Generating Categories for Sets of Entities

Shuo Zhang*
Bloomberg
London, United Kingdom
szhang611@bloomberg.net

Krisztian Balog
University of Stavanger
Stavanger, Norway
krisztian.balog@uis.no

Jamie Callan
Carnegie Mellon University
Pittsburgh, USA
callan@cs.cmu.edu

ABSTRACT

Category systems are central components of knowledge bases, as they provide a hierarchical grouping of semantically related concepts and entities. They are a unique and valuable resource that is utilized in a broad range of information access tasks. To aid knowledge editors in the manual process of expanding a category system, this paper presents a method of generating categories for sets of entities. First, we employ neural abstractive summarization models to generate candidate categories. Next, the location within the hierarchy is identified for each candidate. Finally, structure-, content-, and hierarchy-based features are used to rank candidates to identify by the most promising ones (measured in terms of specificity, hierarchy, and importance). We develop a test collection based on Wikipedia categories and demonstrate the effectiveness of the proposed approach.

CCS CONCEPTS

• **Information systems** → *Semi-structured data*; *Incomplete data.*

KEYWORDS

Category generation; Wikipedia categories; entity typing; knowledge base population

ACM Reference Format:
Shuo Zhang, Krisztian Balog, and Jamie Callan. 2020. Generating Categories for Sets of Entities. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3412019

1 INTRODUCTION

Category systems provide a hierarchical and topical organization of entities and concepts. They are meant to help arrange and access topically related items, and are immensely useful. Compared to type hierarchies of knowledge bases, such as the DBpedia ontology or Freebase types, we are focusing on category systems that are much finer-grained and larger scale (e.g., Wikipedia categories or YAGO classes). There, many categories have complex names that reflect human classification and organization, and as such, encode knowledge about class attributes, taxonomic, and other semantic relations [23, 26]. As a result, categories represent a unique and valuable resource, which has been exploited for various tasks, including entity retrieval [9, 15, 39], query understanding [2], and knowledge acquisition [23, 26].

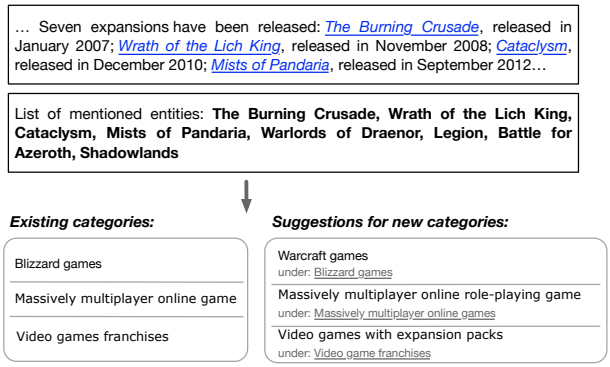

Figure 1: Given a set of entities along with context (surrounding text), we aim to create new categories, if no fine-grained enough category exists in the category system. To ensure that a new category can meaningfully complement the current category system, we also find its place (i.e., parent category) in the hierarchy. These suggestions are presented to a human editor for consideration.

A large body of prior work focuses on assigning entities to categories, which is usually referred to as the task of fine-grained entity typing [8, 25, 43]. Entity typing deals with a single entity as input (often with some surrounding text, as context), and it implicitly assumes that there exist suitable types/categories. However, since new entities continuously emerge [11, 13, 42], this assumption is unrealistic in practice, and brings about the need for creating new categories.[1] The problem of expanding hierarchical category systems of entities with new categories has not received due attention to date.

An important consideration in this work is that category creation should not be driven by individual entities. Looking at single entities may not be informative, as each could be an outlier. On the other hand, when there is a set of entities for which no suitable, i.e., fine-grained enough, category exists, that would be a reason for expanding the category system. Note that it is very natural to operate with sets of entities; lists and tables are ubiquitous on the Web. Tapping into these would allow us to complement existing category systems. Taking Fig. 1 as an illustration, we can recommend existing categories for this set of mentioned entities, but they are not fine-grained enough. In this case, the creation of new categories would be desired.

*Work done while visiting CMU, USA.

[1] We note that it is a less severe issue for types, which are coarse-grained.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412019

The challenge we tackle in this paper is the following: given set of entities, generate new categories to meaningfully extend a given category system. The main difference between "plain" entity sets and categories is that the latter are "named" sets which are organized hierarchically. This gives rise to two specific novel subproblems we are addressing in this work: (i) generating a label for the new category (comprising of the input set of entities), and (ii) finding its place in the category system (i.e., locating an appropriate parent category).

To study this problem, we take Wikipedia categories as a representative of a large-scale hierarchical category system that is widely utilized. Currently, Wikipedia categories are created manually by editors ("Wikipedians"). With the increasing number of Wikipedia pages, more categories are created and added for organization. In 2012, there were about 660k categories, while at the time of writing there are over 1.1M categories. Nevertheless, the category system is still extremely sparse and noisy, as it contains duplications, errors, and oversights [6]. Given the sheer number of categories, the maintenance and expansion of the category system are becoming increasingly difficult. It would be of great practical value to provide a method for automatically generating categories, triggered by updates made to a Wikipedia article. The specific instantiation of our general problem in this application scenario is the following. We assume a user is editing a Wikipedia article, where a list of entities has already been mentioned. Against this setting, our objective is to see if new categories could be generated, based on the list of entities, which may be added to Wikipedia. These suggestions are presented to a human editor for consideration. More generally, we aim to generate new categories in an automatic manner for sets of entities. While in this work we focus exclusively on Wikipedia categories, the methods we develop generalize to other category systems as well.

To control the quality of category labeling, we follow two general rules (in accordance with Wikipedia guidelines): (1) a label should be as specific as possible; and (2) similar categories should be avoided [20]. Inspired by this, we identify four main challenges related to the automatic generation of categories.

- **(C1) Specificity:** A proper category should be sufficiently specific to capture the entity set intent. For example, *2018 Oscar Winners* is a more specific category than *Actors*, although both might describe the contents of an entity set.
- **(C2) Hierarchy:** The more specific a category, the deeper down it is located in a hierarchy. We explicitly focus on leaf categories and aim to place them in the hierarchy by finding their respective parent categories.
- **(C3) Redundancy:** Redundant categories should be avoided. While we are not addressing this problem explicitly in this paper, we postulate that by identifying parent or sibling categories, and presenting these to the human editor, would help address this issue.
- **(C4) Importance:** An important category is expected to organize salient entities and encode knowledge, which is not already covered by sibling categories.

