Learning Defining Features for Categories

Bo Xu[1], Chenhao Xie[1], Yi Zhang[1], Yanghua Xiao[123*], Haixun Wang[4], Wei Wang[1]
[1]School of Computer Science, Fudan University, Shanghai, China [2]Xiaoi Research, Shanghai, China
[3]Shanghai Internet Big Data Engineering and Technology Center, Shanghai, China [4]Facebook, USA
{xubo, 15210240026, 11300290006,shawyh, weiwang1}@fudan.edu.cn, haixun@gmail.com

Abstract

Categories play a fundamental role in human cognition. *Defining features* (short for DFs) are the key elements to define a category, which enables machines to categorize objects. Categories enriched with their DFs significantly improve the machine's ability of categorization and benefit many applications built upon categorization. However, defining features can rarely be found for categories in current knowledge bases. Traditional efforts such as manual construction by domain experts are not practical to find defining features for millions of categories. In this paper, we make the first attempt to automatically find *defining features* for millions of categories in the real world. We formalize the defining feature learning problem and propose a bootstrapping solution to learn defining features from the features of entities belonging to a category. Experimental results show the effectiveness and efficiency of our method. Finally, we find defining features for overall **60,247** categories with acceptable accuracy.

1 Introduction

Great efforts have been dedicated to constructing a variety of knowledge bases in recent years. Many knowledge repositories such as Yago [Suchanek *et al.*, 2007], DBpedia [Auer *et al.*, 2007], Freebase [Bollacker *et al.*, 2008] and Probase [Wu *et al.*, 2012] now are widely available and successfully applied in many real applications, such as type inference [Paulheim and Bizer, 2013], entity linking [Volz *et al.*, 2009], question answering [Unger *et al.*, 2012], etc.

Categories are one of the most important elements in knowledge bases. Humans understand the world by categories [Aristotle and Ackrill, 1963]. Categorization, the process in which objects or instances are recognized, differentiated and understood, is the most important cognition procedures of humans. How humans categorize an object is an intriguing topic that can be traced back to thousands of years ago. Many theories about categorization, such as *classical categorization* [Aristotle and Ackrill, 1963], *conceptual clustering* [Fisher, 1987; 1996] and *prototype theory* [Lakoff, 1990] have been proposed. One of them stated by `Aristotle` [Aristotle and Ackrill, 1963] claims that objects are grouped as categories by their similar properties.

Thus, an interesting problem naturally arises: *how to enable machines to do property-based categorization?* The prerequisites of property-based categorization are knowledge bases consisting of categories and their properties. Given these knowledge bases, machines are able to categorize an object by comparing object's properties against those of categories. To ensure the accuracy of the categorization, we hope the properties of categories can precisely define the category. That is, we need *defining features* (short for DF) to characterize a category. For example in Figure 1, we use `Multicellular, Eukaryotic, Kingdom Animalia` to define category `ANIMAL`, and these properties together are called DFs of the category.

Theoretically, defining features are assumed to establish the **necessary** and **sufficient** conditions to characterize the meaning of the category. That is any entity with the defining features should belong to the category, and any entity belonging to the category must contain the defining features. However, it is not practical to find precise DFs for categories due to three reasons, First, *we can always find outliers in a category*. For example one may consider `can fly` is part of DFs of category `BIRD`, but `ostrich` can not fly. Second, *defining a category is always subjective*. For example, different people has disparate beliefs about what is a `good person`. Third, *real data is often sparse, which prevents us from finding precise DFs*. When more member entities and their features are given, DFs of categories are subject to change. Hence, more realistically, instead of finding the exact defining features, we find a set of features that are most probably to be the DFs of the category. But for the simplicity of description, we still use DFs to represent the features that we are looking for.

Categories enriched with defining features enable the *property-based categorization*, which is a significant complement to the current categorization techniques that heavily rely on isA relations. Many taxonomies materializing

*Correspondence author. This paper was supported by the National NSFC (No.61472085, 61171132, 61033010, U1509213), by National Key Basic Research Program of China under No.2015CB358800, by Shanghai Municipal Science and Technology Commission foundation key project under No.15JC1400900.

isA relations between instances and their categories, such as Yago [Suchanek *et al.*, 2007], WikiTaxonomy [Ponzetto and Strube, 2008], Probase [Wu *et al.*, 2012] have been constructed. However, it is hard to use these taxonomies to categorize emerging entities since in most cases we only have some description about an entity's specific property instead of its category. For example when human built the first plane, we had no way to categorize it according to the taxonomy in our mind although we knew all its specific properties. Instead, given the properties of categories, the comparison to existing category properties can easily activate the creation of a new category.

