Fachbereich 4: Informatik

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<br>Institute for Web Science and Technologies |
| Zweitgutachter: | Dr. Mahdi Bohlouli<br>Institute for Web Science and Technologies |

Koblenz, im Februar 2017

## Erklärung

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

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

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

(Ort, Datum) (Unterschrift)

# Contents

## List of Figures

# 1 Introduction

**TODO: Motivation. Related work. Justify novelty. Solution. Evaluation.**

The thesis is structured as follows. **TODO: apply changes of outline to this paragraph** Section 2 defines the concepts related to Wikidata, taxonomies, similarity measures and k-nearest-neighbors classification. Subsequently, the problem statement is formalized and its corresponding challenges listed. Section 2.6 explains the notion of neural networks at the example of a simple feedforward network with backpropagation. Subsequently, neural networks for graph and word representation are compared in regards to suitability in the problem's context. Section 2.7 compares the problem, solved by the thesis, to related work in the field of ontology learning. The problem is classified and solutions to similar problems analyzed. The novelty of the work is justified. Section 3 presents the results of the analysis of Wikidata's taxonomy. Characteristics of unlinked classes are identified, and a specific subset of unlinked classes chosen, which fulfills necessary requirements as input for an algorithm. Section 4 describes the developed baseline algorithm, which combines Word2Vec by Mikolov et al. [25] and k-nearest-neighbors. Possible variations of the algorithm are presented and compared in regards to possible gains and problems. One of these variations is also implemented for evaluation purposes. Section 5 describes the evaluation methodology and presents the results. The baseline algorithm, using multiple different sets of different hyperparameters, and a variation of it will be compared. Section 6 summarizes the results of the thesis and lists possible future work, which could extend upon it.

# 2 Background and related work

## 2.1 Wikidata

Wikidata is an open, free, multilingual and collaborative knowledge base. It is a structured knowledge source for other Wikimedia projects. It tries to model the real world, meaning every concept, object, animal, person, etc. Wikidata is mostly edited and extended by humans, which in general improves the quality of entries compared to fully-automated systems, because different editors can validate and correct occurring errors. However, Wikidata, like most knowledge bases, is incomplete and therefore has to be operated under the *Open World Assumption* (OWA). OWA states that if a statement is not contained in a knowledge base, it is not necessarily false but rather unknown [15].

In Wikidata items and properties exist. Items are the aforementioned concepts, objects, etc. While properties are used to make claims about items, e.g. *photographic film (Q6293)* is a *subclass of (P279) data storage device (Q193395)* (see Figure 1). 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 to make items and properties multi-

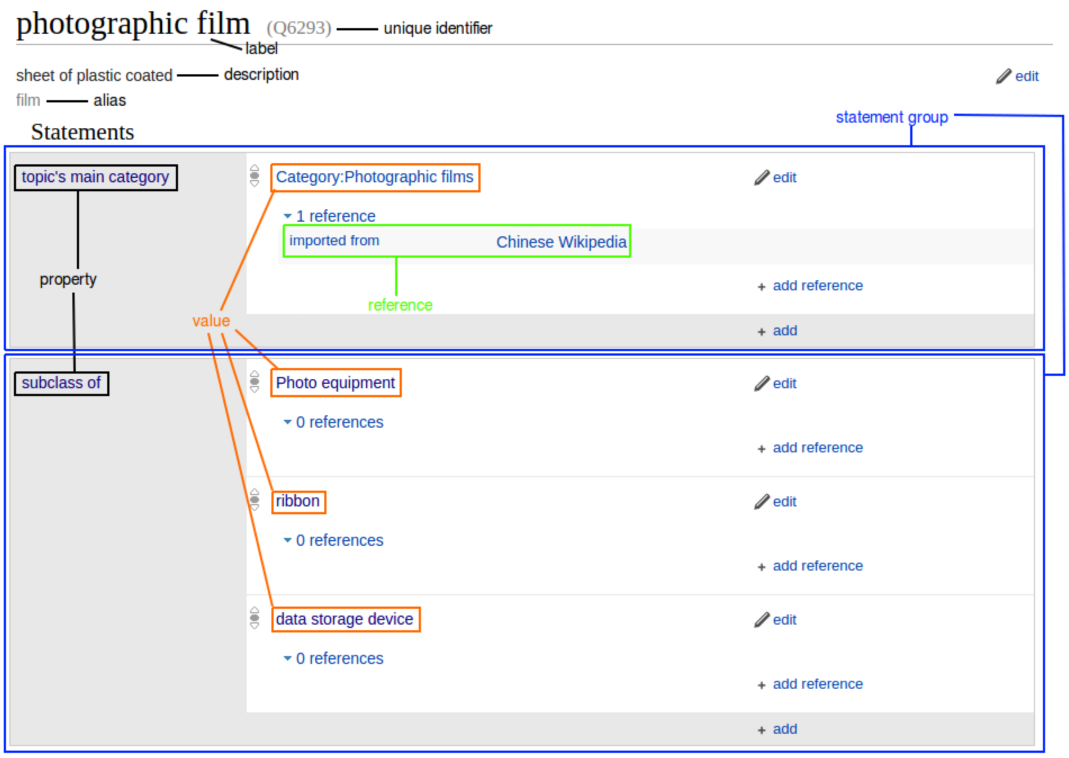

Figure 1: Example of Wikidata class: photographic film (Q6239)

lingual.

Items consist of labels, aliases and descriptions in different languages. Sitelinks connect items to their corresponding pages of Wikimedia projects like Wikipedia articles. Most importantly item are described by statements. Statements are in their simplest form a pair of property and value, assigned to a specific item. A value is either a literal value or another item. It should be noted that an item can have multiple statements with the same property. The set of statements with the same property is called statement group. Statements can be annotated with qualifiers, which specify the context of the statement, e.g. population at a certain point of time. Additionally, references can be used for statements to include its source. See Figure 1 for an example of a Wikidata item.

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

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

- $id \in \mathbb{N}$ is the numerical item ID;
- $label \in String$ is the English label of the item;
- $aliases \in \mathcal{P}(String)$ is the set of English synonyms for the label;

- $\mathit{description} \in \mathit{String}$ is a short sentence describing the item;
- $\mathit{sitelinks} \in \mathit{String} \times \mathit{String}$ is a set of tuples $(\mathit{site}, \mathit{title})$, where $\mathit{site}$ refers to a specific site of the Wikimedia projects, e.g. enwiki, and $\mathit{title}$ is the corresponding article title of the item on this site.

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

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

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

**Definition 3** (Class)**.** Given a set of items $I$ and a set of statements $R$. $c = (\mathit{classid}, \_, \_, \_, \_) \in I$ is a class, if at least one of the following assertions are true:

$$\begin{aligned}
&\exists i = (\mathit{instanceid}, \_, \_, \_, \_) \in I\ \exists s = (\mathit{itemid}, \mathit{pid}, \mathit{value}, \_, \_) \in R : \\
&\qquad \mathit{instanceid} = \mathit{itemid} \land \mathit{pid} = 31 \land \mathit{value} = \mathit{classid} \text{ (has instance)} \\
&\exists s = (\mathit{itemid}, \mathit{pid}, \_, \_, \_) \in I : \mathit{itemid} = \mathit{classid} \land \mathit{pid} = 279 \text{ (is subclass)} \\
&\exists i = (\mathit{subclassid}, \_, \_, \_, \_) \in I\ \exists s = (\mathit{itemid}, \mathit{pid}, \mathit{value}, \_, \_) \in R : \\
&\qquad \mathit{itemid} = \mathit{subclassid} \land \mathit{pid} = 279 \land \mathit{value} = \mathit{classid} \text{ (has subclass)}
\end{aligned}$$

_ is used as an anonymous placeholder, for the purpose of not naming unused elements in tuples. For example, *photographic film (Q6293)* (Figure 1) 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" [24]. Additionally it is possible for ontologies to contain axioms used for validation and constraint