We propose a pipeline architecture consisting of several steps to overcome the above challenges. Specifically, given a set of entities as well as the surrounding text, we aim to generate a ranked list of categories that capture the semantics of that set. Categories can be generated by summarizing the words contained in the input. The ranked list may be further refined by incorporating signals from the context (e.g., surrounding text and page title). A particularly important subtask is to find the generated category's place in the hierarchy, by identifying its parent category. This information can then be further utilized as an additional ranking signal. In summary, the contributions of this work are as follows.

- We propose the task of generating categories for sets of entities, and develop an approach to generate categories that are specific, hierarchical, nonredundant, and important.
- We develop a test collection based on Wikipedia categories and lists, and perform an extensive evaluation of the proposed approach.

2 RELATED WORK

The task of generating new categories from entity sets for extending the category system of a knowledge base is related to the problems of entity typing, conceptual labeling, knowledge base population using semi-structured data, and ontology generation.

Entity typing refers to the task of assigning types to mentions of entities in context [8, 25, 29, 43]. The roots of this problem may be traced back to named entity recognition (NER), which focuses on the detection of entity mentions in text and type-annotation of these mentions from a small set of coarse types (such as person, organization, location, and miscellaneous) [22]. Over the years, attention in NER has shifted from annotating against a small set of coarse types to using fine-grained types from a type taxonomy, which allows multiple and hierarchical types [37]. Unlike traditional NER work, recent entity typing approaches assume entity mentions to be provided as part of the input. For example, AFET [29] is a hierarchical partial-label embedding method for automatic fine-grained entity typing. It learns embeddings for mentions and types-paths, and iteratively refines the model. Choi et al. [8] introduce a new entity typing task to predict a set of free-form phrases, namely ultra-fine types, that describe the role the entity plays in a given sentence. Having free-form noun phrases as type descriptors, as opposed to existing fined-grained types from a taxonomy, is intended to improve downstream entity-focused tasks. Though both entity typing and our task deal with types and entities, we focus on creating new categories for entity sets, as opposed to dealing with a single entity.

Another related problem is *conceptual labeling*, which is the task of generating a small set of labels that best describe a set of words or phrases [36]. Labels can be concepts, entities, or types/categories from a knowledge base. While conceptual labeling takes a set of items as input, its objective is language understanding with the help of existing concepts/types. This is fundamentally different from our objective of expanding category systems of knowledge bases.

Knowledge base population (KBP) typically involves two tasks, entity linking and slot filling [14, 41]. The former aims to link (often ambiguous) entity mentions in text to specific entries in a knowledge base, while the latter is concerned with completing the information available on a given entity. Sets of entities, as defined by tables and lists, have been considered in KBP. For example, table-to-KB matching is considered as a fundamental step towards utilizing tables for KBP, and involves two specific sub-problems:

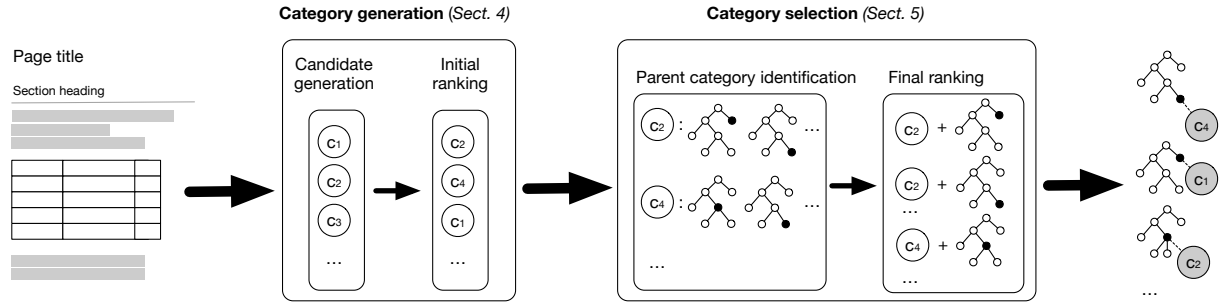

Figure 2: Illustration of our category generation pipeline.

entity linking for tables and table schema to predicate matching. Ritze et al. [31] propose an iterative method for matching tables to DBpedia. They develop a manually annotated dataset for matching between a Web table corpus extracted from Common Crawl [19] and DBpedia. In follow-up work, Ritze and Bizer [30] focus on a feature study for the same task. Specifically, they consider features extracted based on the tables themselves or from the knowledge base, and evaluate the utility of these for the matching task.

Category creation is also related to the task of *ontology generation* [1, 3, 5, 28, 32, 35, 38]. It can be based on various sources, from unstructured free text to structured tabular data. An example for the latter is the work by Pivk [28], who proposes automatic ontology generation using tabular structures. The steps include extracting and transforming a table into a regular matrix form, detecting table structure like orientation, schema and cells, identifying table type, rearranging table regions into a directed acyclic graph, and generating an ontology based on the graph. McGuinness [21] describes the scope of ontology specifications from simple ontologies to structured ones. The Wikipedia category structure corresponds the former because of its information hierarchy, but it is not a strict and logically grounded ontology due to the inconsistencies and loose relationships [38]. There are two main distinguishing aspects of our approach. First, we expand an existing taxonomy, unlike most prior approaches that generate categories from top to bottom. Second, we generate suggestions to assist a human editor instead of aiming for a fully automated category population.

The machine-readable semantic knowledge provided by taxonomies has proved to be beneficial in an array of natural language understanding problems [4]. Wikipedia categories represent a unique and extremely valuable resource in this capacity, and have been utilized for a broad range of tasks [2, 9, 15, 20, 23, 24, 26].

3 PROBLEM STATEMENT AND OVERVIEW

We provide a formal definition of the problem we are addressing.

Definition 1 (Category generation based on entity sets): *Given a set of entities E and its context E_C, category generation is the task of generating a ranked list of category suggestions $\langle c_1, \ldots, c_n \rangle$. Each category suggestion c is to be added as a leaf node in the category hierarchy under an existing parent category c_p.*

We assume that the context of E_C includes the title of the page and of the context where the entities lie, as well as any existing categories that have already been assigned to that article (by a Wikipedian).

We address the category generation task using a pipeline approach, shown in Figure 2. The two main components of this pipeline are category generation (Sect. 4) and category selection (Sect. 5). In *category generation*, we first generate candidate categories that are relevant to the input entities and their context. E.g., given a set of entities about "European national teams not affiliated to FIFA," it generates candidates like *National football teams in the isle of man*, *Football teams in the isle of man*, *Isle of man*, *National association football teams* and *European national and official selection-teams not affiliated to FIFA*. Then, we perform an initial ranking of the candidates, using a set of inexpensive features based on structure and content information.

