses only of relevant classes and their corresponding instances.

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 [39] and in taxonomy construction [17].

For the task of computing word embeddings, the SGNS model (Section 2.6.2) will be used, which has shown to generate effective word embeddings relatively fast [39] [34]. It has been proven through theoretic analysis as well as experiments that SGNS creates very effective word embeddings, if its hyperparameters are chosen well [39] [34].

Based on Levy et al. [34], Mikolov et al. [39] [40] and Baroni and Dinu [6] 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. [39] uses context windows of size 2. Bigger context windows show slight improvements in the quality of word embeddings [34]. 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 [47] 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 22.

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

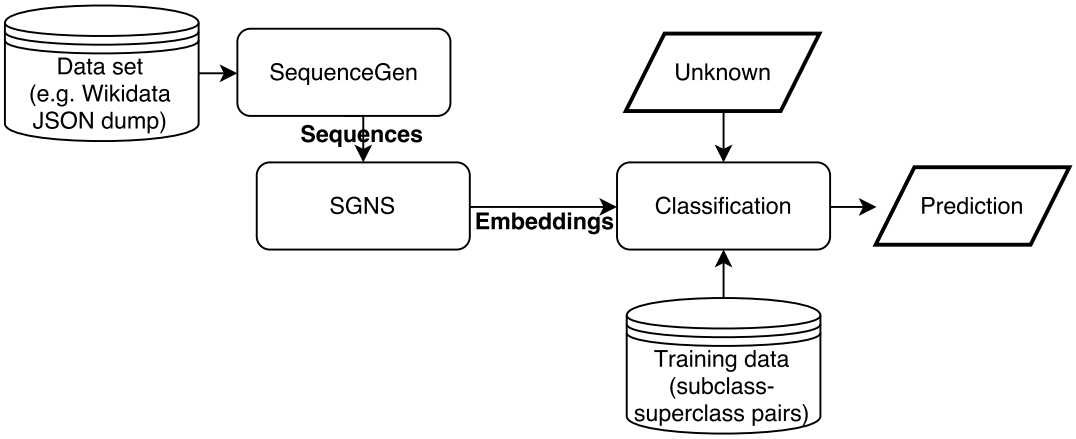

Figure 22: Data flow between components

is more alike to an RDF graph [16]. 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 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 [17]. The classification component should implement a single-label multiclass classification.

4.2 SequenceGen

SequenceGen is the first interchangeable component in the proposed hybrid algorithm. SGNS is trained on a set of word sequences (or sentences). Purpose of the SequenceGen component is the generation of sentences, where the words represent Wikidata IDs. In this work, the input for SequenceGen is a Wikidata JSON dump [3], as it directly uses Wikidata IDs and is structured data, therefore easy to automatically process. It is possible to use other knowledge bases as input, if a mapping to Wikidata can be found. For example, an incomplete mapping between Wikidata and DBpedia exists in DBpedia via the *owl:equivalentProperty* and *owl:equivalentClass* properties [26].

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.

Figure 23 shows an example mapping of Wikidata to triple sentences. Edges between entities, e.g. $(Q42, P31, Q5)$, are expressed as triple sentences, e.g. "Q42 P31 Q5", while edges between an entity and a literal, e.g. $(Q42, P1559, \textit{"Douglas Adams"})$, ignore the literal and are therefore double sentences, e.g. "Q42 P1559".

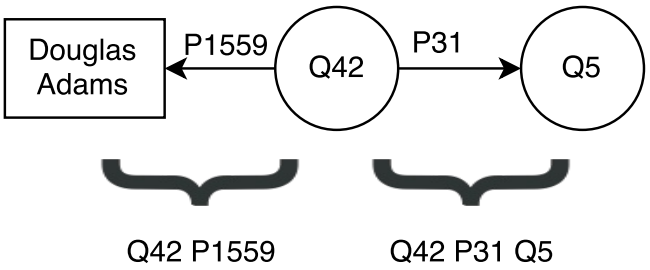

Figure 23: Generation of triple sentences from Wikidata

4.2.2 Graph walk sentences

A graph walk, developed by Ristoski and Paulheim [48] and explained in Section 2.6.3, 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 [48]. 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 [48]. The exponential runtime can be counteracted by computing a limited number of random walks instead of computing all walks [44]. 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: 200

These parameters are given by Ristoski and Paulheim [48] and have shown good results in their work.

Figure 24 shows an excerpt of the Wikidata graph. 2 random walks of depth 4 are generated beginning at the node *Q42*. The walks by the Arabic and Roman numerals. The walk indicated by Arabic numerals has a length of 3 and ends at the node *Q223557*. If no further successors can be found from a given node, the walk ends prematurely. The walks indicated by roman numerals has a length of 4 and ends at node *Q215627*. The Wikidata graph contains cycles, as is shown between nodes *Q5* and *Q8205328*. The random walks can therefore also display cycles and therefore visit nodes multiple times. The output of graph walk sentences with 2 maximum walks and depth 4 with source node *Q42* (based on Figure 24 is the following:

- "Q42 P31 Q5 P1542 Q8205328 P279 Q223557" $(1, 2, 3, 4)$
- ""Q42 P31 Q5 P1542 Q8205328 P170 Q5 P279 Q215627" (*I, II, III, IV, V*)

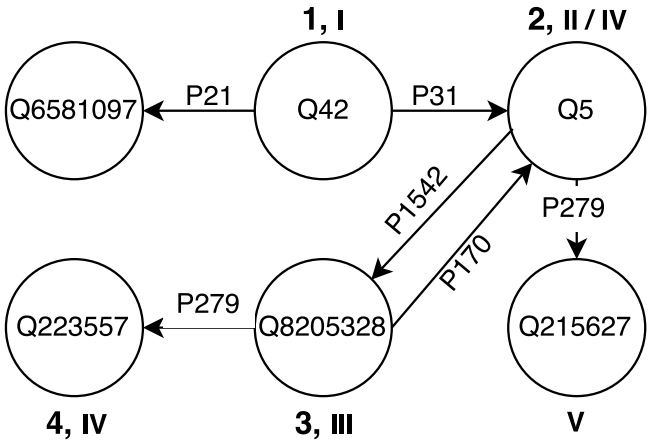

Figure 24: Two random graph walks in Wikidata with depth 4.