Property-based categorization is indispensable for many real applications. The first application is knowledge base completion. Given the DFs of one category, we can infer that all its members have the corresponding DFs, which can complete facts about the entities. The second application is search intent understanding. To understand the intent behind the query *birds red-breasted*, we just need to compare the DFs of categories with the query. From Figure 1, we know that `Robin` is a `Bird` with `red-breasted` DF. So we can infer the user's intent is finding robins. Actually, property-based categorization can benefit many applications where isA relation based categorization is applicable, such as topic search [Wang *et al.*, 2010], short text understanding [Hua *et al.*, 2015] and web tables understanding [Wang *et al.*, 2012].

Despite its great value, DFs of categories are still very rare. Many knowledge bases in general contain rare DFs for categories. For example, Probase contains millions of categories, but has no DFs associated with each category. DBpedia has rich property-value information for entities only. Inferencing DFs of categories from their entities is not easy. In the traditional efforts, psychologists manually construct DFs for a very limited number of popular categories, such as *birds*, *animals*, and *cars* [Collins and Quillian, 1969; Tulving, 1972; Smith *et al.*, 1974]. However, it is impractical to find the DFs for millions of categories in the real world manually, especially considering that there are millions of *tail* categories which are unfamiliar to most of us.

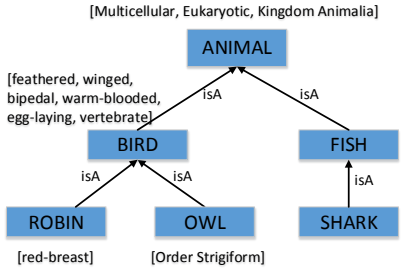

Figure 1: An example of DFs of categories.

The goal of this paper thus is to automatically find *defining features* for millions of categories in the real world. Now, we have a strong foundation to solve this problem. Many knowledge bases, such as DBpedia [Auer *et al.*, 2007], Yago [Suchanek *et al.*, 2007] and Freebase [Bollacker *et al.*, 2008], contain rich categories as well as their members. DBpedia further provides rich structured information of instances. These knowledge bases make it possible to automatically mine the DFs for millions of categories. With these data, we formalize the DF finding problem and propose a bootstrapping algorithmic solution that learns the DFs from entities belonging to the category. To the best of our knowledge, this is the first effort to automatically find DFs for millions of categories.

2 Preliminary

Our ultimate goal is to find DFs for categories. However, there are rare descriptions about categories that we can use to mine DFs. Hence, we turn to mining defining features for a category from the entities belonging to the category. Because many online knowledge bases contain rich entity information. Our idea is that *if most members of a category share a set of features, these features are likely to be the DFs of the category*. Next, we clarify the representation of entities and categories in the current knowledge bases.

2.1 Representation of Entity

We extract entity information from DBpedia, a large-scale well-structured online knowledge repository. We use two kinds of information widely available in DBpedia to represent entities. The first is *property-value* pairs of an entity. They can be extracted from **Infobox** of the entity. The second is *Types* of an entity, which are classes that entities belonging to. We use *Property-Value (PV)* and *Type* features to represent an entity. Each feature about an entity thus essentially is an assertion about the fact that the entity belongs to a type or has a certain property-value pair. Table 1 shows the representation of the film `Inception`. One of its PV features `(language, English)` claims that the film's `language` is `English`, and one of its type features claims that it is a `Film`.

PV Features	(language, English), (director, Christopher Nolan)
Type Features	(Type, Film), (Type, Work)

Table 1: Two classes of features of entity `Inception`.

2.2 Representation of Category

A category is also represented by **PV** and **Type** features, which can be inferred from its member entities. For example, the Wikipedia category `films directed by Christopher Nolan` can be defined by its type feature `film`, and PV feature `(director, Christopher Nolan)`.