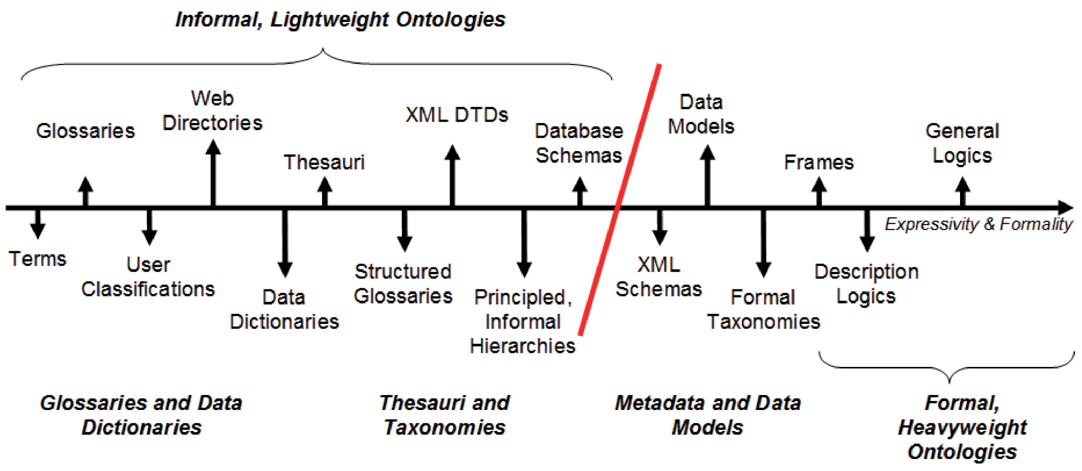

Figure 2: The spectrum of ontology kinds. [36]

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 [17]. In comparison, a knowledge base like Wikidata can be seen as an instantiation of such an ontology, since every knowledge base has to be conceptualized by an ontology [36]. Different types of ontologies can grouped by their level of formality and expressiveness. Wong et al. [36] differentiates ontologies as lightweight and heavyweight ontologies (see Figure 2). *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 [2].

For the purpose of developing a formal definition of the thesis' problem statement the notion of taxonomy needs to be formalized. Cimiano [8] defines a heavyweight ontology, which includes a taxonomy, as follows:
"

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

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

consisting of

- four disjoint sets $C$, $R$, $\mathcal{A}$, and $\mathcal{T}$ whose elements are called concept identifiers, relation identifiers, attribute identifiers and data types, respectively,
- a semi-upper latice $\leq_C$ on $C$ with top element $root_C$, called concept hierarchy or taxonomy,
- 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 $\sigma_{\mathcal{A}} : \mathcal{A} \rightarrow C \times \mathcal{T}$, 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)
$\forall x x \leq \mathit{top}$ (top element)
$\forall x \forall y \exists z(z \geq \wedge z \geq y \wedge \forall w(w \geq x \wedge w \geq y \implies w \geq z))$ (supremum)

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

A taxonomy can be modeled as a semi-upper lattice. This induces two important assumptions about the structure and to some degree completeness of the observed taxonomies. First, there is only one *root class*, top element of the lattice, of which every other class is (transitively) a subclass. Second, because of the supremum property, the taxonomy is fully connected, which means each class, but the root class, has a superclass. Wikidata's taxonomy does therefore not fulfill the definition by Cimiano [8], as it is not fully connected.
In the following, definitions wil bel presented, which attempt to model an incomplete taxonomy based on the already presented data model and structure of Wikidata. Refer to Appendix A for the necessary definitions on graphs.

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

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

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

If the taxonomy defined by Cimiano [8] is mapped on this graph-based taxonomy model, the following assumption is true, for $T = (C, S)$:

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

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

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

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

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

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

In Wikidata, the root class is *entity (Q35120)* [3]. All other classes, which are not subclasses of *entity (Q35120)*, are therefore either orphan classes, or subclasses of orphan classes. In Section 3, it will be shown that the Wikidata taxonomy graph is not fully connected. But the component, which contains the root class *entity (Q35120)*, contains 97% of all classes. This component will be referred to as *root taxonomy* in later sections.

## 2.3 Similarity

For the task of ontology learning [17] as well as classification, e.g. k-nearest-neighbors [6], the concept of similarity is of importance. A basic intuition of similarity is for example given by Lin [23]. Similarity is related to the commonalities and differences between two objects. More commonalities implies higher similarity. Vice versa, more differences implies lower similarity. Two identical objects should have the maximum similarity. In addition, only identical objects should be able to achieve maximum similarity. A similarity measure computes the similarity between two objects [35]:

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

A similarity of $1$ implies identical objects and a similarity of $0$ implies that there are no commonalities between the input objects. The information-theoretic definition of similarity states that a similarity measure should factor in both commonalities as well as differences to compute the similarity between objects [23].

Many types of information can be encoded as feature vectors in $\mathbb{R}^n$, e.g. words [22] [25], graphs [31] [5], etc. The similarity between feature vectors can be measured using the distance $\delta_S(\vec{p}, \vec{q})$ between two vectors $\vec{p}$ and $\vec{q}$. The distance represents the differences between two feature vectors. A typical distance measure is the $L_S$-Norm [35]:

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

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

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

Similarity between vectors can now be defined by mapping the distances to similarity values using a correspondence function [35]:

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

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

Aggarwal et al. [4] show that $L_S$ distance measures especially suffer the curse of dimensionality for increasing values of $S$. The curse of dimensionality implies that "the ratio of the distances of the nearest and farthest neighbors to a given target in high dimensional space is almost $1$ for a wide variety of data distributions and distance functions" [4]. Cosine similarity can offset this problem to some degree by including the direction of the vectors into the calculation of the similarity [18]. This is done by computing the dot product between two vector. It is defined as follows:

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

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

In ontology learning, semantic similarity is used to great effect, e.g for clustering objects to create hierarchies[17] [36] or mapping between different ontologies [11] [32]. 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 [32] 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. $\alpha$ is a function, which defines the importance of differences between $a$ and $b$. $A \cap B$ is the set of commonalities, and $A/B$ the set of differences between the $a$ and $b$. The defined similarity function is not symmetric, $sim(a, b) \neq sim(a, b)$.

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

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

Calculating the similarity between vectors is an easier task than calculating the similarity between objects in an ontology. Using neural word or graph embeddings, which are presented in Sections 2.6.2 and 2.6.3, enables the representation of classes and instances in ontologies as feature vectors. The mentioned curse of dimensionality is a non-issue as it applies, if the the number of irrelevant of features is high. This is typically solved by reducing the dimension of the feature vectors to include only relevant features [12]. Neural embeddings however seem to implicitly capture only relevant features [25].

## 2.4 Problem statement

The task of this thesis is the classification of orphan classes in Wikidata. In other words a function is needed, which given an orphan class $u$ of a taxonomy $T = (C, S)$ with a root class $root_T$, find an appropriate superclass for $T$. Doan et al. [11] suggests that for the task of placing a class into an appropriate position in $T$, either finding the most similar class, most specific superclass, or most general subclasses of $u$, are sensible approaches. This induces that the appropriate superclass for an orphan class $u$ is the superclass of the most similar class to $u$. Therefore we can define the problem, 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 $f : \mathbb{N} \mapsto \mathcal{P}(\mathbb{N})$, which, given an orphan class $u \in C$, returns a set of classes $P = f(u)$, fulfilling the following criteria: **TODO: redefine second criteria**

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

$$P = succ(\max_{c \in C}(sim(u, c))) \text{ superclasses of most similar class} \tag{8}$$

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

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

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

3. **Representation of items**. Items in Wikidata are structured information, similar to nodes in RDF graphs and Wikidata can be represented as such [31] [13]. 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 [37] [6]. 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.

k-nearest-neighbors 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 [6].