In the second pipeline component, *category selection*, we first attempt to find the place of each candidate category in the category hierarchy by locating possible parent categories. For instance, for a candidate category *European national and official selection-teams not affiliated to FIFA*, possible parent include *National and official selection-teams not affiliated to FIFA*. Then, we perform a final ranking of the candidates, using their predicted place in the hierarchy as an additional ranking signal. This allows us to more accurately estimate category importance as well as to avoid creating redundant categories. In the above example, the candidate *National association football teams* would be excluded. Finally, the top-ranked suggestions would then be presented to a human editor, who can decide which of these categories, if any, should be created.

4 CATEGORY GENERATION

The first component of our pipeline is responsible for the generation of candidate categories. In particular, in this part, we address the challenge of *specificity* (**C1**). Our approach consists of a candidate generation step based on pointer-generator networks (Sect. 4.1), followed by an initial ranking of the candidates, using a set of inexpensive features (Sect. 4.2).

4.1 Candidate Generation

Abstractive summarization models, which are used for summarizing documents into a few sentences, have proved effective to generate titles for the semi-structured data [12]. In this work, we consider three neural text summarization models to learn to generate categories. We concatenate all the entities and contextual data into key-value pairs and use them as the representation of input data to be fed into the summarization models.

- *Pointer-generator network* [33] is a hybrid neural network that combines pointing and generating mechanisms.[2] It uses a bi-LSTM encoder and LSTM decoder with attention. The generation probability $P_g \in [0, 1]$ for each step is calculated from the context vector c_t, the decode state h_t, and the decode input x_t:
$$p_g = \sigma(W_c \cdot c_t + W_h \cdot h_t + W_x \cdot x_t + b) \,, \tag{1}$$
where W_c, W_h and scalar b are learnable parameters, and σ is the sigmoid function. P_g is used as soft switch to choose to sample from the vocabulary distribution P_v or to copy a word from the input with the attention distribution P_a. The loss for each token w is:
$$P = P_g \cdot P_v(w) + (1 - p_g) \cdot Pa(w) \,. \tag{2}$$
The loss function is the average negative log likelihood of the generated sequence. In addition, See et al. [33] propose a coverage mechanism to overcome token duplications; here, we take the pointer-generator itself as the base model.
- *NATS* [34] is an abstractive text summarization approach that extends the pointer-generator network. One issue summarization models suffer from are repetitions (both word-level and sentence-level). To overcome it, Intra-decoder [27] allows a decoder to keep track of previously decoded tokens apart from the source data, not repeatedly producing the same information. Additionally, sharing the weights with the decoder is a common solution that can boost performance. In summary, we take the pointer-generator network as the base model and equip it with the coverage mechanism and unknown words replacement, as in [34].
- *FAST* [7] is a neural generative model that first selects salient sentences and then rewrites them abstractively. Making use of salient information from the extraction process is another way to improve the summarization [34]. The FAST model consists of an extractor agent and an abstractor network. The extractor aims to learn to extract the salient data from the source by training a pointer network, while the abstractor rewrites the extracted sentences to get the final summarization.

4.2 Initial Ranking

We perform an initial ranking of candidate categories using structure-based and content-based features. These features are fed into a Random Forest regressor in order to obtain an initial ranking of the candidates. Only the top-k ranked candidates are kept for downstream processing.

4.2.1 Structure-based Features. The first set of features is based on structure (or patterns) in the category names; these are listed in the top block of Table 1. Most features are simple characteristics. The first one is the length of the category name ($|c|$). The next feature (*IsPrepos*(c)) is motivated by the observation that prepositions are a strong indicator of semantic relations [17]. E.g., *Films directed by Joss Whedon* indicates the explicit relation between *Films* such as *The Avengers* and *Joss Whedon*. We thus consider the presence of a preposition as a feature. Further, we decompose category names based on entities mentioned in them and write E_c to denote the set of entities in category c. We find that over 91% of Wikipedia categories contain entities. Then, two features are constructed based on E_c, namely, the number of entities $|E_c|$, and the aggregation on the number of categories of the elements in E_c according to:
$$CatNum_{aggr}(E_c) = aggr\big(\{|C_e| : e \in E_c\}\big) \,, \tag{3}$$
where C_e indicates the set of categories entity e is a member of and *aggr* is an aggregator function. Specifically, we use *max*, *sum*, and *avg* as aggregators. Similarly, we also take the aggregated category term frequencies as signals.

4.2.2 Content-based Features. The second set of features, listed in the middle block of Table 1, are based on the content of categories, measured in terms of labels, entities contained, and parent categories. One group of features is based on the similarity of category labels. Intuitively, a new category should be consistent with the naming of existing categories. Using the candidate category name as a keyword query, we rank all existing categories C in Wikipedia using BM25, and take the retrieval scores of the top-k highest ranked categories as features (*CatNameSim*(c, C)). Similarly, we also rank all entities $\mathcal{E}$ in Wikipedia using the candidate category name as the query and use the top-k scores as features (*EntityNameSim*($c, \mathcal{E}$)).

We can also measure the similarity between the candidate category c and existing categories $c' \in C$ in terms of the entities they contain. For this, we operate on an index of categories where tokens are member entities. To estimate the member entities of the candidate category $\mathcal{E}_c$, we take all entities that are present in the input set. Then, we create a search query by enumerating all entity tokens in $\mathcal{E}_c$ to rank existing categories $c' \in C$. The top-k retrieval scores are used as features (*EntitySim*($\mathcal{E}_c, \mathcal{E}_{c'}$)). Additionally, we compute the entity overlap between $\mathcal{E}_c$ and each of the top-k existing categories $\mathcal{E}_{c'}$ in two different ways ($\frac{|\mathcal{E}_c \cap \mathcal{E}_{c'}|}{|\mathcal{E}_c|}$ and $\frac{|\mathcal{E}_c \cap \mathcal{E}_{c'}|}{|E_{c'}|}$).

Finally, we use parent categories (C_c and $C_{c'}$), analogously to member entities, by creating an index of categories where tokens are their parent categories (i.e., treating categories as atomic units, without splitting up their labels) and computing the same features.

To rank candidates, all structure- and content-based features are fed into a Random Forest regressor.

5 CATEGORY SELECTION

Next, we select categories that are deemed appropriate, by ranking the candidates generated in the previous section, then pruning the ranked list. Specifically, we start by identifying the potential parents of each candidate category (Sect. 5.1). Our final ranking then considers the place of the candidate category in the hierarchy (Sect. 5.2), thereby addressing the challenges of *hierarchy* (**C2**), *redundancy* (**C3**), and *importance* (**C4**).

5.1 Parent Category Identification

An important element of our approach is to find the place of the candidate category in the hierarchy. This is cast as a ranking problem: given a candidate category, return a ranked list of possible parent categories.