Multiple test runs with graph walk-based classification have shown that embeddings trained on graph walks produced worse results than the trivial baseline algorithm described in Section 5.1. This stands in contradiction to intuition and experiments by Ristoski and Paulheim [48]. The following is a list of possible issues, which may be responsible for the unexpected bad results:

- Erroneous implementation of random graph walk. The most likely reason at this time is errors in the implementation of the random graph walk, which lead to the generation of "bad", duplicate or otherwise suboptimal walks.
- Common words are dis-proportionally overrepresented. Random graph walks are more likely to capture common words like properties such as *instance of (P31)*. Classes are relatively rare words, and will therefore occur even less often in the graph walk sentences than in triple sentences, because they are captured less often by the graph walks. Consequently the generated embeddings for classes are less expressive and contain less semantic information.

Due to time constraints, the graph walk sentence component is dropped in the following evaluation.

4.3 Classification

The purpose of the classification component is the prediction of a superclass for a given orphan class, as defined in the problem statement (see Section 2.4). The classification component is trained of subclass-superclass pairs, which consist of their IDs and their respective word embeddings generated by the SGNS component. Generally, a classifier assigns an unknown object a label from a given finite list of labels. The unknown object is the word embedding of the orphan class and the output is the Wikidata ID of a superclass. The kNN algorithm, which is described in Section 2.5, is an example for a classification algorithm. Instead of predicting the label directly, another approach would be predicting the word embedding of a superclass. Then the label of the closest existing superclass embedding is chosen as output of the classifier. Predicting continuous values instead of discrete values is accomplished by regression methods. Two of these indirect regression approaches, linear projection (see Section 4.3.3) and non-linear regression via neural networks (see Section 4.3.4), are described in this section.

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.

Distance-based kNN using the euclidean norm is implemented as classification component. The kNN implementation provided by the scikit-learn library [42] is used.

4.3.2 Vector offset

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 [39]. Given a

subclass-superclass pair with word embeddings $\vec{x}, \vec{y} \in \mathbb{R}^d$ and the word embedding $\vec{u} \in \mathbb{R}^d$ of an unknown class, the superclass with word embedding $\vec{p} \in \mathbb{R}^d$ of $\vec{u}$ could be calculated, as follows

$$\vec{p} = \vec{y} - \vec{x} + \vec{u}$$

where $d \in \mathbb{N}$ is the embedding size. The subclass-of relationship is represented by the offset $\vec{y} - \vec{x}$. Using the subclass offsets would simplify the difficult task of classifying an unknown class in a big taxonomy to a vector addition, which represents a translation. This is however only possible, if the subclass-of offset is similar for all existing subclass-superclass pairs [17].

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 $204,158$ subclass-superclass pairs were computed. The subclass-superclass pairs were extracted from the set of relevant classes (Section 3.3). The offsets were clustered using KMeans with $K = 15$ clusters.

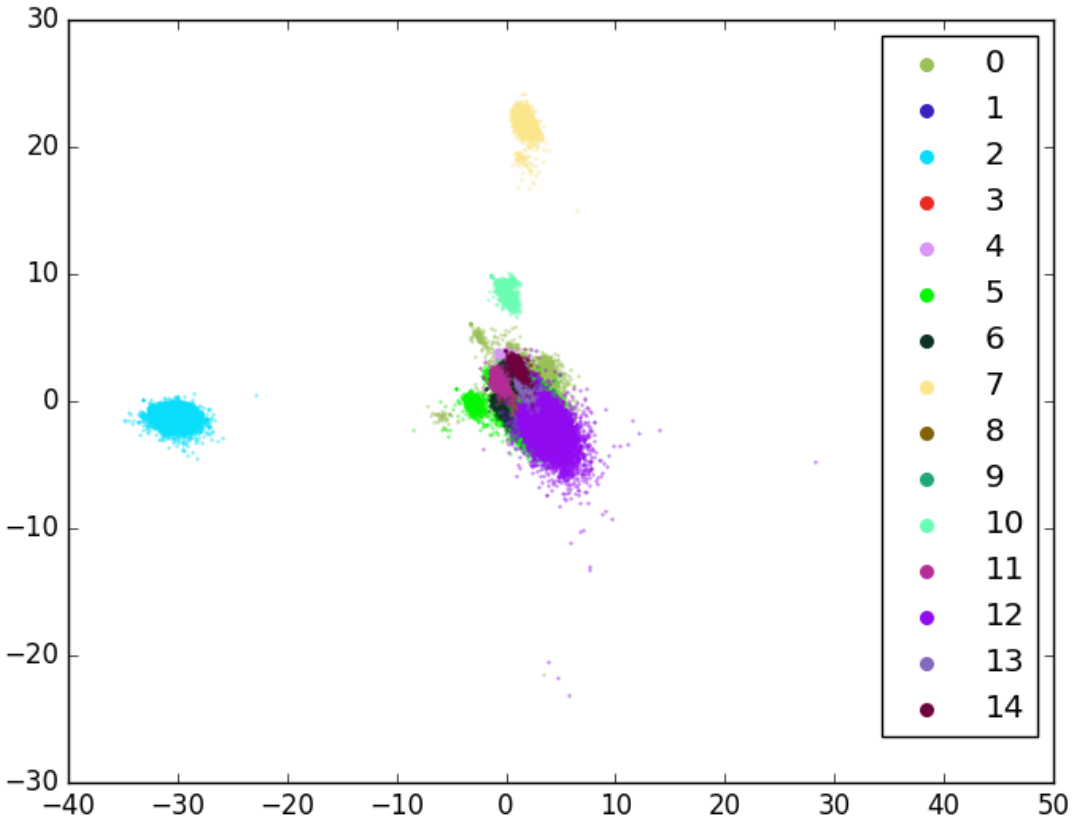

Figure 25: Subclass-of offsets of $240,789$ subclass-superclass pairs. Each cluster is identified by a specific color. The offsets are clustered into 15 clusters using KMeans clustering.

Using Principal Component Analysis (PCA), the 300-dimensional vectors were reduced to 2-dimensional vectors, which are shown in Figure 25. $204,158$ subclass-superclass pairs were retrieved from the set of relevant classes (Section 3.3). For each pair, the offset $\vec{\textit{superclass}} - \vec{\textit{subclass}} \in \mathbb{R}^{300}$ was computed using triple sentence word embeddings (Section 4.2.1). The offsets were clustered into 15 clusters using KMeans clustering. PCA was applied on the clustered offsets for plotting. The legend indicates the color of each cluster $k \in [0, 14]$. Table 1 shows the corresponding percentage of offsets in each cluster, as well as the type of offsets in the cluster.