The basic notion of k-nearest-neighbors is presented in Figure 3. Given a set of points in $\mathbb{R}^2$ with classes blue, red and yellow. To classify an unknown class $u$, the closest $k$ classes are selected. In the example, the solid circle indicates $k = 4$ and the dashed circle indicates $k = 10$. The $k$ selected neighbors vote for their own class with a given weight, which is typically dependent on its similarity to $u$. In this simple example, it is assumed that all points have a uniform weight. Following, for $k = 4$ $u$ is classified as yellow and for $k = 10$ as red.

In comparison to the example, the weights assigned to each neighbor are not uniform. Weights are assigned, so that similar neighbors are given larger weights (affinity). Because in practical applications many neighbors may be very similar to each other, which can skew the classification results, Chen et al. [6] propose to additionally down-weight very similar neighbors (diversity). This can be accomplished by using kernel ridge interpolation (KRI) [6]. The following quadratic programming

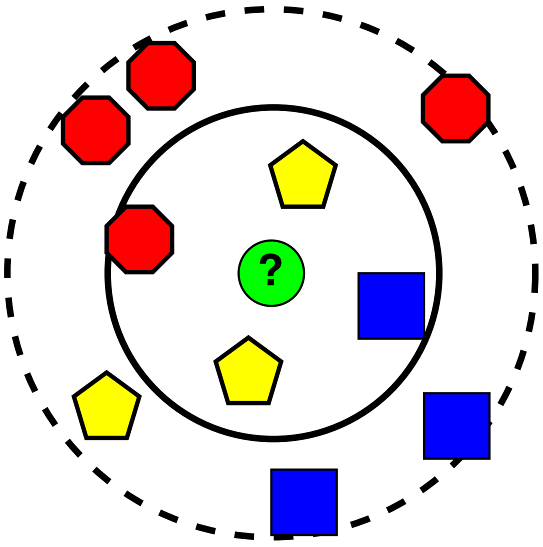

Figure 3: Example for k-nearest neighbors for 3 classes with k=4 and k=10.

has to be solved to calculate the weights:

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

where $w \in \mathbb{R}^{k \times 1}$ the weights of the nearest neighbors, $S \in [0,1]^{k \times k}$ is the similarity matrix between the nearest neighbors of $u$, $s \in [0,1]^{k \times 1}$ is the similarity between $u$ and its nearest neighbors, $\lambda > 0$ is a regularization parameter. It is common to assume that weights are positive and the sum of weights is $1$. Minimizing $-s^T w$ alone would return all weight on the nearest neighbor with $w_1 = 1$. The ridge regularization term $\frac{\lambda}{2} w^T w$, which controls the variance of the weights in $w$, counteracts this by pushing the weights closer to uniform weights. Increasing $\lambda$ increases the uniformity of the weights. Together these terms fulfill the affinity requirement. The term $\frac{1}{2} w^T S w$ represents the diversity requirement, as the weights of very similar neighbors are down-weighted to minimize the term. Solving this problem therefore returns weights, which fulfill the requirements of affinity and diversity. Experiments show that KRI kNN on average generates better results than uniformly weighted kNN [6].

## 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 dis-

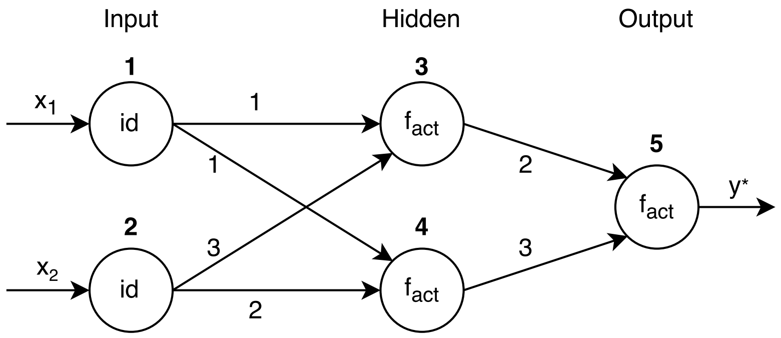

Figure 4: Example for feedforward neural network

cussed. Solutions for generating word and graph embeddings by Mikolov et al. [25], Ristoski and Paulheim [31], Cao et al. [5] 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 $(i,j) \in C$ [19]. Each neuron has an activation function $f_{act}$, which commonly is the Fermi function

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

mapping values to the interval of $(0,1)$. This function is non-linear and therefore allows the neural network to solve non-linear problems. Input neurons typically use the identity function $id$ as activation function [19]. The output of a neuron is computed by summing up the products of weights and inputs of the neuron, which is called net input, and applying the activation function on this value:

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

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

A feedforward neural network consists of an input layer with $s_i$ input neurons, an output layer with $s_o$ output neurons and $d$ hidden layers with $h$ neurons per layer. A neural network with many hidden layers is called deep neural network. The connections in a feedforward network only connect the neurons of a layer $i$ with the next layer $i+1$. This means that a neuron can never influence itself. Networks, which allow this, are called recurrent neural networks. See in Figure 4 a feedforward neural network with $s_i = 2, s_o = 1, d = 1, h = 2$ with already initialized weights.

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

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

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

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

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

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

Training objective is to minimize the error function $E$. This can be accomplished by adjusting the weights $w_{ij} \in W$. Rather than brute-forcing the optimal weights for a network, backpropagation using gradient descent is applied in training. The idea of gradient descent is to look at the slope of the error function in respect to the current weights and adjust the weights into the descending direction. To do so the partial gradient in respect to each weight $w_ij$ has to be derived from the error function [27]:

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

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

Subsequently, the weights can be updated by adding $\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$ to each corresponding weight [27]. 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 [25], the connection weights between the input and hidden layer. More specifically, if the weights are represented as a matrix (e.g. see Equation 12), each row is an embedding for the corresponding input neuron.

### 2.6.2 Word embeddings

Mikolov et al. [25] introduce two neural network language models, Continuous Bag-of-Words and Skip-gram, 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

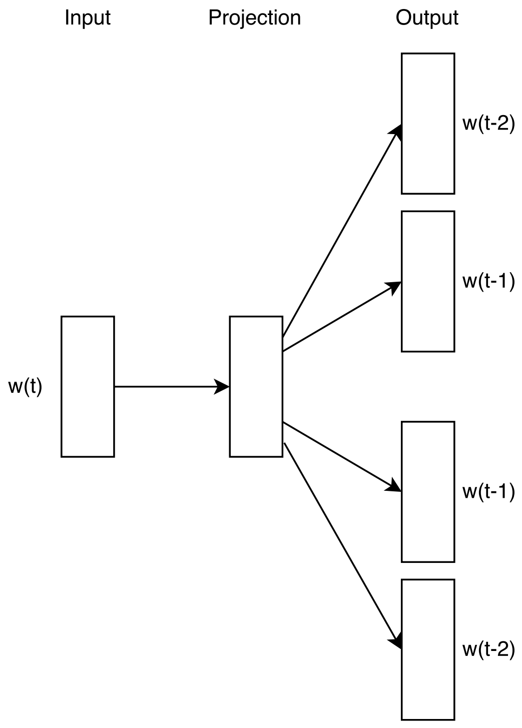

Figure 5: Skip-gram model

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: $vector($"biggest"$) - vector($"big"$) + vector($"small"$)$ [25].

Further research in this topic has shown that SGNS is generally more effective in generating high-quality word embeddings than CBOW [25] [26] [22], as it is better in representing rare words. Therefore only the skip-gram model with negative sampling will be introduced in this thesis. The following description is based on work by Mikolov et al. [25] [26], Levy and Goldberg [20] [21] [16], Rong [33], and Levy et al. [22].