It is intuitive to think that a category would be named similar to its parent categories. Indeed, we find that nearly 90% of the category-parent ($\langle c, c_p \rangle$) pairs in Wikipedia share at least one term, e.g., ⟨*Companies established in 1974*, *Clothing companies established*

[2] The network allows to copy words via pointing as well as generate words from a fixed vocabulary.

Table 1: Features for category classification. Initial ranking (in Sect. 4.2) uses the first two blocks of features, while final ranking (in Sect. 5.2) uses all features.

Feature	Explanation	#Features
I. Structure-based features		
$\|c\|$	Length of the category (number of terms)	1
$IsPrepos(c)$	Binary indicator whether c contains prepositions	1
$IsStopwords(c)$	Binary indicator whether c contains stopwords	1
$IsEntity(c)$	If c is an entity	1
$\|C_e\|$	Number of categories if c is an entity	1
$CatNum_{aggr}(E_c)$	Aggregation of the number of categories for entities contained in c	3
$TermFreq_{aggr}(\mathcal{T}_c)$	Aggregation of term frequencies for terms in c	3
II. Content-based features		
$CatNameSim(c, C)$	Top-k BM25 scores using category labels	k
$EntityNameSim(c, \mathcal{E})$	Top-k BM25 scores using entity labels	k
$EntitySim(\mathcal{E}_c, \mathcal{E}_{c'})$	Top-k BM25 scores using member entities	k
$EntityOverlap(\mathcal{E}_c, \mathcal{E}_{c'})$	Top-k member fractions against input category ($\|\mathcal{E}_c \cap \mathcal{E}_{c'}\|/\|\mathcal{E}_c\|$)	k
$EntityOverlap2(\mathcal{E}_c, \mathcal{E}_{c'})$	Top-k member fractions against candidate category ($\|\mathcal{E}_c \cap \mathcal{E}_{c'}\|/\|\mathcal{E}_{c'}\|$)	k
$ParentCatSim(C_c, C_{c'})$	Top-k BM25 scores using categories of c	k
$ParentCatOverlap(C_c, C_{c'})$	Top-k category fractions against input category ($\|C_c \cap C_{c'}\|/\|C_c\|$)	k
$ParentCatOverlap2(C_c, C_{c'})$	Top-k category fractions against candidate category ($\|C_c \cap C_{c'}\|/\|C_{c'}\|$)	k
III. Category importance features		
$Importance_{aggr}(c)$	Aggregation of the number of categories containing each segment of the category name	3
$Inlinks_{aggr}(c)$	Aggregation of the numbers of inlinks of member entities	3
$Outlinks_{aggr}(c)$	Aggregation of the numbers outlinks of member entities	3
$GraphSize(c, c_p)$	Total number of member entities, siblings and parent categories	1
$GraphStat(c, c_p)$	Aggregation of the number categories of member entities in the parent category (c_p)	3

in 1974⟩. However, there are also numerous examples of categories that share terms with many other non-related categories, e.g., *Dragons* (mythological monsters) and *Dragon age* (fantasy role-playing game). In other cases, the category and its parent only share the topic, but not any of the terms, e.g., ⟨*Middle-earth Valar*, *Fictional deities*⟩. Additionally, the Wikipedia category system suffers from the violation of the transitivity principle, i.e., a category may contain irrelevant subcategories [16], e.g., ⟨*Flight*, *Motion (physics)*⟩. Simple term-based matching techniques are thus unlikely to be sufficient, making it an extremely challenging task.

To overcome the above issues, we construct a topic graph based on all (existing) $\langle c, c_p \rangle$ pairs in Wikipedia. Inspired by Lawrie et al. [18], we aim to find topic words for multi-document summarization. The model is composed of all conditional probabilities $P(t_a|t_b)$, where t_a is a topic term in c_p and t_b is a category term in c. For simplicity, we will be referring to terms, noting that these can also be multi-word phrases. We segment the categories by proposition. For example, the parent category of *1903 establishments in Colombia* is *1903 establishments in South America*, and we segment them to {*1903 establishments, South America*}, and {*1903 establishments, Colombia*} respectively, and take *South America* as a and *Colombia* as t_b. We segment categories by tokens for those that do not have prepositions. Then, we set

$$P(t_a|t_b) = \frac{n(t_a, t_b)}{n(t_b)}\,, \tag{4}$$

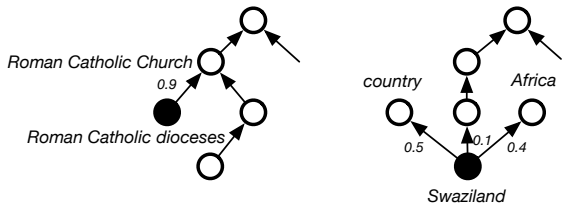

Figure 3: Excerpt from the topic graph, which is utilized both for query expansion and for ranking parent categories. When used for query expansion, the selected topic words for a candidate category *Swaziland* would include *country* and *Africa*. When used for ranking, $P(t_a|t_b)$ between *Roman Catholic Church* and *Roman Catholic dioceses* is 0.9.

where $n(t_a, t_b)$ is the number of $\langle c, c_p \rangle$ pairs where c_p contains t_a and c has t_b, and $n(t_b)$ is the number of categories containing t_b. We form a graph by considering each parent topic term and category term as vertices, and assign $P(t_a|t_b)$ as the weight of the edge between them.

This topic graph, illustrated in Fig. 3, is then leveraged in two orthogonal ways: for query expansion and for ranking.

5.1.1 Query Expansion. The name of the category c can be used as a keyword query to find related categories. Additionally, we propose a query expansion mechanism to generate an expanded query. We leverage the topic graph to add the top topic parent terms. Instead of using a fixed rank-based cutoff, we employ dynamic thresholding. Specifically, we set $\frac{1}{\alpha} \max_{t_a} P(t_a|t_b)$, where $t_b \in c$, as the threshold. Notice that this ensures that at least one topic term is always selected. Terms with weights exceeding the threshold are chosen and put in the topic set $\tilde{T}_t$. Terms in the set will form a new query by appending the topic words to the original query, i.e.,

$$\tilde{c} = c \cup \tilde{T}_c \ , \tag{5}$$

where $\tilde{T}_c = \cup_{t \in c} \tilde{T}_t$ is the entire set of the topic parent terms of c. Then, c or $\tilde{c}$ is used as a keyword query to retrieve parent categories.