The clusters 1 (9.9%), 6 (23.27%) and 13 (45.06%), which attribute to 78.23% of all offsets, contain a variation of different subclass-of relation types, which is most likely the case because 15 clusters are not sufficient to separate all different types of subclass-of relations. Therefore algorithms like piecewise linear projection [17], which exploit the similarity of specific types of subclass-of relations, need to use more clusters K . The other clusters show very specific types of offsets. It can be seen in Figure 25, that the clusters 2, 7 and 10 have very different orientations in respect to the origin $(0, 0)$ than the other clusters. Therefore directly using vector offsets would not be applicable, as offsets can not represent the complex subclass-of relation.

Fu et al. [17] reached a similar conclusion and proposes the use of linear projections as means to represent the subclass-of relation. Additionally, it can be concluded that 15 clusters is too low for usage in piecewise linear projection, since 3 clusters, representing 78.23% of all offsets, exist, which do not have specific topics. Therefore higher cluster sizes K should be considered as parameter for piecewise linear projection. In the following section, the approach by Fu et al., which is described in Section 2.7.3, is adjusted for usage in the hybrid algorithm.

4.3.3 Linear projection

Following the idea of using vector offsets to find the correct superclass $\vec{p} \in \mathbb{R}^d$ for a given unknown $\vec{u} \in \mathbb{R}^d$, the translation represented by the subclass offset can be extended by adding a linear projection matrix $\Phi \in \mathbb{R}^{d \times d}$. This follows the idea of Fu et al. [17], who uses linear projection without translation to find the superclasses for a given unknown. The superclass $\vec{p}$ for an unknown $\vec{u}$ is accordingly computed:

$$\vec{p} = \Phi\vec{u} + \vec{v} = \begin{bmatrix} \Phi & \vec{v} \end{bmatrix} \begin{bmatrix} \vec{u} \\ 1 \end{bmatrix} \tag{24}$$

where $\vec{v} \in \mathbb{R}^d$ is a translation vector. Combining the translation and projection into a single matrix $T = \begin{bmatrix} \Phi & \vec{v} \end{bmatrix}$ enables the use of Fu et al. [17]'s approach for finding $T^* \in \mathbb{R}^{d \times d}$, which is the best fit for the given subclass-superclass pairs. T^* can be computed by minimizing the following mean squared error [17]:

$$T^* = \arg\min_T \frac{1}{N} \sum_{(\vec{x},\vec{y})} \|T\vec{x} - \vec{y}\|^2 \tag{25}$$

k	perc.	topic	example
2	0.42%	cheeses	*Rigotte de Sainte-Colombe* $\lhd$ *cow's-milk cheese*
3	7.61%	badminton tournaments	*1987 Swiss Badminton Championships* $\lhd$ *badminton tournament*
4	0.32%	chemicals	*Actinomycin* $\lhd$ *chemical compound*
5	1.18%	diseases	*African swine fever* $\lhd$ *disease*
7	3.16%	locomotives, military aircrafts	*GWR 103 President* $\lhd$ *tender locomotive Heinkel He 50* $\lhd$ *military aircraft*
8	3.94%	Alcalde (Spanish mayors)	*mayor of Granollers* $\lhd$ *Alcalde*
9	0.41%	automobiles	*Acura TLX* $\lhd$ *automobile*
10	0.67%	food	*Sour cream doughnut* $\lhd$ *food*
11	1.8%	aircrafts	*Bell XH-15* $\lhd$ *aircraft*
12	1.22%	wine (mostly Italian wines)	*Greek wine* $\lhd$ *wine*, *Alghero frizzante rosato* $\lhd$ *wine*
14	0.42%	RNA	*Small nucleolar RNA SNORD86* $\lhd$ *RNA*
15	0.65%	ambassadors	*Ambassador of the Republic of China to the United States*

Table 1: Topics of subclass-of offset clusters (K=15); total $= 204,158$.

where N is the number of $(\vec{x}, \vec{y})$ subclass-superclass pairs in the training data [17].

Fu et al. [17] proposes an improvement to the linear projection. As shown in Figure 25, the subclass offsets appear in clusters, therefore it is intuitive to assume that training a matrix $T_k \in \mathbb{R}^{d \times d}$ for different clusters k increases the accuracy of the method [17]. However, the approach by Fu et al. [17] is not applicable for the given problem. In Fu et al.'s stated problem, a an unknown class can have zero, one, or multiple superclasses, but in the thesis' problem an unknown class is always assigned one superclass. Additionally, the input for the thesis' classification is a single unknown class, therefore it cannot be decided in which subclass-of offset cluster the unknown class belongs. Instead of using subclass-of clusters to train the piecewise projections, the subclass-superclass pairs $(\vec{x}, \vec{y})$ are clustered by their input vector $\vec{x}$. This leads to the following training objective for piecewise linear projections [17]:

$$T_k^* = \arg\min_{T_k} \frac{1}{N_k} \sum_{(\vec{x},\vec{y}) \in C_k} \|T_k \vec{x} - \vec{y}\| \tag{26}$$

where $N_k \in \mathbb{N}$ is the amount of word pairs in the k^{th} cluster $C_k = \{(\vec{x_i}, \vec{y_i}) | \vec{x_i}$ *in cluster* $k, i \in [1, N]\}$ [17]. Both training objectives describe multivariate linear regression tasks. Fu et al. [17] uses stochastic gradient descent (SGD) to solve the objective. SciPy's SGD regressor [27] is used to train the linear projection matrices.

The projection for a given unknown $\vec{u}$ using piecewise linear projection can be

computed as follows:

$$\vec{p} = T_k^* \begin{bmatrix} \vec{u} \\ 1 \end{bmatrix} \tag{27}$$

where $\vec{u}$ in cluster k. The appropiate superclass is subsequently the closest class embedding to the computed projection.

The thesis' evaluation will show whether linear projection is superior to the distance-based kNN in the given task, and to what degree it benefits from word embeddings using graph walk sentences instead of triple sentences.

4.3.4 Non-linear regression via neural nets

The linear projection approach uses linear regression to predict a superclass embedding from a given unknown embedding. The complexity of the subclass-of relation is simplified by clustering the embeddings and computing a linear regression for each cluster. However, it is possible that a linear regression is not sufficient for capturing all correlations between the input and output embeddings. Leshno et al. [31] states that a neural network with a non-polynomial activation function is able to approximate any function. Therefore it is possible to use a neural network to implement a non-linear regression, which is able to better model the subclass-of relation. The expressiveness of such a network is dependent on the number of neurons and hidden layers of the network [24]. Increasing the number of layers and neurons can theoretically maximize the expressiveness of a deep neural network [46], but also increases the memory usage and runtime, as well as the required amount of training data necessary to train the model [46]. In the following a multi-network model for non-linear multi-variate, multi-target regression is proposed.