One of the key issues to define a category is the granularity of the type. A type always has its subtypes or super types. More generally, types are organized as a hierarchy by the *subClassOf* relations among types. For example, a more general type of `films directed by Christopher Nolan` is **work** and the category can be defined with **work** and additional characterization of film. Obviously, such a definition is also correct but not concise. In general, selecting an appropriate type to define a category is not trivial. In this paper, we exploit DBpedia type hierarchy, and we always select the *most*

specific type to define a category. For `films directed by Christopher Nolan`, we use its basic type `film` as part of the DFs.

3 Defining Feature Mining

We first present our solution framework, then elaborate each main step in the framework.

3.1 Framework

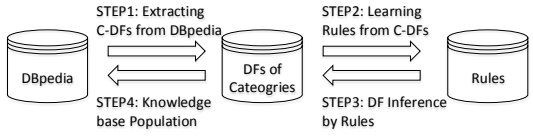

Figure 2: Framework of Defining Feature Mining.

We propose a bootstrapping approach to find the DFs of categories in DBpedia. We illustrate its iterative procedure in Figure 2. We refer to a category and its DFs as a C-DFs pair. Each iteration consists of four major steps. In the first step, we extract DFs of categories from DBpedia. In the second step, we learn rules from the C-DFs pairs extracted in the previous step. In the third step, we extract more C-DFs pairs from these rules. Finally, we populate DBpedia by using C-DFs discovered so far. The above procedure is repeated iteratively until no new DFs/Rules/Knowledge generated/discovered in any step.

3.2 Extracting C-DFs from DBpedia

We first give our metric to quantify how likely a feature set is the DFs of a category, then we formalize the defining feature learning problem, finally we elaborate our solution.

Scoring Candidate Feature Set We first elaborate how to measure the *"goodness"* of a feature set $\mathbf{f} = (f_1, f_2, \ldots, f_k)$ to be the DFs for a category c. The goodness score expresses our belief that a candidate feature set is the DFs of the category. We say one entity has feature set $\mathbf{f}$, if it has all the features $f_1, f_2, \ldots, f_k$ in $\mathbf{f}$. We use $P(\mathbf{f}|c)$ to denote the probability that an entity in category c has the feature set $\mathbf{f}$. We use $P(c|\mathbf{f})$ to denote the probability that an entity, which has feature set f, belongs to category c. These probabilities are defined as follows:

$$P(\mathbf{f}|c) = \frac{\#\textit{ of entities in c that have } \mathbf{f}}{\#\textit{ of entities in c}} \quad (1)$$

$$P(c|\mathbf{f}) = \frac{\#\textit{ of entities in c that have } \mathbf{f}}{\#\textit{ of entities that have } \mathbf{f}} \quad (2)$$

Thus, we define the *"goodness"* score of a feature set $\mathbf{f}$ with respect to a category c as follows:

$$score(c, \mathbf{f}) = P(\mathbf{f}|c) \times P(c|\mathbf{f}) \quad (3)$$

The higher the score is, the more likely that this feature set is the DFs of the category. In particular, our score is a trade-off between *intra-class similarity* and *inter-class dissimilarity* of objects, where objects are described as a set of feature set [Fisher, 1987]. Intra-class similarity is reflected by $P(\mathbf{f}|c)$. The larger this probability, the greater the proportion of category members sharing $\mathbf{f}$ and the more *predictable* the feature set is of category members. Inter-class similarity is a function of $P(c|\mathbf{f})$. The larger this probability, the fewer the objects in contrasting categories that share this feature set and the more *predictive* the feature set is of the category.

Problem Model Theoretically, when $\mathbf{f}$ is the DFs of c, both $P(\mathbf{f}|c)$ and $P(c|\mathbf{f})$ equal to 1, and consequently $score(c, \mathbf{f}) = 1$. Because as DFs of c, all entities in c have $\mathbf{f}$ and all entities that have $\mathbf{f}$ belong to c. However, this is the ideal case. In reality, due to the *incompleteness* of knowledge base, $score(c, \mathbf{f})$ is far less than 1. Because some entities might miss some features in the knowledge base, which would underestimate $P(\mathbf{f}|c)$. Some entities might miss some categories, which would underestimate $P(c|\mathbf{f})$. Hence, more realistically, we expect to find a feature set $\mathbf{f}$ which maximizes the score:

$$\hat{\mathbf{f}}(c) = \arg\max_{\mathbf{f}} score(c, \mathbf{f}) \quad (4)$$

Still due to the incompleteness of knowledge bases, some features might be absent in knowledge base. As a result, for some categories, it is possible that we cannot find a feature set that has a large enough $score(c, \mathbf{f})$. Hence, we set a threshold α ($0 < \alpha \leq 1$) to prune any feature set whose goodness score is below the threshold.