The skip-gram model with negative sampling 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 $w_1, \dots, w_n$, e.g. a text corpus like a Wikipedia text dump, the context to a word $w_i$ are the $2 * L$ words around it: $w_{i-L}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+L}$. $L \in \mathbb{N}$ is the size of the context window. The network (see Figure 5) 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 [33].

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 [20]:

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

, which leads to the maximization objective [21]:

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

This problem has a trivial solution with $\vec{c} = \vec{w}$ and $\vec{c} \cdot \vec{w} = K$, for a large enough $K$ [21] [20]. 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 [21]:

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

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

Additionally Mikolov et al. [25] 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 down-sampled (occur less often). This is done before context generation and increases the effective size of context windows [20], which improves the quality of word embeddings [26].

Research has shown that the effectiveness of word embeddings is highly dependent on the choice of hyperparameters [22] [26]. Experiments by Levy et al. [22] 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 [26].

### 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 [5].

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 [31] proposes the use of graph walks to generate linear sequence samples. Given an undirected, 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 $v_i \rightarrow e_{ij} \rightarrow v_j \rightarrow \dots$, 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. [29] is used, where the number of walks per vertice is limited. A random walk $W_v$ rooted at vertice $v$ consists of random variables $W_v^1, W_v^2, \dots, W_v^k$. $W_v^{i+1}$ 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 [29]. Evaluation results show that this approach outperforms standard feature generation approaches [31].

Cao et al. [5] 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 $v_i$. A Similar to PageRank a teleportation probaility $\alpha$ 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 [5]:

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

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 $v_i$ [5]:

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

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.

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 [25] [5]. 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 [5].

## 2.7 Ontology learning

Manually building ontologies is an expensive, tedious and error-prone process [17]. Maedche and Staab [24] 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 ontology learning supports ontology engineers in the construction and maintenance of ontologies by providing OL techniques in the form of tools like *OntoEdit* [24]. 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. [7].

The process of ontology learning 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. [7], Maedche and Staab [24], and Hazman et al. [17]. 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 ontology learning can be divided into subtasks. Maedche and Staab [24] 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. [7] introduces a generic ontology learning 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 [34] 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 ontology learning is concerned with learning taxonomic relations. The following approaches, categorized by Cimiano et al. [7], use text as input.

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

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

, where $NP$ stands for noun phrase, and $NP_{hyper}$ is a hypernym or superclass, while $NP_{hypo}$ are hyponyms or subclasses. These patterns provide reasonable results, but the manual creation of patterns is involve high cost and time investments [36].

*Clustering* uses some measure of similarity to organize objects into groups. This can be achieved by representing the words or terms as vectors [9], 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 [28] 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. [14] develop a method to construct semantic hierarchies given a hyponym (subclass) and a list of corresponding superclasses. The solution uses neural word embeddings generated with the Skip-gram model [25], which was presented in Section 2.6.2. Using the linguistic regularities of these word embeddings, Fu et al. train a linear projection on existing sets of hypernym-hyponym pairs. A linear projection is used instead of an offset, because it was observed that the subclass-of relation is too complex to be represented as a simple offset. The embeddings are trained on a 30 million Chinese sentence corpus. And the projection is trained on $15247$ hypernym-hyponym word pairs. The handpicked testing data consists of $418$ entities and their hypernyms. The method achieves an F-score of $73.74\%$ on a manually labeled dataset, and outperforms existing state-of-the-art methods.

Petrucci et al. [30] develop a recurrent neural network for ontology learning. The purpose of the work is a first step to verify whether neural networks are able to support the ontology learning process. The task solved by the specific network is the translation of encyclopedic text to logical formulas. This task is divided into the task of sentence transduction, identify logical structure and generate corresponding formula template, and sentence tagging, identifying the roles of words in the input sentence. The tagged sentence can then be mapped unto the generated formula template. This is achieved by the use of a recurrent neural network for sentence tagging and a recurrent encoder-decoder for sentence transduction. Both are supported by gated recursive units, which provide a short-term-memory effect. The developed network is, at this time, under evaluation. Therefore, no statement about the effectiveness of recursive neural networks for the task of ontology learning can be made based on the paper.

**TODO: Interpret related work in context of this thesis.**

# 3 Analysis of the Wikidata taxonomy

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

The taxonomy contained in the Wikidata dump of 2016/11/07 was analyzed. Classes were recognized as defined in Section 2.1. Orphan classes were identified by checking whether a class does not have the *subclass of (P279)* property. The taxonomy contained a total of 1299501 classes at this time.

## 3.1 Root taxonomy

The state of the taxonomy was captured in regards to the root class *entity (Q35120)* (see Figure 6). 1260842 classes are currently subclasses of *entity (Q35120)*. 97% of all classes are therefore nodes in the root taxonomy. This implies a high agreement in the Wikidata community on which class is considered root, and thereby also supports the modeling decision made in Section 2.2, which assumes that a taxonomy only has one root, and this root is *entity (Q35120)* in Wikidata. **Last sentences of this paragraph are mostly interesting trivia, but have currently no further use.** The longest shortest distance between the root and a leaf class is 20. The classes *Q639064*, *Q15978631* and *Q151055* fulfill this characteristics. Each of them is subclass of *Homo (Q171283)*.

## 3.2 Orphan classes

The characteristics of all orphan classes and the root class were analyzed. This set contains a total of 16373 classes. 13807 classes have an English label and 11534 classes have a corresponding English Wikipedia page.

Regarding the number of unique properties (or statement groups) per class (see Figure 7) the median is 3 and the average is $\approx 4.8$. Combined with the analyzed unique property frequency (see Figure 8), it can be seen that many orphan classes are linked to other knowledge bases and taxonomies , e.g. Freebase or GND, using an identifier. Another observation is that 901 of the orphan classes have properties related to taxons, which is a, to some degree, separate taxonomy in Wikidata, created with properties like *taxon name (P225)*, *parent taxon (P171)*, etc.

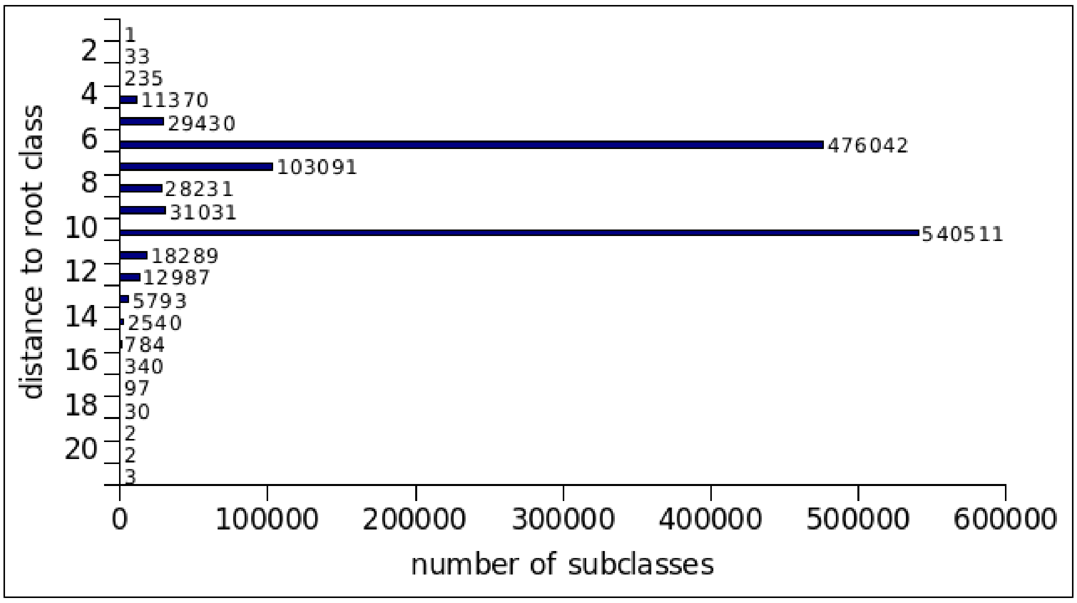

Figure 6: Distance of subclasses to root class *entity (Q35120)*. Wikidata (2016/11/07)

Most orphan classes have only $0$ to $1$ instances, the median is $1$ and the average is $\approx 4.65$ instances per class (see Figure 9). There are however outliers with big numbers of instances, which skew the average number of instances per class. This may for example imply, that classes are created mainly for the purpose of grouping newly created instances.