The rectifier is well-suited as an activation function for non-linear regression, as it has multiple favorable characteristics in comparison to the classically used sigmoid and tanh functions [36] [19]. The rectifier is defined as a maximum function $f(x) = \max(0, x)$, which either returns the input or zero. Glorot et al. [19] states that this function allows the network to learn sparse representation of inputs, since only a subset of neurons are activated for each input, because the rectifier returns 0 for all negative inputs. The remaining active neurons describe a linear function, which prevents gradient vanishing [19] and makes computations and backpropagation cheaper [19] [36].

Neurons using rectifier as activation are called Rectified Linear Units (ReLU). The output of a densely connected layer of ReLU neurons is computed, as follows:

$$ReLU(\vec{x}) = \max(W\vec{x} + b, 0) = \begin{cases} W\vec{x} + \vec{b} \text{ for } W\vec{x} + b > 0 \\ 0 \text{ else} \end{cases}$$

where $\vec{x}$ is the input, the output, W the input edge weights and $\vec{b}$ the corresponding bias.

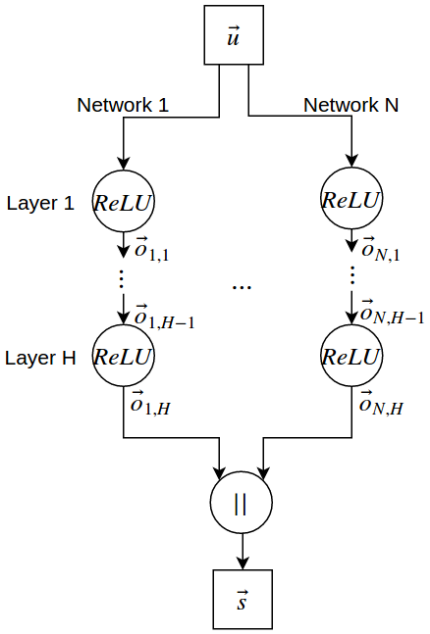

Figure 26: Multi-network regression model

Regression tasks involve predicting a single target value from a given input vector. The thesis' tasks however requires the prediction of a d-dimensional vector from a given d-dimensional vector, where d is the dimension of the word embeddings. The recommended configuration, as stated in Section 4.1, sets d as 300. Training 300 networks to predict each of the single targets to a suitable accuracy requires either a high memory capacity to train 300 networks in parallel or a long training time to train the networks sequentially. A compromise can be reached by training a smaller number of networks to predict multiple targets at once and then combining the results into the final d dimensional output.

The multi-network model (see Figure 26) consists of N hidden deep neural networks with H ReLU layers (including input and output layer). Each hidden network predicts $o = \frac{d}{N}$ targets of the complete target vector. The output of each hidden network is concatenated, which results in the model's d dimensional output.

The model is defined by the following parameters:

- $N \in \mathbb{N}$: number of hidden networks
- $H \in \mathbb{N}$: number of layers per hidden network
- $d \in \mathbb{N}$: dimension of word embeddings (input and output dimensions of full model)
- $h \in \mathbb{N}$: dimension of hidden layers of hidden networks (number of neurons in each hidden layer)

- $o = \frac{d}{N} \in \mathbb{N}$: output dimension of hidden networks

The trainable variables are the weights and biases of the fully connected layers of the hidden networks and the output layer, which are defined as follows:

- $W_{i,1} \in \mathbb{R}^{h \times d}, i \in \{1, \cdots, N\}$: input weights of hidden networks, maps word embeddings to hidden layer, with corresponding biases $\vec{b}_{i,1} \in \mathbb{R}^h$.
- $W_{i,j} \in \mathbb{R}^{h \times h}, i \in \{1, \cdots, N\}, j \in \{2, \cdots, H-1\}$: weights between hidden layers with corresponding biases $\vec{b}_{i,j} \in \mathbb{R}^h$.
- $W_{i,H} \in \mathbb{R}^{o \times h}, i \in \{1, \cdots, N\}$: output weights for hidden networks with corresponding biases $\vec{b}_{i,H} \in \mathbb{R}^o$.
- $W_{con} \in \mathbb{R}^{d \times d}$: concatenation output layer with corresponding bias $\vec{b}_{con} \in \mathbb{R}^d$.

Let $\vec{u} \in \mathbb{R}^d$ be the input word embedding and $\vec{s} \in \mathbb{R}^d$ the predicted superclass embedding. $\vec{s}$ is computed from $\vec{u}$, using $ReLU$ as activation, as follows:

$$
\begin{aligned}
\vec{v}_{i,1} &= \max(0, W_{i,1} \cdot \vec{u} + \vec{b}_{i,1}) \\
\vec{v}_{i,j} &= \max(0, W_{i,j} \cdot \vec{v}_{i,j-1} + \vec{b}_{i,j}) \text{ for } j \in \{2, \cdots, H-1\} \\
\vec{v}_{i,H} &= \max(0, W_{i,H} \cdot \vec{v}_{i,H-1} + \vec{b}_{i,H}) \\
\vec{s} &= W_{con} \cdot \begin{bmatrix} \vec{v}_{1,H} & \cdots & \vec{v}_{N,H} \end{bmatrix} + \vec{b}_{con}
\end{aligned}
$$

with $i \in \{1, \cdots, N\}$.

The proposed network has a higher expressiveness than the previously described linear projection approach. The expressiveness of the model can be experimentally optimized by adjusting the different hyperparameters like depth and width of the hidden networks [46] [24]. Additionally the network uses rectifier as activation function, which is non-polynomial, and therefore, according to Leshno et al. [31], able to model an arbitrary function. In comparison, the linear projection approach can only model a linear function and has limited reconfigurability with only one hyperparameter, the number of clusters. Therefore it is predicted that the non-linear neural regression model will perform better than linear projection.

5 Evaluation

5.1 Method

The hybrid algorithms are evaluated using a gold standard. The gold standard is fetched from the set of relevant classes (Section 3.3) in the 2016/11/07 Wikidata dump [3]. $208,502$ classes and their corresponding superclasses were retrieved.

The gold standard test set G consists of sample tuples (c_i, S_i), where $c_i \in R$ is a subclass and $S_i = succ(c_i)$ is the set of direct superclasses of c_i, where R is the set of relevant classes and $i \in [1, |G|]$. Because it is assumed that every class, except the root class, should have a superclass, $|S_i| > 0$ applies for all gold samples (c_i, S_i) with $i \in [1, |G|]$. Therefore the computation of the F_1-score as an evaluation measure is applicable, because only correct and false classification can occur. Instead the first applied evaluation measure is the accuracy acc_{alg}, which is calculated as follows for an algorithm alg:

$$acc_{alg} = \frac{1}{|G|} \sum_{i=1}^{|G|} \begin{cases} 1, & \textit{for } p_{alg}(c_i) \in S_i \\ 0, & \textit{for } p_{alg}(c_i) \notin S_i \end{cases} \tag{28}$$

where $p_{alg} : R \mapsto R$ is the prediction function of alg, which returns the predicted superclass of alg given an unknown class.