Algorithm A naive solution to find the best feature set suffers from exponential cost. A category with N candidate features would have $2^N - 1$ different feature sets, only a few of which can be used as DFs. Hence, precisely computing the "goodness" score for each possible feature set is costly and wasteful. To solve this problem, we propose to use a frequent itemset mining based method to speed up the computation. The procedure is illustrated in Algorithm 1. For each category c, we first find its frequent feature sets $\mathcal{F}(c)$ by using frequent itemset mining (line 3). Then, we just compute the scores for the frequent feature sets (line 4-10).

Finding frequent feature sets of categories can be modeled as a Frequent Itemset Mining problem. For each category c, items are all features of entities in c. Each entity belonging to c has a transaction consisting of the entities' features. Let *support* be the percentage of instances in a category c that have the feature set. Given a minimum support threshold, we use FP-growth algorithm [Han *et al.*, 2000] to find frequent feature sets. By focusing on only frequent ones, the feature sets to be evaluated are significantly pruned. We illustrate the modeling and the pruning in Example 1.

e_1	e_2	e_3	e_4	e_5
f_1, f_2, f_3, f_4	f_1, f_2, f_5	f_1, f_2, f_6, f_7	f_1, f_2, f_8, f_9	f_3, f_{10}

Table 2: A toy example for frequent feature set mining.

Example 1 (Find frequent itemsets for categories)**.** *In Table 2, one category contains 5 entities and 10 features. To find DFs, a naive method needs to enumerate all the possible itemsets. The search space is* $2^{10} - 1 = 1023$*. If we set the*

minimum support threshold as 0.4, by frequent itemset mining, we obtain only 4 frequent feature sets $\{f_1\}$, $\{f_2\}$, $\{f_3\}$ and $\{f_1, f_2\}$. We only need to compute the scores for these 4 feature sets, and find the one with the largest score as the DFs. This greatly reduces the number of candidate feature sets.

Next, we elaborate how to set the minimum support threshold. As illustrated in Algorithm 1 (in line 3), we directly reuse the goodness score threshold α that is used in Equation 4. The rationality is shown in Theorem 1, which shows that if a feature set is infrequent in c (i.e., the frequency is less than $\alpha|c|$), then its goodness score must be less than α, thus could be pruned.

Algorithm 1 Extracting C-DFs from DBpedia.

Input: Entities and their features; α: minimum score to accept a $\mathbf{f}$ as DF
Output: $\hat{\mathbf{f}}(c)$ for $c \in C$.
1: **for** c in C **do**
2: $\delta \leftarrow 0$ *//maximal score found so far*
3: Find frequent feature sets $\mathcal{F}(c)$ for category c with minimum support α
4: **for** each $\mathbf{f}$ in $\mathcal{F}(c)$ **do**
5: compute $score(c, \mathbf{f})$
6: **if** $score(c, \mathbf{f}) > \alpha$ and $score(c, \mathbf{f}) > \delta$ **then**
7: $\delta \leftarrow score(c, \mathbf{f})$
8: $\hat{\mathbf{f}}(c) \leftarrow \mathbf{f}$
9: **end if**
10: **end for**
11: **end for**

Theorem 1. For a category c, if a feature set $\mathbf{f}$ is infrequent in c (i.e., the frequency is less than the minimum support count $\alpha \times |c|$), then we have $score(c, \mathbf{f}) < \alpha$.

Proof. If $\mathbf{f}$ is infrequent in c, which means that the number of entities in c owning $\mathbf{f}$ is less than $\alpha \times |c|$. This implies that $P(\mathbf{f}|c) < \frac{\alpha \times |c|}{|c|} < \alpha$. As the multiplication of two conditional probabilities, we have

$$Score(c, \mathbf{f}) = P(\mathbf{f}|c) \times P(c|\mathbf{f}) \leq P(\mathbf{f}|c) \quad (5)$$

Thus, we have $score(c, \mathbf{f}) < \alpha$. Hence, this feature set $\mathbf{f}$ will not be the DFs of c. □

3.3 Learning Rules from C-DFs

Due to the incompleteness of knowledge bases, some DFs of categories would be infrequent, thus cannot be found by frequent itemset mining. For these DFs, we first learn rules from C-DFs already discovered and evaluate their quality. Then, we use rules of high quality to extract more C-DFs. Next we illustrate how to learn the rules, and how we use these rules to obtain more C-DFs.