The median and average for subclasses per class are $0$ and $\approx 0.85$ respectively (see Figure 10). This implies that the graph components of the orphan classes are very small, and in most cases contain only the respective orphan class.

The question has to be asked, whether an algorithm should try to handle the complete set of orphan classes, or only a specific subset, which fulfills certain requirements. The answer depends on what information such an algorithm requires as input. The full set of orphan classes is problematic for use in an algorithm as there are few shared characteristics over all classes. $\approx 16\%$ of classes are not labeled and $\approx 31.4\%$ of classes have no instances. Both labels and instances are useful characteristics for a class to have. The label allows the recognition of the class in natural text and ensures that the class fulfills a basic quality criteria, since the label is the first characteristic of a Wikidata item, which should be created by a user [1]. Instances represent classes and thereby describe them. For example, Rodríguez and Egenhofer [32] use instances to compute semantic similarity between classes. Therefore, I propose to only consider the set of classes, which have a label in the English language and at least $1$ instance, as input for the algorithm.

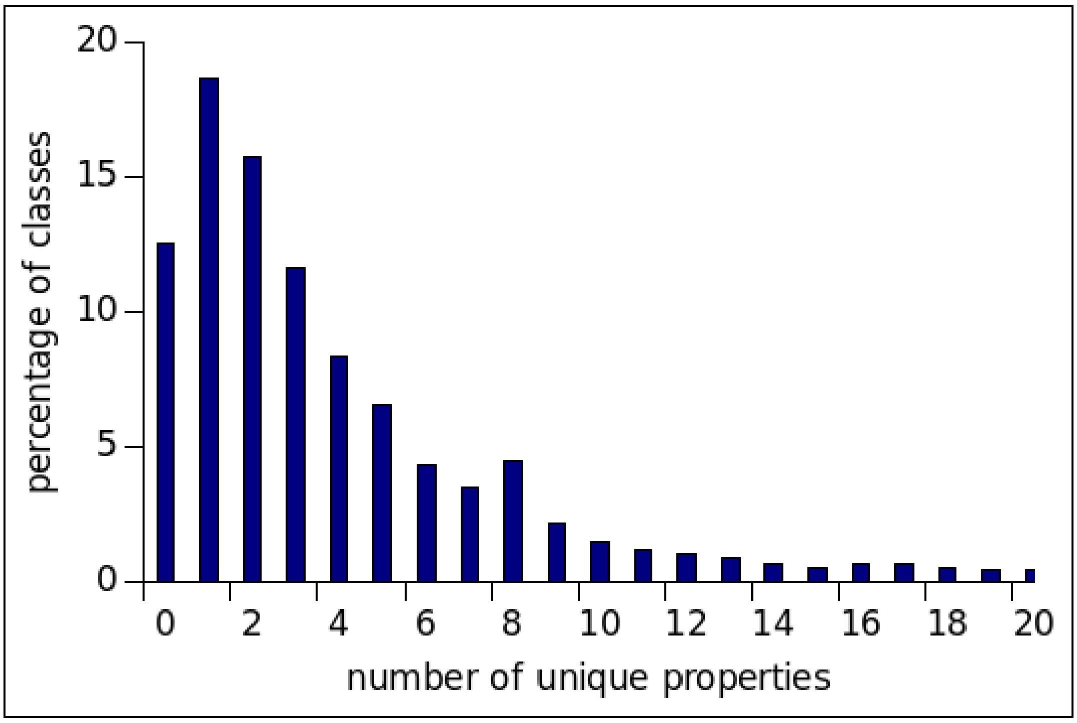

Figure 7: Percentage of orphan classes with a specific amount of unique properties. Wikidata (2016/11/07)

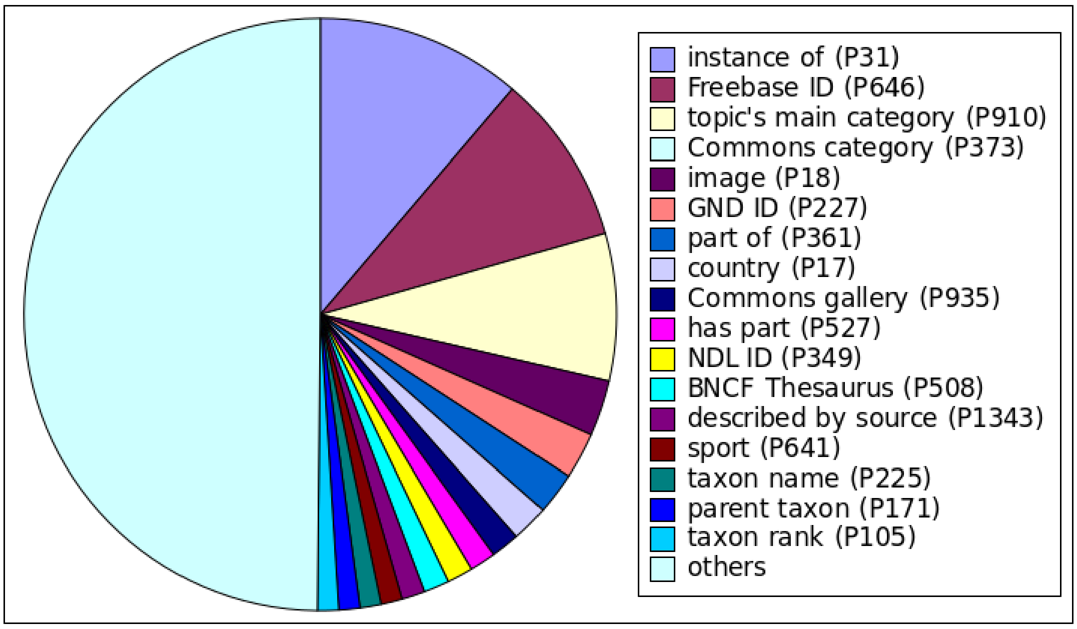

Figure 8: Frequency of properties in orphan classes. Wikidata (2016/11/07)