However, accuracy alone is not sufficient in evaluating a classification in a hierarchical context like taxonomies [29]. Accuracy ignores the taxonomy and other possible dimensions, as it only measures how often the algorithm is able to correctly guess the superclass. Dellschaft and Staab states that a good measure should "allow[s] to evaluate an ontology along multiple dimensions" [13]. Another dimension, next to the accuracy of the algorithm, which can be observed is the taxonomic relation between the predictions and their corresponding gold standards. Given two algorithms $algA$ and $algB$ with equal accuracies $acc_{algA} = acc_{algB}$, assume $algB$ predicts either correctly or predicts the root class, while $algA$ predicts either correctly or overspecializes by predicting the subclass of the gold standard. Which one of the two algorithms performs better? If only the accuracy is considered, both algorithms are equally good and it would not matter, which algorithm would be chosen for practical application. But in practice $algA$ would be preferred, since even false predictions are useful to some degree, while $algB$'s false predictions are not usable. Therefore the taxonomy dimension should be included in the evaluation. For this the **taxonomic overlap** measure, defined by Dellschaft and Staab [13], is also used in the evaluation. The taxonomic overlap counts how many superclasses are shared by the prediction and the gold standard for each sample. This is called the **semantic upwards cotopy** sc defined as follows [13]:

$$sc(c, T) = \{c_i | c_i, c \in C \wedge c \lhd_T c_i\} \tag{29}$$

where $T = (C, \_)$ is a taxonomy and $c \in C$ is a class in the taxonomy T. Based on the semantic upwards cotopy, the taxonomic overlap to_{sc} between classes c_1 and c_2

in C is defined as follows [13]:

$$to_{sc}(c_1, c_2, T) = \frac{|sc(c_1, T) \cap sc(c_2, T)|}{|sc(c_1, T) \cup sc(c_2, T)|} \tag{30}$$

The taxonomic overlap between two classes ranges between 0, if the classes have no common superclasses, and 1 if the classes are identical or all direct superclasses are identical. To apply the taxonomic overlap in the evaluation the overlaps for all prediction-gold pairs have to summed up, which leads to the **average taxonomic overlap** $ato_{sc,alg}$ for an algorithm alg:

$$ato_{sc,alg} = \frac{1}{|G|} \sum_{i=1}^{|G|} \max_{s \in S_i} (to_{sc}(p_{alg}(c_i), s, T)) \tag{31}$$

where G is gold standard, which samples (c_i, S_i) with $i \in [1, |G|]$ and p_{alg} the prediction function of an algorithm alg. The maximum overlap between prediction and gold is chosen, if there are multiple possible superclass for a sample, since the taxonomic overlap is a function, which should be maximized by the algorithm [29].

A trivial **baseline** algorithm is implemented. Evaluated algorithms should yield better accuracies and taxonomic overlaps than the baseline. The baseline algorithm is the **most-frequent** classifier, which assigns the most frequently occurring class from the training data to all unknowns. Due to the characteristics of the problem statement, no comparable state-of-the-art methods could be found at the time of writing. Therefore no state-of-the-art methods could be used as baseline, which would be preferable.

Each possible combination of components is evaluated as a separate hybrid algorithm. For each classification component different hyperparameters are tested to find a good configuration. A naming scheme is used to uniquely identify each hybrid algorithm with their corresponding hyperparameters. Because the SGNS configuration is constant over all hybrid algorithms, an algorithm combination can be identified by their classification component, as well as their classification hyperparameters. Consequently, hybrid algorithms have the following name format:

<classification>(<hyperparameters>)

To shorten the identifiers, the following abbreviations for components are used:

component	abbreviation
most-frequent classifier	baseline
distance-based kNN	distknn
piecewise linear projection	linproj
non-linear regression multi-network	concatnn

Table 2: Abbreviations for components

Algorithms are implemented as described in Chapter 4. The non-linear regression multi-network model is called concatnn due to its concatenation layer.

The following hybrid algorithms are evaluated:

- baseline
- distknn(k =?) with $k = 5, 10, 15, 20$
- linproj(c =?) with $c = 1, 25, 50$
- concatnn(net =?, $h = 3, n = 1200$) with $k = 1, 10, 20$

where k, c, net, h, n are hyperparameters of classification algorithms. For kNN, k is the number of nearest neighbors. For linear projection, c is the number of clusters used. For non-linear regression multi-network model, net is the number of hidden networks, h is the number of hidden layers for each hidden network, and n is the number of neurons per hidden layer.

By varying hyperparameters for classifiers, their influence on the algorithm's performance can be experimentally analyzed, and previously stated hypothesis' can be proven.

It can be answered whether clustering is important to linear projection and how many clusters are required to optimize the performance of linear projection. In Section 4.3.2, it is observed that more than 15 clusters are needed to fully cluster the subclass offsets. Therefore it is assumed that more than 15 clusters are needed to achieve a good performance with linear projection.

Performance of the multi-network model should be correlated to the depth, width and number of networks in the model. Existing work has already proven that width and depth can increase the performance of networks given an appropriate amount of training data [46] [24]. Therefore the influence of the parameters h and n are not further explored. The design decision to use multiple separate networks to compute the non-linear regression should be evaluated. This is done by comparing three models with different numbers of hidden networks.

5.2 About the dataset

The Wikidata dump from 2016/11/07 [3] was used in the evaluation The gold standard is retrieved from the set of relevant classes (Section 3.3). A total of $178,771$ gold sample tuples (t_i, S_i) was retrieved. $10,000$ of the samples are used for testing, the $168,771$ remaining samples are used for training.

Figure 27 depicts the extraction of gold samples from the Wikidata taxonomy. A subclass, e.g. *province of Ireland (Q202156)*, and its direct superclasses are retrieved from Wikidata. The numeric item ids are extracted from these classes, e.g. *Q202156* $\mapsto 202156$, and yields the gold sample $(202156, \{3356092, 1620908, 34876\}) \in G$, where G is the set of all gold test samples.

The most frequent class in the training data set is *badminton tournament (Q13357858)*. This is consistent with the subclass-of offset analysis (Section 4.3.2), which has shown

that offsets corresponding to *badminton tournament* are the most frequent offsets. The baseline will therefore assign *badminton tournament* to all test samples.