C: Films directed by (.*)	DFs
Christopher Nolan	(Type, Film) (director, Christopher Nolan)
James Cameron	(Type, Film) (director, James Cameron)
Steven Spielberg	(Type, Film) (director, Steven Spielberg)
David Fincher	(Type, Film) (director, David Fincher)
Ben Affleck	(Type, Film) (director, Ben Affleck)
Left part of rule: r_l	Right part of rule: r_r
Films directed by <Person>	(Type, Film) (director, <Person>)

Table 3: Five categories and their DFs in DBpedia, all of them match the same rule.

Preliminaries Let c be a category, $\hat{c}$ be its name. Let P be a string pattern. In this paper, we focus on the conceptualized pattern. That is we generate the pattern by replacing a substring (say s) by its concepts [1] if s represents an instance of a concept. For example for all the categories in Table 3, their names match the pattern `Films directed by <Person>`, where `<>` is a placeholder and `Person` is the type of instances that should be placed therein. When instantiating the pattern, we just need to replace an instance of `Person` at the placeholder. We define `Match` as the operator determining whether a string matches a pattern. For each PV and type feature of a category, we can define a predicate. For example, $Type(c, \texttt{Film})$ claims that a category c is of type `Film`. The PV feature can be generalized to a conceptualized pattern by conceptualizing the value of properties. For example, all PV features in Table 3 share the `(director, <Person>)` pattern, where `<Person>` is the range type of property `director`.

Rules and its generation Now, we are ready to define our rules to populate DFs of categories. From Table 3, we derive the following rule:

$$Match(\hat{c}, \text{``Films directed by <Person>"}) \Rightarrow$$
$$Type(c, \texttt{Film}) \wedge director(c, \texttt{<Person>})$$

Next, we elaborate how we learn the rules. The input is tuples $\{(c_i, \{(p_{ij}, v_{ij})\})\}$, which are the C-DFs pairs we discovered in the previous steps. We develop our rule learning solution based on [Etzioni *et al.*, 2005]. The algorithm proceeds as follows: We start with a set of seed C-DFs pairs (c, PV) generated in Section 3.2. Then, for each seed $(c, PV = \{(p, v)\})$, if one value v in DFs appears in its category name, we generate a candidate rule by replacing the string of v in category name and DFs with the concept of v. Here, we use the value type defined in DBpedia as the concept. The conceptualized category and conceptualized DFs are the left part r_l and right part r_r of the rule r, respectively. Example 2 illustrates the rule generation procedure. Finally, we output good rules according to the metrics defined below.

Example 2 (Rule Generation)**.** Table 3 lists 5 categories that share similar category names and DFs. For each category, the value of `director` appears in their category names. Thus, we use the range type of property `director` (i.e., `<Person>`), which actually is the type of the value instances, to generate the pattern. In this way, we generate a conceptualized category name "`Films directed by <Person>`," and conceptualized DFs `(Type, Film)` and `director(c, <Person>)`.

Quality Metrics of Rules Given a conceptualized rule r : $r_l \Rightarrow r_r$. We use `SupportCount` and `Confidence` to measure its quality. We define:

$$SupportCount(r) = \#\textit{ of C-DFs matching } r_l \textit{ and } r_r \quad (6)$$

$$Confidence(r) = \frac{SupportCount(r)}{\#\textit{ of C-DFs match } r_l} \quad (7)$$

[1] More specifically, we use types in DBpedia.

`SupportCount` measures the significance of the rule by counting the number of C-DFs pairs that match r. `Confidence` measures the conditional probability that a C-DF pair matches r_r given that the pair matches r_l. Rules that satisfy both a minimum support count threshold and a minimum confidence threshold are called *strong*. If a rule is strong, we accept this rule and use this rule to populate the DFs of categories. In our implementation, we only keep the rules with support count at least 10 because we found that most rules with a frequency lower than 10 are meaningless.The minimum confidence threshold will be tuned in the experiments.