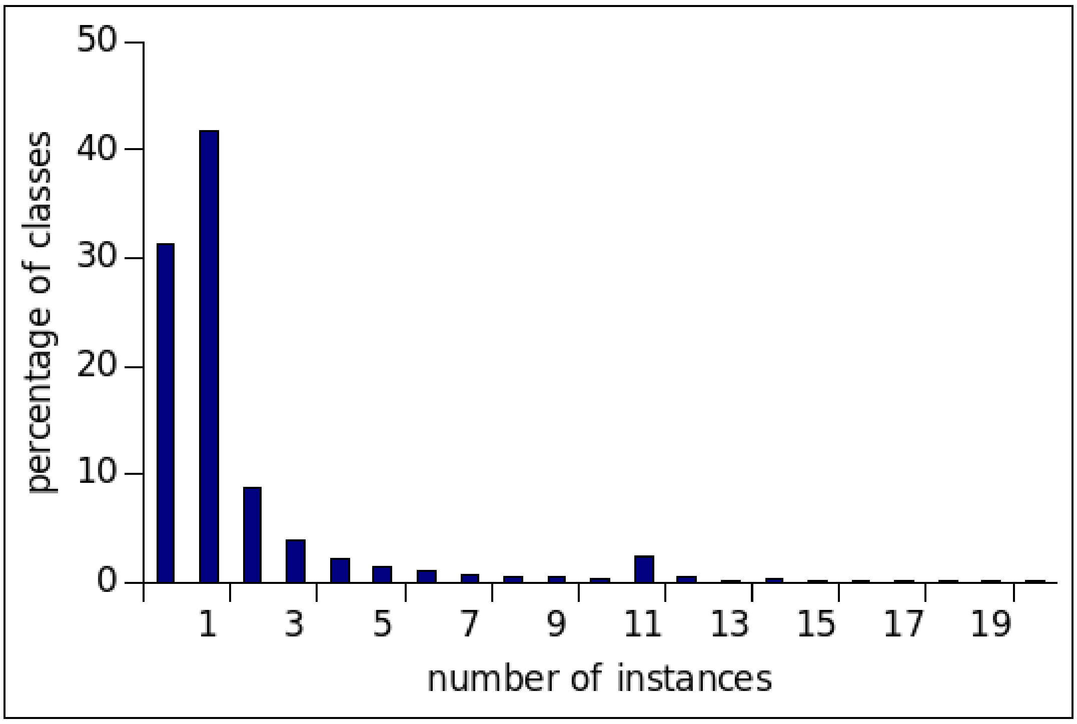

Figure 9: Percentage of orphan classes with a specific amount of instances. Wikidata (2016/11/07)

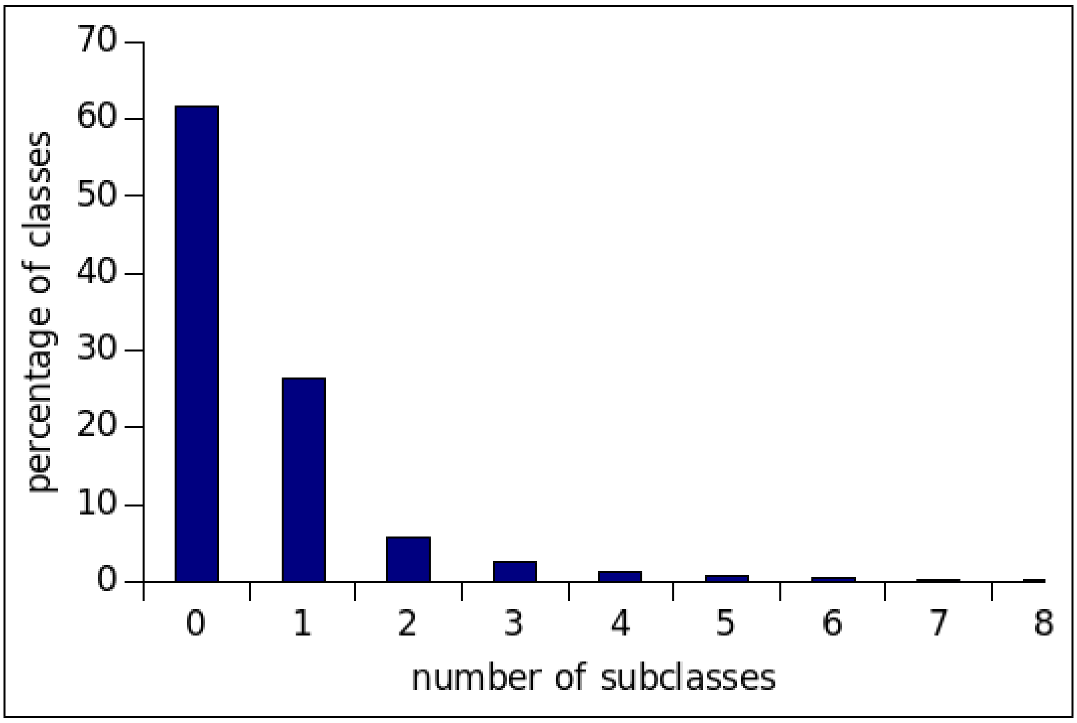

Figure 10: Percentage of orphan classes with a specific amount of subclasses. Wikidata (2016/11/07)

### 3.3 Labeled, instantiated orphan classes

Following this argument, the subset of labeled and instantiated orphan classes was analyzed. This set contains a total of $9157$ classes. $7557$ classes have an English Wikipedia page, which is $\approx 82.5\%$ of all analyzed classes in the subset, and an increase of $\approx 12\%$ in comparison to the full set of orphan classes.

The average of unique properties per class increases from $4.8$ to $5.5$, while the median remains at 3 (see Figure 11). The frequency of different properties appearing in classes however is more distributed (see Figure 12). This implies that the classes of this subset are related to different topics, and share only few commonalities.

As expected the average of instances per class increased to $\approx 7.37$ from $\approx 4.65$. However the median is still $1$ (see Figure 13). $\approx 59\%$ of classes have only $1$ instance.

At the same time the average of subclasses per class halved to $\approx 0.4$ (see Figure 14). Combined with the unchanged median of $1$ instance per class, supports the assumption, that classes are mainly created for the purpose of grouping instances, while adding the newly created class to the main taxonomy is being neglected.

The classes of the subset have, on average, a better level of description, more unique properties and instances, than the full set of unlinked classes. Between these classes the number of commonalities is lower, because the same properties occur not as frequently for different classes. The lack of commonalties between unlinked classes is not very problematic, as the algorithm would not compare unlinked classes with each other but with classes in the root taxonomy.

### 3.4 Discussion (TODO: new subsection title)

After completing the analysis, it can be seen that *entity (Q35120)* is the root class of the taxonomy, and Wikidata's taxonomy is in a good state, since $97\%$ of classes are part of the root taxonomy. **TODO: not sure if i can just say that it is in a good state** Unlinked classes share few commonalities, and as shown above, a percentage of classes lack even basic information like a label. For developing an algorithm it is necessary to set a baseline, of what can be expected from the input. In this case, input is required to be labeled and have at least one instance. The benefit of this specific restriction is, that a basic level of descriptiveness can be assumed for every class. A label additionally allows supplementing the information, provided by Wikidata, to be supplemented with natural text sources, like Wikipedia, since it is now possible to recognize occurrences of the class in text.

# 4 Hybrid algorithm

## 4.1 Components

The hybrid algorithm will use a Word2Vec model to generate word embeddings, which then can be used in a classification task. This leads to the identification of

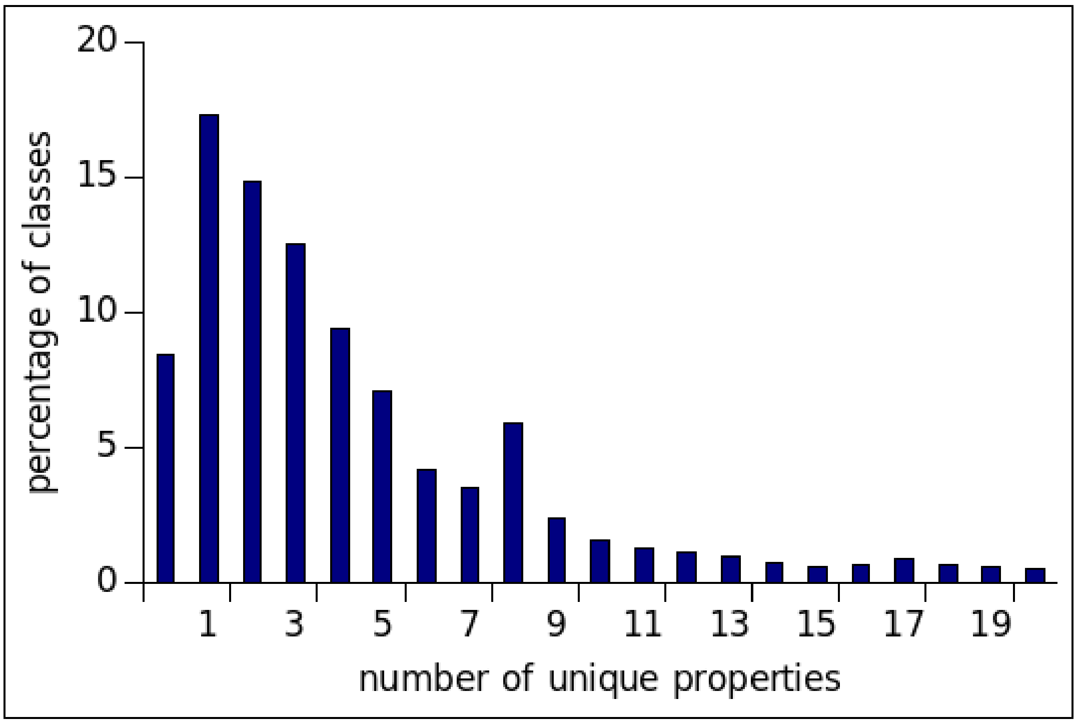

Figure 11: Percentage of labeled and instantiated orphan classes with a specific amount of unique properties. Wikidata (2016/11/07)