A total of $109,267,818$ triple sentences were extracted from the Wikidata dump [3]. All entities, which have irrelevant properties, as defined in Section 3.3, were excluded. All sentences representing test sample subclass-of relations were also excluded. Figure 27 shows an example extraction of a gold standard sample from Wikidata.

Due to memory limitations, SGNS is only able to capture a certain vocabulary size. Therefore only $5,000,000$ words are represented by SGNS. Less frequent words have therefore no embeddings. This also affects a small number of relevant classes, which occur in the test samples. Therefore only $8,943$ out of $10,000$ test samples can be used in the evaluation.

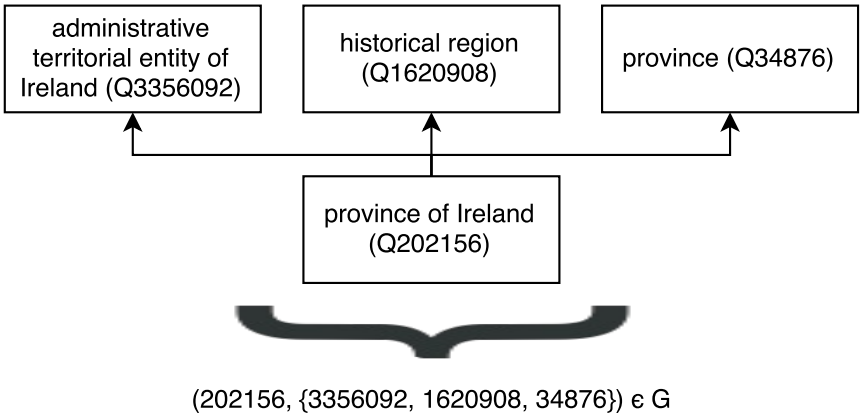

Figure 27: Example for creating a gold standard sample.

5.3 Results

The hybrid algorithms with the previously described parameters were executed. The accuracies and average taxonomic overlaps are listed below in Table 3:

algorithm	accuracy	taxonomic overlap
baseline	8.8%	31.26%
distknn ($k = 5$)	21.38%	45.27%
distknn ($k = 10$)	22.26%	46.17%
distknn ($k = 15$)	22.62%	46.56%
distknn (k = 20)	**22.83%**	**47.10%**
linproj ($c = 1$)	8.54%	20.04%
linproj ($c = 25$)	11.90%	26.68%
linproj ($c = 50$)	13.09%	28.65%
concatnn ($net = 1, h = 3, n = 1200$)	%	%
concatnn ($net = 10, h = 3, n = 1200$)	18.79%	34.92%
concatnn ($net = 20, h = 3, n = 1200$)	18.92%	36.46%

Table 3: Evaluation results for all hybrid algorithms. Best performing hybrid algorithm is highlighted.

Figure 28 displays the accuracy (green) and taxonomic overlap (blue) of the best performing algorithm for each group with distinct classification component. distknn($k = 20$) is the best performing algorithm with an accuracy of 22.83% and an overlap of 46.56%. distknn algorithms perform better than the regression-based methods, linproj and concatnn. linproj($c = 50$) is the best linear projection algorithm with an accuracy of 13.09% and an overlap of 28.65%. concatnn($net = 20, h = 3, n = 1200$) is the best multi-network model algorithm with an accuracy of 18.92% and an overlap of 36.46%.

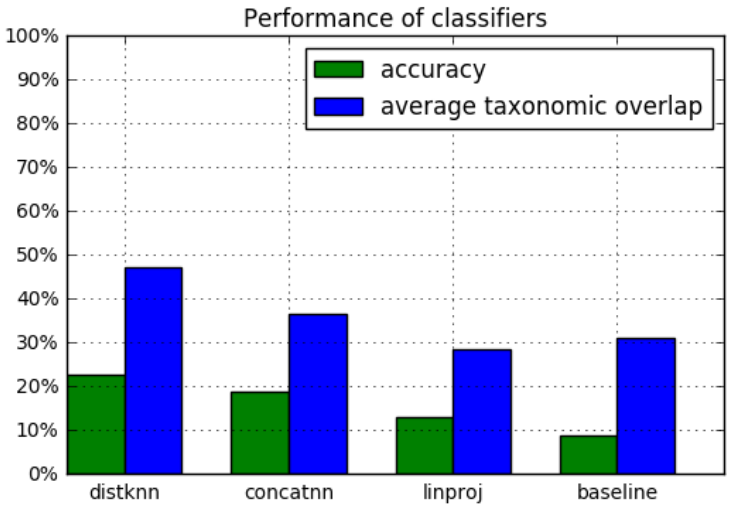

Figure 28: Comparison of best performing classifier of each method: distknn($k = 20$), linproj($c = 50$), concatnn($net = 20, h = 3, n = 1200$)

As stated in Section 5.1, accuracy and taxonomic overlap are to some degree correlated. For the baseline, taxonomic overlap equals ≈ 4 times the accuracy, while for all other algorithms the taxonomic overlap equals ≈ 2 times the accuracy. Additionally, taxonomic overlap for linear projection algorithms is lower than the baseline. This implies that the implemented algorithms tend to place an unknown class far from its correct position, if a misclassification occurs. Observing the local taxonomic overlaps between all prediction and gold standard pairs supports this hypothesis. Figure 29 shows the local taxonomic overlap histogram for the best performing algorithms of each classification component and the baseline.

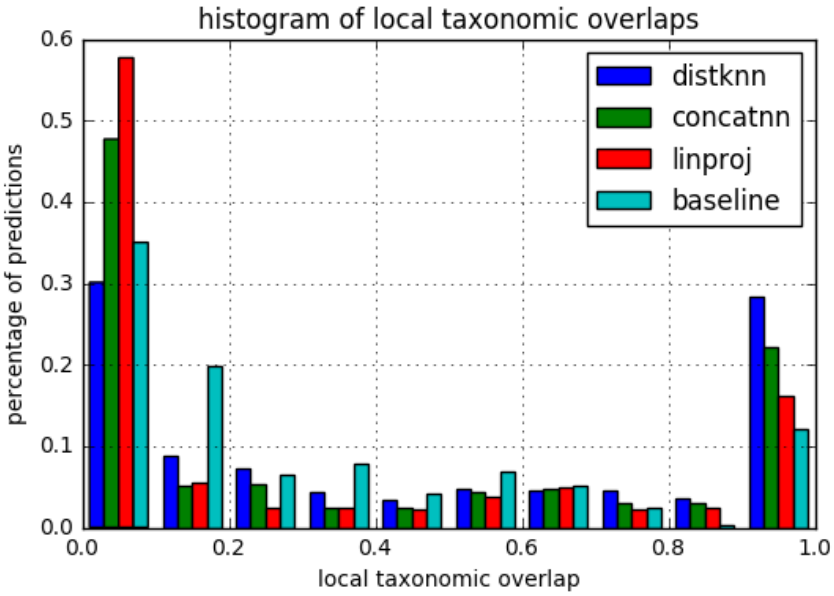

Figure 29: Comparison of local taxonomic overlaps for best performing classifier of each method. distknn = distknn($k = 20$), linproj = linproj($c = 50$), concatnn=concatnn($net = 20, h = 3, n = 1200$)