3.4 DF Inference by Rules

Given a good rule, we search every category in DBpedia to test whether it matches its left part. If true, we infer that it contains the corresponding DFs specified in the right part of the rule. Note that DFs found by the rule based inference in general are not necessarily the same as the one found by frequent feature set based approach (Section 3.2). Our test shows that rule based inference has a higher quality than frequent feature set based approach. Hence, when there is an inconsistency, we choose the result derived from rule based inference.

Example 3 (Inferring C-DFs from Rules)**.** For example, a new category `Films directed by Martin Scorsese` also matches the left part of the rule in Table 3. Hence, we infer that the DFs of `Films directed by Martin Scorsese` are `(Type, Film)` and `(Director, Martin Scorsese)`

3.5 Knowledge Bases Population

We populate the DBpedia knowledge base by using C-DFs discovered so far. We use the following rules to enrich DBpedia entities with more types, property-value pairs and categories.

- *Rule 1: type completion*. For each C-DFs pair, if an entity belongs to the category c, then enrich the entity with the type from DFs of the category and all the ancestor types.
- *Rule 2: property-value completion*. For each C-DFs pair, if an entity belongs to the category c, then enrich the entity with the **PV** DFs of the category.
- *Rule 3: category completion*. For each C-DFs pair, if an entity contains the feature set DFs, then it must belong to category c. We enrich the entity with the category

Example 4 (Knowledge Base Population)**.** For example, if we know that entity `Inception` belongs to category `Films directed by Christopher Nolan`, then we can infer that it has type `Film` and all the ancestor types of `Film` such as `Work` and `Thing` (from `Rule 1`) and it contains the property-value pair `(director, Christopher Nolan)` (from `Rule 2`). If `Inception` misses the category `Films directed by Christopher Nolan`, by `Rule 3` we can infer its membership from the facts that `Inception` is a `Film` and one of its property-value pairs is `(director, Christopher Nolan)`.

4 Experiments

We use *DBpedia2015-04* version [2] as our dataset. Specifically, we use `DBpedia Ontology`, `Entity Types`, `Entity Infobox` and `Entity Categories` as input, and return DFs for DBpedia categories. There are overall 753,524 DBPedia categories. We run our solution on all these categories, and successfully *find DFs for overall* **60,247** *categories*.

Metric Since no ground truth is available, we resort to humans to judge the quality of DFs of categories with graded scores. The criteria for manual evaluation is show in Table 4. For each DF of a given category, we ask three volunteers to rate the quality of the DFs. We report the average of users' rates as the *quality score*. The higher the quality score, the more likely the feature sets are the DFs of the category.

Score	Meaning	Example
3	Perfect	(Type, album), (Singer, Jay Chou)
2	One DF not found / One non-DF found	(Type, album), (Singer, Jay Chou) (genre, Pop music)
1	At least one DF found	(Type, single), (Singer, Jay Chou), (genre, Pop)
0	No DFs found	(Type, single), (genre, Pop)

Table 4: Criteria for manual evaluation for category `Jay Chou albums`.

4.1 Extracting C-DFs from DBpedia

Recall that our core idea is finding DFs from only frequent feature sets. The minimum support threshold is set as α, equivalent to the goodness score threshold used for the pruning of the unpromising feature sets. Thus, the setting of α is critical for both efficiency and effectiveness of our solution. In general, with the increase of α, we hope to find DFs of higher quality with less time.

α	# of C-FFs	# of C-DFs	Avg. Quality Score
0.5	5,978,069	38,277	2.31
0.6	4,494,896	29,141	2.49
0.7	3,357,449	20,196	2.51
0.8	2,375,901	12,904	2.52
0.9	1,191,224	6,295	2.55

Table 5: Performance under different α.

We first report the number of frequent feature sets of categories (C-FFs) in Table 5. We can see that the numbers of C-FFs and C-DFs pairs discovered by our approach decrease when α increases. On average, we only need to compute the "goodness" scores for less than 8 feature sets per category (for example, when $\alpha = 0.5$ we have $5,978,069/753,524 < 8$). The computation cost is reduced greatly by pruning infrequent feature sets. We also evaluate the quality of the extracted C-DFs pairs. We randomly select 100 C-DFs pairs and evaluate their average quality score by human judgement. The result is shown in the last column in Table 5. We can see that the quality score increases with α. We notice that when α goes up to 0.6, the increase rate slows down. In order to

[2] http://wiki.dbpedia.org/Downloads2015-04

obtain as more as possible reasonable C-DFs pairs, we set α as 0.6.