5.1.2 Ranking using Topic Graph. Given a keyword query q, which may be the category name c or the expanded query $\tilde{c}$ (cf. Eq. (5)), we propose the following *hierarchy-based* retrieval model based on the topic graph:

$$\phi_H(q, c_p) = \frac{\sum_{t_a \in c_p} \sum_{t_b \in q} P(t_a|t_b)}{|c_p| \cdot |q|} \ , \tag{6}$$

where ϕ stands for the retrieval score, and $|c_p|$ and $|q|$ are measured in the number of segmented terms. This hierarchy-based model can then be combined with a content-based retrieval model (here: BM25) in a simple multiplicative manner:

$$\phi_{combined}(q, c_p) = \phi_{BM25}(q, c_p) \cdot \phi_H(q, c_p) \ . \tag{7}$$

5.2 Final Ranking

The final ranking step considers the place of each category in the hierarchy. Given a candidate category c and the set of its top-k possible parent categories C_p, we score each $\langle c, c_p \rangle$ pair ($c_p \in C_p$), denoted as ψ. Then, the final score for each candidate category is computed using:

$$score(c) = \max\{\psi(c, c_p) | c_p \in C_p\} \ . \tag{8}$$

The above score determines the order in which suggestions are presented to Wikipedia editors. Additionally, we filter out low quality suggestions, whose score is below a pre-defined threshold (i.e., $score(c) < \gamma$).

To estimate $\psi(c, c_p)$, we use the same feature-based supervised learning approach as for the initial ranking (cf. Sect. 4.2), but introduce a third group of features. These *category importance features* are listed in the bottom block of Table 1. We assume that the importance of a category depends on its content. To capture content importance, we segment the candidate category into the set $\mathcal{S}$ using the following rules: prepositions are isolated, longest prefix and suffix are kept if they can be found in other categories. E.g., *Geography of North Yorkshire* is segmented as {*Geography*, *of*, *North Yorkshire*} (as *North Yorkshire* is the longest prefix found in other categories before a preposition). We estimate a category's importance by aggregating those of its segments:

$$Importance_{aggr}(c) = aggr_{s \in \mathcal{S}}\{n(s)\} \ , \tag{9}$$

where $n(s)$ is the number of (existing) categories that contain the segment s in their name and $aggr$ is an aggregator function (*max*, *sum*, or *avg*). Alternatively, category importance can also be characterized by hierarchical properties. Since we know the corresponding parent category c_p, we can consider the number of member entities $\mathcal{E}_c$ of the candidate category (as before, approximating it by taking all entities that are present in the input set), number of siblings, as well as the size of other categories that the member entities $\mathcal{E}_c$ belong to. Apart from the number of member entities, their importance also matters. We estimate the member entities' importance by their number of Wikipedia inlinks and outlinks.

6 EXPERIMENTAL SETUP

We describe the creation of purpose-built test collections and detail our experimental setup.

6.1 Test Collections

Since no test collection exists for our task, we need to develop evaluation resources for the end-to-end task as well as for specific components of our pipeline.

6.1.1 End-to-end Evaluation. Recall that our input is a set of entities in a Wikipedia page, and the output is a ranked list of category suggestions, presented to a Wikipedia editor (cf. Fig. 1). Rather than in random paragraphs, entity sets are more likely to appear in semi-structured formats such as tables and lists [40]. We sample tables/lists from a Wikipedia dump as inputs and try to heuristically recover the categories that could be created based on the set of entities contained in them. We do so by leveraging the categories that are associated with the corresponding Wikipedia page.

Specifically, our test collection consists of over 10k input tables/lists that are sampled from Wikipedia. We limit ourselves to tables/lists that contain a reasonable amount of information, that is, have at least five entities, which is the entity set E. For each input entity set E, we obtain the ground truth categories based on the categories that are assigned to the embedding Wikipedia page. For each of the page's categories c, we check if over half of the entities in E are members of that category. If yes, then c is added to the ground truth, i.e., is a good suggestion. Then, category c, along with all its subcategories, is removed from the page as well as from the Wikipedia category system.[3] Our aim will thus be to "rediscover" c based on the input entities (The remaining categories of the page will be utilized as part of the input.) We avoid "trivial" categories, that is, when the name of the category is the same as the title of the corresponding Wikipedia page. Further, we make sure that no pair of entity sets have identical ground truth (so as to avoid "leakage" between training and test data). Our test collection comprises of 10,542 tables/lists, originating from 10,149 Wikipedia pages (a page might contain multiple tables/lists). On average, there are 1.55 correct category suggestions for entity set in the ground truth. Finally, the test collection is split 80/10/10 into train/validation/test splits.

6.1.2 Category Ranking. The category ranking component is used in both the initial and final ranking steps of our pipeline (cf. Sects. 4.2 and 5.2). To train a machine-learned model, we require a set of positive and negative category examples. While the former is straightforward, the selection of negative ("bad") categories is challenging

[3] We manually exclude a handful of general categories, like *Living people*.

as those categories are non-existent. This is the very fact we exploit: categories that existed for a while but got removed from Wikipedia are bad ones, while those that still exist are likely to be good ones. Thus, we take snapshots of the Wikipedia category system at three different points in time (2012, 2016, and 2019) and check the existence of a given category across them. If a category is present in all three snapshots (2012, 2016, and 2019), it is believed to be a sound category (positive example) given its long-lasting existence. In contrast, if a category exists in the 2012 snapshot, but does not appear in both 2016 and 2019, then it is deemed to be a poor category (negative example). We sample a total of 50k positive and 50k negative categories. From this set, we remove categories that had their labels updated or replaced (by Wikipedians) because of naming conventions. E.g., *Speakers of the National Assembly of Mauritius* was replaced with *Speakers of the National Assembly (Mauritius)*. We detect such changes using a set of simple rule-based methods (based on edit distance and member entity overlap). We further subsample 5k negative and 5k positive categories from the remaining categories. Then, the resulting 10k categories are used as training data for the category ranker. Note that some features also consider the parent category (cf. Table 1). Since some categories have multiple parents, we select a single parent category that is the largest one (i.e., has the most member entities).

6.1.3 Parent Category Identification. Additionally, we create a separate test collection for evaluating the parent category identification component. We randomly sample 5k leaf categories as input, and take their corresponding parent categories (2.4 on average) as the ground truth.

6.2 Experimental Setup

6.2.1 Evaluation Measures. Both the end-to-end task and the various components are evaluated using standard rank-based measures (NDCG, MAP, MRR, and Precision) at cut-off k. We measure statistical significance using a two-tailed paired t-test, with Bonferroni correction. We use †/‡ to denote significance at the 0.05 and 0.01 levels, respectively.

6.2.2 Category Generation. We employ three candidate generation methods. The pointer-generation network is based on [33], following the settings used in [12] (which is used to generate titles for Web tables). For NATS [34], we use their publicly released toolkit.[4] We leverage the features of the coverage mechanism and unknown words replacement apart from the pointer-generator network. We train the model for 35 epochs, and set the learning rate to 0.0001. The FAST model is based on [7]. Word embeddings with 128 dimensions are generated using gensim[5] for the training data. We use the Adam optimizer and the same learning rate as for NATS for training the full network. We perform beam search for all three approaches to choose the suitable models. The categories generated by the three models are combined in a candidate pool.