It can be seen that classes are either correctly classified and thereby have a local taxonomic overlap of 1 or are completely misplaced and have a local taxonomic overlap between 0 and 0.1. All methods are able to place $\approx 5\%$ of classes very close to the correct position in the case of misclassification, since the values in the bucket of 0.9 to 1.0 are $\approx 5\%$ higher than the corresponding accuracy for each algorithm.

Distance-based kNN is a superior classification component in comparison to the regression-based methods. The regression task requires the prediction of multiple targets, while kNN only has to predict a single target, which is the label. The corresponding label of the regression is the closest superclass embedding to the predicted embedding. Intuitively, the regression-based algorithms have a higher complexity and are therefore more susceptible to errors. For example, overfitting may occur for the multiple regression, if not all values in the embedding are required to represent

the target fully. The corresponding coefficients could in this case randomly affect the target results, and thereby worsen the performance [22].

Non-linear regression via neural networks may still be a promising approach, if techniques like drop-out or trimming are used. These methods reduce the number of active neurons in the network and thereby are able to prevent overfitting [51].

Because all evaluated algorithms ignore the taxonomy in the classification process, it is unsurprising that the taxonomic overlaps are low. Future algorithms may use taxonomy-based classification approaches, which were mentioned in Section 2.7.2. Using these classifiers as classification approaches is likely to result in higher taxonomic overlaps and may improve the accuracy.

5.4 Influence of hyperparameters on classification

Choosing good hyperparameters is critical for producing good embeddings with SGNS [32]. Following this notion, choosing good hyperparameters may also be important in the classification process.

Distance-based kNN's hyperparameter k is the number of nearest neighbors. Increasing k shows only little improvements of accuracy and taxonomic overlap, which is shown in Figure 30. Because class embeddings is very clustered as shown in Section 4.3.2, an unknown class is likely to belong to a cluster. Increasing k would either add classes of the same cluster to the neighbors, which does not change the classification decision, or add classes from different distant clusters to the neighbors, which would also not change the decision, because weights are distance-based.

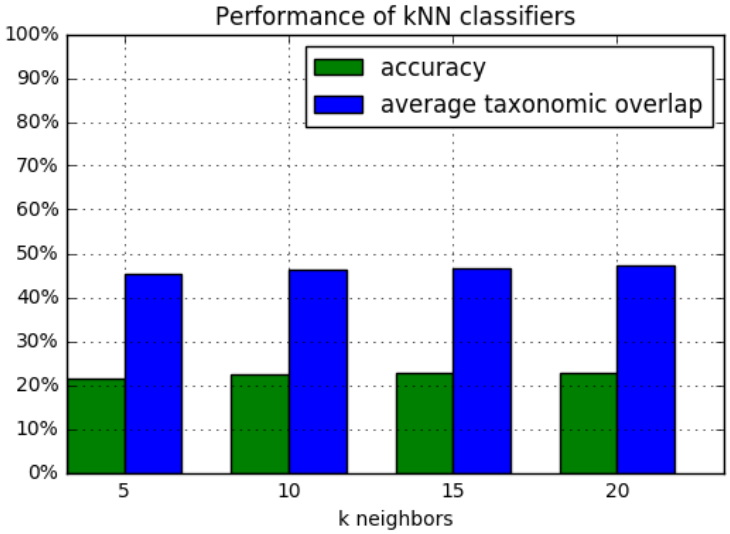

Figure 30: Comparison of kNN classifier with different neighbors $k = 5, 10, 15, 20$)

Number of clusters c is the hyperparameter for linear projection. Figure 31 shows the accuracy and taxonomic overlap for different cluster numbers c.

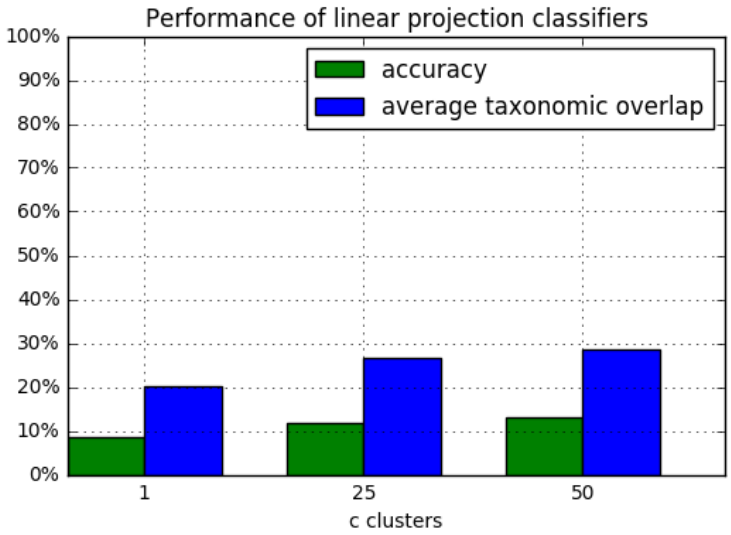

Figure 31: Comparison of linear projection classifier with different cluster counts $c = 1, 25, 50$)

As previously predicted, increasing the number of clusters improves the performance of the algorithm. Linear projection with $c = 1$ performs worse than the baseline algorithm, which implies that the subclass-of relation is not linear. This follows the argumentation by Fu et al. [17] and the observed clustering of subclass offsets in Section 4.3.2.

TODO: neural network hyperparameters

5.5 Discussion

The problem of classification with a very high number of labels ($> 100,000$) is very difficult. Comparable state-of-the-art methods were not found at the time of writing. Therefore the trivial most-frequent classifier was used as baseline. The executed evaluation is able to at show, whether the proposed solutions are able to support users in enriching Wikidata's taxonomy and which of the implemented algorithms should be used in practical application.

With an accuracy of 22.83% in testing, distknn($k = 20$) is able to correctly predict the superclass of an orphan class in approximately 1 out of 4 cases. distknn($k = 20$) outperforms the baseline by 14.03%. Based on these results, it is concluded that the proposed solutions are able to support a user in enriching Wikidata's taxonomy by providing suggestions for given orphan classes and if a practical application is implemented to provide such suggestions distknn($k = 20$) should be used, as it is the best performing algorithm.