4.2 Expanding C-DFs from Rules

Next, we evaluate the precision of C-DFs generated from rules. We use the C-DFs pairs extracted in the previous step as input to generate the rules. We got 87 rules with support count no less than 10. We give some rule examples in Table 6.

Good Rules	Sup.	Conf.
Match($\hat{c}$, `Lakes of <Country>`) ⇒ Type(c, `Lake`) ∧ country(c, `<Country>`)	19	0.82
Match($\hat{c}$, `Films directed by <Person>`) ⇒ Type(c, `Film`) ∧ director(c, `<Person>`)	1,511	0.78
Match($\hat{c}$, `<Company> books`) ⇒ Type(c, `Book`) ∧ publisher(c, `<Company>`)	31	0.76

Table 6: Three good rules in DBpedia.

To get high-quality C-DFs pairs, we need to set a minimum confidence threshold β. We randomly select 100 C-DFs pairs generated from these rules with confidence no less than β, and ask three volunteers to evaluate their quality. From Figure 7, we notice that when the threshold goes up to 0.6, our approach achieves the highest score 3.0. As a result, we use 0.6 as the confidence threshold. Finally, we obtain 33,449 C-DFs pairs with average quality score as 3 from the good rules. By merging with the C-DFs pairs generated with the previous experiment, we totally obtain 58,740 C-DFs pairs with average quality score 2.85.

β	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Avg. Quality Score	2.85	2.88	2.86	3.0	3.0	3.0	3.0

Table 7: Performance under different β.

4.3 Knowledge Population by C-DFs

	Count	Precision
(entity, type)	207,499	98%
(entity, property, value)	713,021	92%
(entity, category)	402,719	100%

Table 8: Performance of knowledge population.

By using the C-DFs pairs from previous step, we populate DBpedia with three different classes of information. As shown in Table 8, we obtain 207,499 new types, 713,021 new property-value pairs and 402,719 new categories for DBpedia entities. To evaluate their precision, we also randomly select 100 of them and judge the correction by volunteers. Finally, the precision is 98%, 92% and 100%, respectively. The precision of property-value pairs is lower than others, because there exist some categories whose **PV** DFs have a logically `OR` instead of `AND` relations to define the category. For example, in category `Real Madrid C.F. players` lists Real Madrid football players past or present. Its **PV** DFs is either `(team, Real Madrid C.F.)` or `(formerteam, Real Madrid C.F.)`. Choosing any one of them as **PV** DFs would reduce the precision of new property-value pairs of entities, while others are not affected.

4.4 Performance in different iterations

Iteration	# of Cumulative C-DFs	Avg. Quality Score
1	58,740	2.85
2	60,247	2.82

Table 9: Performance in different itereations.

We also evaluate the performance in different iterations. As shown in Table 9, we finally obtain 60,247 new C-DFs with average quality score 2.82. The majority of new C-DFs are extracted from iteration 1. After populating the DBpedia, some non-DFs of the categories may become frequent, hence reduce the precision of C-DFs. For example, the DFs of category `EPMD albums` in iteration 1 are `(Type, Album)` and `(Artist, EPMD)`, while in iteration 2, the DFs are `(Type, Album)`, `(Artist, EPMD)` and `(producer, Erick Sermon)`.

5 Related Work

Conceptual Clustering Our work is similar to *Conceptual Clustering*, which clusters instances into different categories based on their property-value pairs. COBWEB [Fisher, 1987; 1996] attempts to maximize both the probability that two instances in the same concept have PV pairs in common, and the probability that instances from different concepts have different property-value pairs. In *Conceptual Clustering*, categories are unknown. While in our setting, categories are given, and our goal is finding DFs of them.