In the initial ranking step, we set $k = 5$ (for content features that consider the top-k similarity scores). We train a regression model based on the 10k samples in our category ranking dataset (cf. Sect. 6.1.2) and apply it to rank the categories in the candidate pool.

[4] https://github.com/tshi04/NATS
[5] https://github.com/RaRe-Technologies/gensim

Table 2: Candidate generation results using three generators (row 1-3) and our initial ranker (row 4). Statistical significance for row $i > 1$ is tested against row $i - 1$.

Method	NDCG@1	NDCG@10	P@5	MRR@10
PG [33]	0.1213	0.2151	0.0614	0.1794
NATS [34]	0.2351‡	0.2459‡	0.0573	0.2483‡
FAST [7]	**0.3735**‡	**0.3821**‡	**0.0948**‡	**0.3876**‡
SCG (Ours)	**0.5137**‡	**0.8098**‡	**0.2383**‡	**0.6759**‡

Specifically, we employ the Random Forest algorithm, with the number of trees set to 1000 and the maximum number of features in each tree set to 10. Note that the initial ranker uses only the structure-based and content-based features in Table 1.

6.2.3 Category Selection. The second component of our pipeline comprises of two steps: parent category identification and final ranking. For parent category identification, we build the topic graph using all the $\langle c, c_p \rangle$ pairs in Wikipedia, excepting the 5k test categories and their parents which we sampled for evaluation (cf. Sect. 6.1.2). Next, we fetch the top 10 parent categories using query expansion. We set α to 2 based on a set of preliminary experiments.

For final ranking, we use all features in Table 1 and train a ranker with the same settings as for the initial ranking step. We filter out low quality suggestions, using a score threshold of $\gamma = 0.1$.

Additionally, we consider BERT [10] as a baseline for the final ranking step. BERT is a highly effective language representation model, which has been designed to be pre-trained from unlabeled text. We take a pre-trained BERT model ("bert_uncased_L-12_H-768_A-12," which has been trained on the English Wikipedia and the BookCorpus) and fine-tune it for a *sentence pair classification* task. Specifically, we compose sentence pairs by taking a Wikipedia category c as the first sentence and its parent category c_p as the second sentence. We utilize the method in Sect. 5.1 to find the parent categories, and use the sentence pair classification score for ranking. Importantly, the pre-trained BERT model has a potential data leakage issue, as the category we want to generate may exist in the corpus that was used for training. Therefore, BERT may have an unfair advantage. Nevertheless, it can be meaningful to see how our approaches fare against it.

7 EVALUATION

We evaluate the performance of category generation (Sect. 7.1), parent category identification (Sect. 7.2), and final ranking (Sect. 7.3).

7.1 Category Generation

The first component of our pipeline is responsible for the generation of specific candidates that are relevant to the input entity set. The results of the three generation models are presented in the top block of Table 2. Pointer-generator (PG) generates 3.5 candidate categories on average for entity set, while NATS and FAST produce 1.5 and 1.6 candidates, respectively. These are ranked by the generation confidence score. After pooling these together, each entity set yields 4.5 candidates on average. Comparing the effectiveness of the three approaches, we can see that NATS outperforms PG

Table 3: Parent category identification results. Statistical significance of the bottom block is tested against the top block (†/‡) and columns two and three against column one (using $^{\diamond}$/$^{\blacklozenge}$ to denote significance at the 0.05 and 0.01 level, respectively).

Method	BM25		Hierarchy-based		Combined	
	MAP@k	MRR@k	MAP@k	MRR@k	MAP@k	MRR@k
Without query expansion ($k = 10$)	0.0714	0.1423	0.1110$^{\blacklozenge}$	0.2342$^{\blacklozenge}$	0.1115	0.2348
Without query expansion ($k = 1000$)	0.0839	0.1536	0.2153$^{\ddagger\blacklozenge}$	0.3639$^{\ddagger\blacklozenge}$	**0.2627**$^{\ddagger\blacklozenge}$	**0.4476**$^{\ddagger\blacklozenge}$
With query expansion ($k = 10$)	0.1901‡	0.3347‡	0.1920‡	0.3302‡	0.2015‡	0.3492‡
With query expansion ($k = 1000$)	**0.2036**‡	**0.3414**‡	0.1160	0.2210	0.1777	0.3178

Table 4: Final ranking (end-to-end category generation) results. Statistical significance for lines 3 and 4 is tested against line 2, and for line 5 it is tested against line 3.

Method	NDCG@1	NDCG@10	P@5	MRR@10
BERT [10]	0.5308	0.8261	**0.2497**	0.6937
Features I	0.5156	0.8149	0.2402	0.6779
Features I+II	0.5516†	0.8290	0.2464†	0.7035†
Features I+III	0.5744‡	0.8372‡	0.2421	0.7195‡
Features I+II+III	**0.6028**‡	**0.8423**†	0.2445	**0.7363**‡

significantly, thanks to its additional features such as the coverage mechanism. FAST outperforms both PG and NATS substantially and significantly. This tells us that making use of the salient information (in this case: tokens scattered in the entity set) from the extraction process can improve the performance of seq2seq models for category generation.

Our initial ranker, *SCG* (short for Simple Category Generator), then ranks these candidates based on structure- and content-based features. The results are displayed in the last row of Table 2. We find that our inexpensive features are very effective in sorting the candidates, improving all metrics substantially and statistically significantly. It should be noted that the comparison between SCG and the individual generators (PG, NATS, and FAST) is not a fair one, as SCF has access to all candidates produced by the individual generators. Nevertheless, these results show that the abstractive summarization methods can produce complementary results to each other, and our SCG method can effectively rank these candidates.

7.2 Parent Category Identification

Next, we evaluate the capability of our parent category identification approach against the test collection developed for this subtask (cf. Sect. 6.1.3). The results are reported in Table 3, using two settings: considering top 10 or top 1000 candidates returned by BM25, and then re-ranking them. We expect that going deeper in the initial BM25 ranking helps to increase coverage, but it also makes the ranking task more difficult by introducing noise. We consider two ways to compose the keyword query. First, we take the name of the category c as is (top block). Second, we apply query expansion and use the expanded query $\tilde{c}$ (bottom block). The columns correspond to the retrieval method that is used: BM25 (first column), hierarchy-based (second column), and the combined method (last column).

The first observation is that our query expansion method substantially improves effectiveness (rows 1 vs. 3 and 2 vs. 4 in the first column). This shows that query expansion could narrow the vocabulary gap caused by the lack of shared tokens between the category and its parent.