The following problems could be identified, which future work could attempt to solve.

The implemented linear projection classifier is flawed. Because subclass-superclass pairs are clustered by subclasses and each subclass may have multiple superclasses, different types of subclass-of relations are grouped into one cluster. This leads to problems in training the corresponding projection, since dissimilar offsets have to represented by a single projection and therefore the projection cannot converge to an optimal solution. In the thesis, this is solved by ignoring additional superclasses. Instead clustering by subclass-of offsets, as done by Fu et al. [17], could provide better results, if an effective method for identifying the correct cluster for an unknown class can be found. Since subclass-of clusters are related to single topics as shown in Section 4.3.2, it may be feasible to identify the correct cluster for an unknown class by analyzing its properties and matching it to the most common properties in a subclass-of cluster.

The problem statement is too difficult. A classification problem with over $100k$ labels is difficult to solve. Approaches in reducing the number of possible superclasses should be considered. Most classes have a distance of 6 to 10 to the root class (Section 3.1), therefore it may be applicable to only consider classes with distance 5 to 9 as superclasses. A topic-based reduction of possible superclasses may have benefits, since classes with similar topics are closely related in the taxonomy, it may be sensible to only observe a certain excerpt of the taxonomy for each classification decision. This approach would however require a method of clustering the taxonomy into topic-based subgraphs.

6 Conclusion and future work

TODO: complete review of conclusion

In this thesis, hybrid algorithms using neural word embeddings for taxonomy enrichment were presented. The enrichment task was modeled as a classification task with over $100,000$ labels. The taxonomy was enriched by adding new subclass-of relations between orphan classes and classes in the root taxonomy. The developed hybrid algorithm consisted of three sequential components. SequenceGen, which generates sequences given a data source, e.g. Wikidata. A Word2Vec neural network [39], specifically SGNS, is trained on the SequenceGen output to produce embeddings for all relevant classes. Finally, a classification component, trained on subclass-superclass pairs, exploits the characteristics of word embeddings to classify orphan classes. Triple sentences and graph walk sentences were implemented as SequenceGen components. **TODO: update based on evaluation results** Due to the fact that graph walks increase the amount of context given to each word, the performance of hybrid algorithm using graph walk sentences was better. kNN and linear projection were implemented as classification component. As shown by the evaluation, the linear projection approach is not suited to the given difficult classification task. kNN outperformed linear projection significantly.

Future work in regards to improving the algorithm for the given classification task would have multiple venues to explore. Other approaches for generating embeddings may be beneficial. Instead of a simple feedforward model implemented by SGNS, recurrent or deep neural networks could provide more effective embeddings [5] [25] [38]. Deep neural graph embeddings may either replace or enrich the word embeddings generated by other models [7].

The use of word embeddings in ontology learning is promising. Subclass-of relations in a taxonomy can be represented by embedding offsets. Different types of subclass-of relations, which are topically related, exist in Wikidata's taxonomy. The successful use of kNN also shows that embeddings effectively represent classes, since similar classes were grouped close together. Further exploration on how embeddings can represent entities in knowledge bases could be beneficial. For example, the thesis' approach could be adjusted for the classification of instances and potentially other more complex relations, e.g. *occupation (P106)*.

Wikidata's taxonomy was analyzed in the thesis. Insights for possible future work with Wikidata was gained. Automated mapping of other specialized knowledge bases like the Entrez Gene knowledge base skews the distribution of classes. Classes, which are relevant to human editors, have a low share of $\approx 15\%$ in the taxonomy. Future work on Wikidata should therefore consider removing every entitity, which has undesirable properties, as a measure for improving validity of evaluation results. An incomplete list of undesirable properties in regards to Wikidata's taxonomy is given in Section 3.3. The taxonomy is in a good state, since most classes ($\approx 97\%$) are in the root taxonomy. Additionally, it can be argued that the constant curation by human editors improves the quality of content in Wikidata. Therefore

using Wikidata's taxonomy as an easy to retrieve gold standard may be applicable for 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] Marco Baroni and Georgiana Dinu. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, 1:238–247, 2014.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] Fariz Darari, Simon Razniewski, Radityo Eko Prasojo, and Werner Nutt. *Enabling Fine-Grained RDF Data Completeness Assessment*, pages 170–187. Springer International Publishing, Cham, 2016. ISBN 978-3-319-38791-8. doi: 10.1007/978-3-319-38791-8_10. URL http://dx.doi.org/10.1007/978-3-319-38791-8_10.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Re-

search - Workshop and Conference Proceedings, 2011. URL http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf.

[20] 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.

[21] 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.

[22] D. M. Hawkins. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. URL http://pubs3.acs.org/acs/journals/doilookup?in_doi=10.1021/ci0342472.

[23] 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.

[24] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL http://www.sciencedirect.com/science/article/pii/089360809190009T.

[25] Alexander G. Ororbia II, Tomas Mikolov, and David Reitter. Learning simpler language models with the delta recurrent neural network framework. *CoRR*, abs/1703.08864, 2017. URL http://arxiv.org/abs/1703.08864.

[26] Ali Ismayilov, Dimitris Kontokostas, Sören Auer, Jens Lehmann, and Sebastian Hellmann. Wikidata through the eyes of dbpedia. *Semantic Web Journal*, pages 1–11, 2017. doi: 10.3233/SW-170277.

[27] 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].

[28] 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.

[29] 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.

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

[31] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL http://www.sciencedirect.com/science/article/pii/S0893608005801315.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[37] 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.

[38] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1045–1048. ISCA, 2010. URL http://dblp.uni-trier.de/db/conf/interspeech/interspeech2010.html#MikolovKBCK10.

[39] 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.

[40] 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.

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

[42] 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.

[43] 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.

[44] 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.

[45] Tim E Putman, Sebastien Lelong, Sebastian Burgstaller-Muelhbacher, Andra Waagmeester, Colin Diesh, Nathan Dunn, Monica Munoz-Torres, Gregory Stupp, Andrew Su, and Benjamin M Good. Wikigenomes: an open web application for community consumption and curation of gene annotation data in wikidata. *bioRxiv*, 2017. doi: 10.1101/102046. URL http://biorxiv.org/content/early/2017/01/24/102046.

[46] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural net-works. 2017. URL https://arxiv.org/pdf/1606.05336.pdf.

[47] 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.

[48] 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.

[49] 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.

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

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

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

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

[54] 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.

[55] 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.