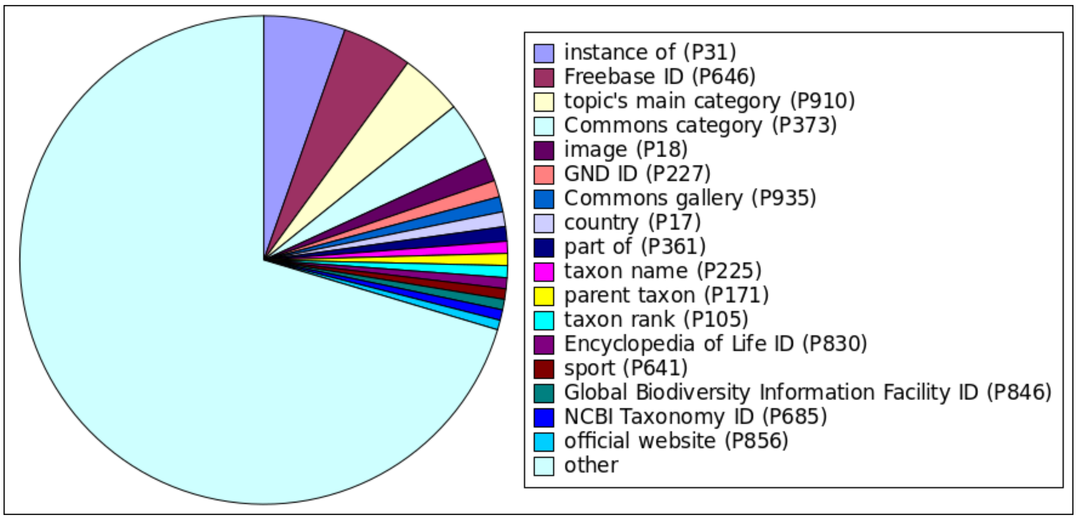

Figure 12: Frequency of properties in labeled and instantiated orphan classes. Wikidata (2016/11/07))

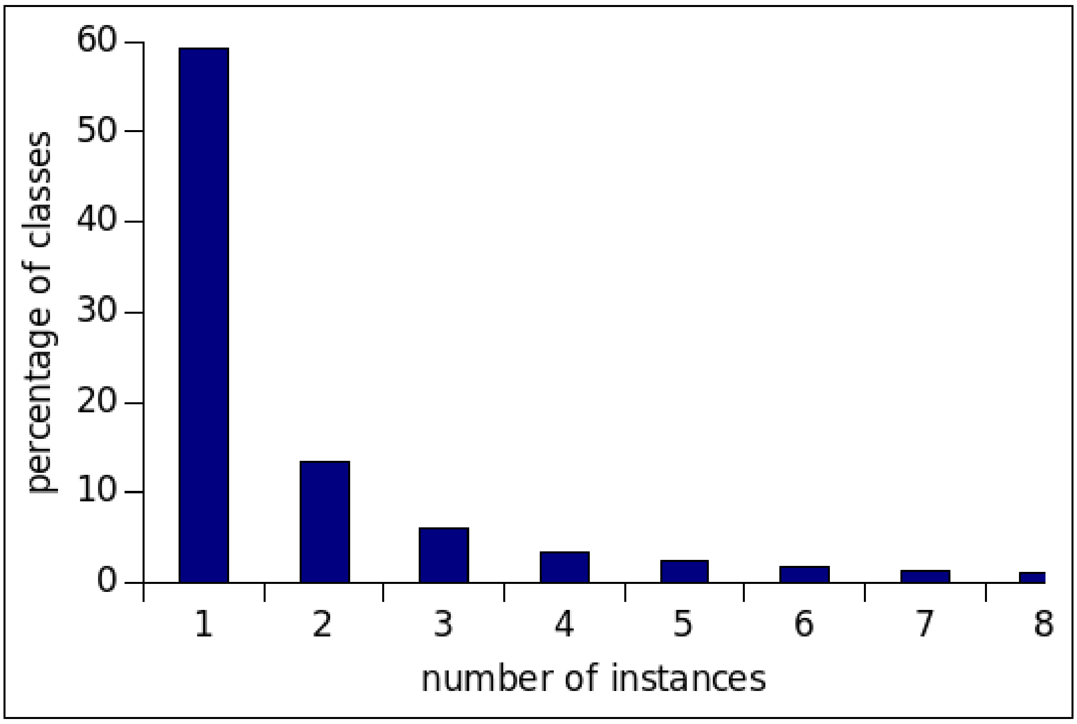

Figure 13: Percentage of labeled and instantiated orphan classes with a specific amount of instances. Wikidata (2016/11/07))

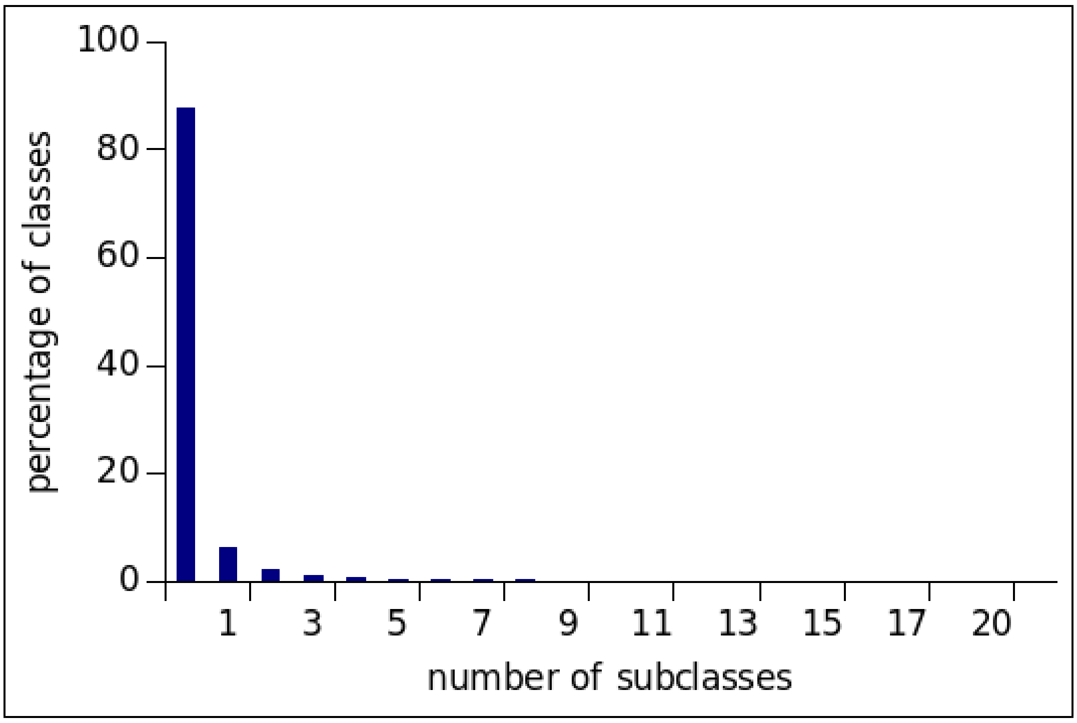

Figure 14: Percentage of labeled and instantiated orphan classes with a specific amount of subclasses. Wikidata (2016/11/07)