Concerning the comparison of the three ranking models without query expansion (first block of three columns), hierarchy-based outperforms BM25 significantly, and the combined method achieves further substantial and significant improvements for $k = 1000$. Notably, for $k = 10$, we do not observe the same improvements. This attests to the utility of the topic graph for effectively ranking a large number of candidates.

When combining the query expansion and ranking methods (bottom block and third column), the performance varies against the settings. With $k = 10$, the method utilizing the expanded query and combined method performs the best; this is the setting that we use for the final ranking, given its good trade-off between effectiveness and computational efficiency. Overall, the combined method without query expansion with $k = 1000$ performs best. These results indicate that it is better to find the topic words in the late phase instead of refining the query at the beginning when we have many candidates. Otherwise, when the set of candidates is small, it is best to combine query expansion with the hierarchy-based method. In summary, the best strategy to identify the parent categories is to perform query refinement and choose the ranking method accordingly, depending on the size of the candidate set.

7.3 Final Ranking

Given the candidate set returned by initial ranking, we identify the corresponding parent categories, and consider $\langle c, c_p \rangle$ pairs as input to the final ranking step. Thus, the results we present in Table 4 are to be seen as the end-to-end evaluation for the whole pipeline. (Note that this step also considers the parent categories, thus the results are not directly comparable to the initial ranking results we presented earlier.) We take BERT as a baseline, and we report its results in the first line of Table 4. For our proposed approach, we report on different combinations of features, such as only structure features (line 2), structure features with content features (line 3), structure features with importance features (line 4), and all features (last line). Comparing the different types of features, we find that content-based features complement the structure features (line 3 vs. line 2), seen by the increase in scores on all the metrics. Importance

Table 5: Example showing the steps of category generation for a given input entity set.

Input	**List of entities**	... Falkland Islands, Gibraltar, Gotland, Greenland, Guernsey, Hitra, Isle of Wight, Jersey, Rhodes, Saare County...
	Context	...The Isle of Man are not members of FIFA or UEFA, as the Isle of Man FA are members of The Football Association (The FA), with similar status to an English county. Since they are not a member of either FIFA or UEFA, they are not eligible to enter either the World Cup or European Championship....
Ground truth		European national and official selection-teams not affiliated to FIFA
Category generation	**SCG**	National football teams in the isle of man, Football teams in the isle of man, Isle of man, European national and official selection-teams not affiliated to FIFA
Category selection	**BERT**	1. National football teams in the isle of man 2. Football teams in the isle of man 3. Isle of man 4. *European national and official selection-teams not affiliated to FIFA*
	Features I+II+III	1. *European national and official selection-teams not affiliated to FIFA* 2. Isle of man 3. Football teams in the isle of man 4. National football teams in the isle of man

features also enhance performance considerably (line 4 vs. line 2). When combining all the features (line 5), we observe further performance improvements. Importance features complement the rest of the features, as can be seen by the increase between line 3 and line 5. As for the comparison against BERT, the NDCG@k and MRR results show that BERT is good at finding items, but not that good at ranking them. Our best method (line 5) outperforms BERT significantly on all the metrics except P@5.

8 ANALYSIS

We perform additional analysis of our results in this section. First, we present a running example, shown in Table 5, to illustrate each of the steps of our pipeline (cf. Sect. 3) for generating and ranking new categories. In this example, the input set of entities is 14 teams in the Isle of Man that are not affiliated to FIFA. The ground truth category, as well as its parent categories, are removed from training data. Apart from the ground truth category, our SCG candidate generator produces three additional candidate categories; these four candidates are then ranked by BERT and by our method (using all feature sets). The respective rankings are shown in the bottom two lines in Table 5. Comparing with BERT, which put the ground truth category in the last place, our method successfully placed it at the top rank.

To investigate the practical utility of our approach, we measure how often it can return a good recommendation at a high rank position. Figure 4 shows the distribution of all test cases (i.e., input entity sets) with respect to reciprocal rank. That is, if the first relevant category suggestion was returned at rank position k, then the reciprocal rank is $1/k$. We find that in over 60% of all test cases, a relevant suggestion is returned at the top rank. Further, in over 94% of the input cases there is always at least one relevant suggestion returned within the top 5 rank positions. These results demonstrate that our approach has great merit to be deployed in a practical environment, e.g., as a category suggestion service in Wikipedia.

9 CONCLUSION

Category systems of large-scale knowledge bases are unique and valuable resources that can be utilized in a wide range of information access tasks. However, they are currently created and maintained manually by editors. This paper presents a pipeline approach

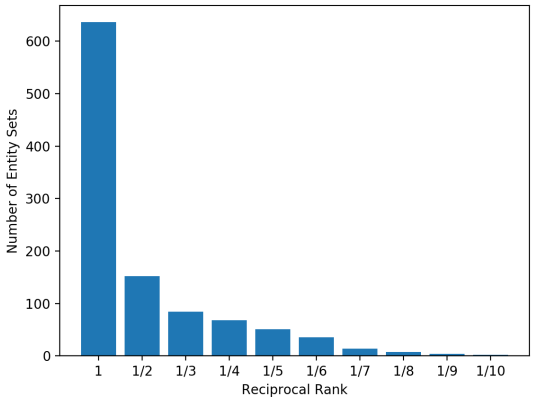

Figure 4: The distribution of the number of entity sets in the test set based on the reciprocal rank at which the first relevant category suggestion is returned.

to generate categories in an automatic manner, given a set of entities and their context as input. We identify four challenges of automatic category generation, which are specificity, hierarchy, redundancy, and importance.

To address the challenge of *specificity*, the first task of candidate generation aims to generate candidate categories that are specific enough given the entity set and context. Given an entity set, we use abstractive summarization models to generate candidates. The candidates are initially ranked, leveraging a set of inexpensive features, based on structure and content, to prune the candidate set. Experimental results show that the abstractive summarization models can generate specific candidates, and the initial ranker can select the most suitable ones.

Parent category identification aims at addressing *hierarchy* and *redundancy* by finding the location of the candidate category in the current category system. To fill the gap caused by the insufficiency of term-based matching and (possible) violation of the transitivity principle, we build a topic graph leveraging all category-parent pairs in the category system. It is utilized in two ways, to expand the query and to rank the parent candidates. Experimental results show that the topic graph can enhance performance by either query expansion or by hierarchy-based ranking.

The final ranking step aims to address the challenge of *importance* by ranking category-parent pairs. Apart from structure- and content-based features, it also considers importance features, which tap into the characteristics of member entities to approximate category importance. This effectively complements the other two groups of features.

We develop a test collection based on Wikipedia categories and perform both end-to-end and component-level evaluation. We show the effectiveness of our approach against a BERT-based baseline and also demonstrate that performance is strong enough to be deployed in a practical application.