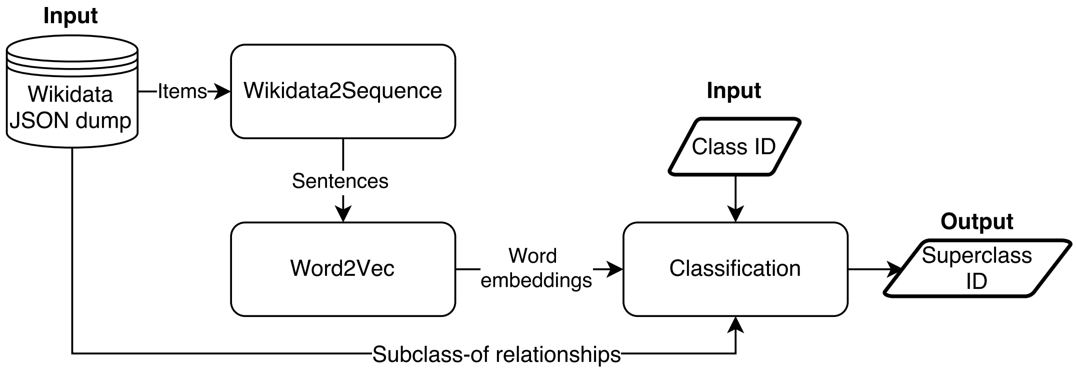

Figure 15: Data flow between components

three main components. These main components will be implemented by a baseline algorithm and for each variation an improvement is proposed, which in general replaces one of the components. The data flow and input/output between the components is visualized in Figure 15

1. **Wikidata2Sequence**. For the task of generating word embeddings the Skip-gram model with negative sampling will be used. As mentioned in Section 2.6.2, both Word2Vec models require a linear sequence of words as input. Wikidata's highly interlinked structure is more alike to an RDF graph [13]. Therefore the first required component for the algorithm is a mapping from Wikidata to word sequences. The output of this method is required to be at least one sequence of words, which is called sentence. Therefore multiple sentences represent a text. Each word in the sequence has to be either a Wikidata item or property ID, or the data type for a literal value (e.g. replace all strings with *string*).

2. **Word2Vec**. The Skip-gram model with negative sampling (SGNS) will be used in every variation of the algorithm. It has been proven through theoretic analysis as well as experiments that SGNS creates very effective word embeddings, if its hyperparameters are chosen well. **TODO: citations**

3. **Classification**. The final component of the algorithm is a classification method. The word embeddings produced by SGNS group similar words close to each other and preserve linguistic regularities. 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 [14]. For the purpose of evaluation each classification component should have the same output, this will allow a comparison between the different variations of the algorithm. Therefore each classification method will output the Wikidata ID (e.g. Q35120) of the most likely superclass.

Wikidata2Sequence and Word2Vec are executed once to compute the word embeddings of the corresponding Wikidata dump. The classification method will then be trained on the word embeddings and can be applied repeatedly to unlinked classes.

## 4.2 Baseline

The baseline algorithm uses a very basic Wikidata2Sequence component and simple weighted kNN for classification. If not otherwise mentioned, the variations will use the same hyperparameters and components as the baseline algorithm.

1. **Wikidata2Sequence**. Trivial sentences will be generated from statements. Each statement can be transformed to a triple $(itemid, pid, value)$, where value is either another item ID or a data type. **Insert example.**

2. **Word2Vec**. As mentioned above a Skip-gram model with negative sampling will be used. Based on Levy et al. [22], Mikolov et al. [25] and Mikolov et al. [26] the following hyperparameters have been proven to be effective in the task of creating word embeddings and are therefore used:

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

   The original paper by Mikolov et al. [25] uses context windows of size $2$. Bigger context windows show slight improvements in the quality of word embeddings Levy et al. [22]. However triple sentences 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 graph walk sentences, if not mentioned otherwise.

3. **Classification**. KRI-kNN by Chen et al. [6] is implemented (see Section 2.5). As similarity measure the cosine similarity will be used, which finds precedence in other work about word embeddings like Mikolov et al. [25].

## 4.3 Variation: Graph walk

A graph walk, developed by Ristoski and Paulheim [31] and explained in Section **??** can be used for generating longer sentences instead of triple sentences. The longer sequences are able to better capture contextual information than three word sentences. This is achieved by creating paths of a depth $d$, starting from each class. Evaluation of RDF2Vec shows that Skip-gram and CBOW both produce better word embeddings this way [31]. Therefore it is expected that graph walk will also improve the performance of this algorithm. However, the potentially exponential number of walks is an issue, as this will increase the runtime for generating the walks to extremely high values. A single-process implementation using breadth-first search requires $\sim 40$ seconds to generate all walks for one vertice. **mention hardware stats, and implementation details in appendix?** Generating walks for all $\sim 24$ million items or $\sim 1.2$ classes is therefore not feasible, e.g. using the mentioned implementation generating walks for all classes would take $\sim 555$ days. Ristoski and Paulheim [31] proposes a first measure to improve the runtime by reducing the maximum number of walks per vertice. Another measure proposed by the author is the sampling of vertices, for which walks should be generated. The sampling should try to maximize the coverage of edges and vertices by the walks. One requirement for the sampling would be the coverage of all classes, as the word embeddings for these classes are used in the classification component of the algorithm. The implementation will combine limiting the number of walks per vertice and a partially random sampling approach. The sampling will first choose only vertices from the set of classes, which are not yet covered by the generated walks. After all classes are covered, the same choice will be made on the set of all items. The following parameters are chosen for the component based on Ristoski and Paulheim [31]:

- Depth of walks: $4$
- Maximum number of walks per vertice: $100$

Because of time constraints the number of walks, which can be generated is restricted. **When should the graph walk terminate? specific node or edge coverage, number of walks, number of nodes, time?**

**TODO: choice of hyperparameters**

## 4.4 Variation: Vector offset

## 4.5 Variation: Linear projection

## 4.6 Variation: Wikipedia

## 4.7 Implementation details

**Is this section necessary?**

# 5 Evaluation

## 5.1 Method

Dellschaft and Staab [10]

### 5.2 Generation of gold standard

### 5.3 Results

## 6 Conclusion and future work

## References

[1] Wikidata: Create a new item. https://www.wikidata.org/wiki/Special:NewItem, . Accessed: 2017-02-05.

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

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

[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] 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`.

[6] 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`.

[7] 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`.

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

[9] 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.

[10] 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`.

[11] 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`.

[12] 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`.

[13] 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.

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

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

[16] 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`.

[17] 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.

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

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

[20] 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`.

[21] 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`.

[22] 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`.

[23] 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`.

[24] 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`.

[25] 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`.

[26] 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`.

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

[28] 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.

[29] 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`.

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

[31] 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`.

[32] 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`.

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

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

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

[36] 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`.

[37] 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.

**TODO: the current definition of component is useless.**

**Definition 18** (Component)**.** Let $G$ and $H$ be directed graphs. $H$ is a component of $G$, if $H$ is a connected subgraph of $G$ and $H$ is not subgraph of another connected subgraph of $G$, which has more vertices or edges than $H$.

**Definition 19** (Cycle)**.** A walk $W = (v_1, \ldots, v_n)$ of length $n$ is called a cycle, if $v_1 = v_n$.

**Definition 20** (Directed acyclic graph)**.** A directed graph $G$ is called directed acyclic graph, if there are no cycles in $G$.

**TODO: definitions**

**Definition 21** (Weighted graph)**.**

**Definition 22** (Graph walk)**.**