REFERENCES

[1] Abeer Al-Arfaj and AbdulMalik Al-Salman. 2015. Ontology Construction from Text: Challenges and Trends. *International Journal of Artificial Intelligence and Expert Systems* 6, Article 2 (2015), 15-26 pages.

[2] Krisztian Balog, Marc Bron, and Maarten De Rijke. 2011. Query modeling for entity search based on terms, categories, and examples. *ACM Trans. Inf. Syst.* 29, 4, Article 22 (Dec. 2011), 22:1–22:31 pages.

[3] Ivan Bedini. 2007. Automatic Ontology Generation : State of the Art. *Molecular Evolution* 44, Article 2 (2007), 226-233 pages.

[4] Chris Biemann. 2005. Ontology Learning from Text: A Survey of Methods. *LDV Forum* 20, 2 (2005), 75–93.

[5] David M. Blei, Thomas L. Griffiths, and Michael I. Jordan. 2010. The Nested Chinese Restaurant Process and Bayesian Nonparametric Inference of Topic Hierarchies. *J. ACM* 57, 2, Article 7 (Feb. 2010), 30 pages.

[6] Paolo Boldi and Corrado Monti. 2016. Cleansing Wikipedia Categories Using Centrality. In *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*. 969–974.

[7] Yen-Chun Chen and Mohit Bansal. 2018. Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 675–686.

[8] Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-Fine Entity Typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. 87–96.

[9] Marek Ciglan, Kjetil Nørvåg, and Ladislav Hluchý. 2012. The SemSets Model for Ad-hoc Semantic List Search. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. 131–140.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018).

[11] David Graus, Daan Odijk, and Maarten de Rijke. 2018. The Birth of Collective Memories: Analyzing Emerging Entities in Text Streams. *Journal of the Association for Information Science and Technology* 69, 6 (2018), 773–786.

[12] Braden Hancock, Hongrae Lee, and Cong Yu. 2019. Generating Titles for Web Tables. In *The World Wide Web Conference (WWW '19)*. 638–647.

[13] Johannes Hoffart, Yasemin Altun, and Gerhard Weikum. 2014. Discovering Emerging Entities with Ambiguous Names. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*. 385–396.

[14] Heng Ji and Ralph Grishman. 2011. Knowledge Base Population: Successful Approaches and Challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11)*. 1148–1158.

[15] Rianne Kaptein and Jaap Kamps. 2013. Exploiting the Category Structure of Wikipedia for Entity Ranking. *Artif. Intell.* 194 (Jan. 2013), 111–129.

[16] Alexander Kirillovich and Olga Nevzorova. 2018. Ontological Analysis of the Wikipedia Category System. In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, (IC3K '18)*. 356–364.

[17] Mark Lauer. 1996. Designing Statistical Language Learners: Experiments on Noun Compounds. *CoRR* cmp-lg/9609008 (1996).

[18] Dawn Lawrie, W. Bruce Croft, and Arnold Rosenberg. 2001. Finding Topic Words for Hierarchical Summarization. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '01)*. 349–357.

[19] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A Large Public Corpus of Web Tables Containing Time and Context Metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*. 75–76.

[20] Denghao Ma, Yueguo Chen, Kevin Chen-Chuan Chang, Xiaoyong Du, Chuanfei Xu, and Yi Chang. 2018. Leveraging Fine-Grained Wikipedia Categories for Entity Search. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. 1623–1632.

[21] Deborah L. McGuinness. 2002. Ontologies Come of Age. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential* (2002), 171–195.

[22] David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30, 1 (2007), 3–26.

[23] Vivi Nastase and Michael Strube. 2008. Decoding Wikipedia Categories for Knowledge Acquisition. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2 (AAAI'08)*. 1219–1224.

[24] Vivi Nastase and Michael Strube. 2013. Transforming Wikipedia into a Large Scale Multilingual Concept Network. *Artif. Intell.* 194 (Jan. 2013), 62–85.

[25] Rasha Obeidat, Xiaoli Fern, Hamed Shahbazi, and Prasad Tadepalli. 2019. Description-Based Zero-shot Fine-Grained Entity Typing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 807–814.

[26] Marius Paşca. 2017. German Typographers vs. German Grammar: Decomposition of Wikipedia Category Labels into Attribute-Value Pairs. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. 315–324.

[27] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A Deep Reinforced Model for Abstractive Summarization. *CoRR* abs/1705.04304 (2017).

[28] Aleksander Pivk. 2005. Automatic Ontology Generation from Web Tabular Structures. *AI Communications* 19 (2005), 83–85.

[29] Xiang Ren, Wenqi He, Meng Qu, Lifu Huang, Heng Ji, and Jiawei Han. 2016. AFET: Automatic Fine-Grained Entity Typing by Hierarchical Partial-Label Embedding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

[30] Dominique Ritze and Christian Bizer. 2017. Matching Web Tables To DBpedia - A Feature Utility Study. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT '17)*. 210–221.

[31] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics (WIMS '15)*. Article 10, 6 pages.

[32] Mark Sanderson and Bruce Croft. 1999. Deriving Concept Hierarchies from Text. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '99)*. 206–213.

[33] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1073–1083.

[34] Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K. Reddy. 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models. *CoRR* (2018).

[35] Emilia Stoica, Marti Hearst, and Megan Richardson. 2007. Automating Creation of Hierarchical Faceted Metadata Structures. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. 244–251.

[36] Xiangyan Sun, Yanghua Xiao, Haixun Wang, and Wei Wang. 2015. On Conceptual Labeling of a Bag of Words. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI '15)*. 1326–1332.

[37] Mohamed Amir Yosef, Sandro Bauer, Johannes Hoffart, Marc Spaniol, and Gerhard Weikum. 2012. HYENA: Hierarchical Type Classification for Entity Names. In *Proceedings of COLING 2012: Posters*. 1361–1370.

[38] Jonathan Yu, James A. Thom, and Audrey Tam. 2007. Ontology Evaluation Using Wikipedia Categories for Browsing. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management (CIKM '07)*. 223–232.

[39] Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. 255–264.

[40] Shuo Zhang and Krisztian Balog. 2019. Auto-completion for Data Cells in Relational Tables. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19)*. 761–770.

[41] Shuo Zhang and Krisztian Balog. 2020. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM Trans. Intell. Syst. Technol.* 11, 2, Article Article 13 (Jan. 2020), 35 pages.

[42] Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. 2020. Novel Entity Discovery from Web Tables. In *Proceedings of The Web Conference 2020 (WWW '20)*. 1298–1308.

[43] Ben Zhou, Daniel Khashabi, Chen-Tse Tsai, and Dan Roth. 2018. Zero-Shot Open Entity Typing as Type-Compatible Grounding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2065–2076.