Typical Property or Property-Value Pairs Finding Many efforts have been dedicated to finding typical/frequent property or property-value pairs from categories. Yago [Suchanek *et al.*, 2007] uses hand-crafted rules to find frequent property-value pairs from Wikipedia categories. Catriple [Liu *et al.*, 2008] uses some predefined templates about categories and their super categories to obtain some frequent property-value pairs. However, both of them specified the rules by humans, while in our work, we learn the rules automatically.

Other approaches tend to find typical properties from web corpus. Most of them focus on finding high quality syntactic patterns to extract and score the properties of categories on sentences [Ravi and Paşca, 2008; Bellare *et al.*, 2007; Pasca *et al.*, 2006], but it is not suitable for knowledge bases.

6 Conclusion

In this paper, we devote our efforts to learning defining features for millions of categories. We formalize the defining feature mining problem and propose a bootstrapping solution. We learn defining features of categories from the features of entities that belong to the category. We propose to use frequent feature set to prune the search space of defining features. We conduct extensive results on real knowledge bases. Our results verify the effectiveness and efficiency of our models and solutions. We find DFs for **60,247** categories with acceptable quality in total.

References

[Aristotle and Ackrill, 1963] Aristotle and J.L. Ackrill. *Aristotle's Categories and De Interpretatione*. Clarendon Aristotle series. Clarendon Press, 1963.

[Auer *et al.*, 2007] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[Bellare *et al.*, 2007] Kedar Bellare, Partha Talukdar, Giridhar Kumaran, Fernando Pereira, Mark Liberman, Andrew McCallum, and Mark Dredze. Lightly-supervised attribute extraction for web search. In *The NIPS 2007 Workshop on Machine Learning for Web Search*, 2007.

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[Collins and Quillian, 1969] Allan M. Collins and M. Ross Quillian. Retrieval time from semantic memory. *Journal of verbal learning and verbal behavior*, 8(2):240–247, 1969.

[Etzioni *et al.*, 2005] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

[Fisher, 1987] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn.*, 2(2):139–172, September 1987.

[Fisher, 1996] Douglas H. Fisher. Iterative optimization and simplification of hierarchical clusterings. *CoRR*, cs.AI/9604103, 1996.

[Han *et al.*, 2000] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[Hua *et al.*, 2015] Wen Hua, Zhongyuan Wang, Haixun Wang, Kai Zheng, and Xiaofang Zhou. Short text understanding through lexical-semantic analysis. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 495–506. IEEE, 2015.

[Lakoff, 1990] George Lakoff. *Women, fire, and dangerous things: What categories reveal about the mind*. Cambridge Univ Press, 1990.

[Liu *et al.*, 2008] Qiaoling Liu, Kaifeng Xu, Lei Zhang, Haofen Wang, Yong Yu, and Yue Pan. Catriple: Extracting triples from wikipedia categories. In *The Semantic Web*, pages 330–344. Springer, 2008.

[Pasca *et al.*, 2006] Marius Pasca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts - step one: The one-million fact extraction challenge. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1400–1405. AAAI Press, 2006.

[Paulheim and Bizer, 2013] Heiko Paulheim and Christian Bizer. Type inference on noisy rdf data. In *The Semantic Web–ISWC 2013*, pages 510–525. Springer, 2013.

[Ponzetto and Strube, 2008] Simone Paolo Ponzetto and Michael Strube. Wikitaxonomy: A large scale knowledge resource. In *ECAI*, volume 178, pages 751–752, 2008.

[Ravi and Paşca, 2008] Sujith Ravi and Marius Paşca. Using structured text for large-scale attribute extraction. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 1183–1192, New York, NY, USA, 2008. ACM.

[Smith *et al.*, 1974] Edward E. Smith, Edward J. Shoben, and Lance J. Rips. Structure and process in semantic memory: A featural model for semantic decisions. *Psychological review*, 81(3):214, 1974.

[Suchanek *et al.*, 2007] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[Tulving, 1972] Endel Tulving. Episodic and semantic memory. *Organization of Memory. London: Academic*, 381(e402), 1972.

[Unger *et al.*, 2012] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web*, pages 639–648. ACM, 2012.

[Volz *et al.*, 2009] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk-a link discovery framework for the web of data. *LDOW*, 538, 2009.

[Wang *et al.*, 2010] Yue Wang, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. Toward topic search on the web. Technical report, Citeseer, 2010.

[Wang *et al.*, 2012] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. Understanding tables on the web. In *Conceptual Modeling*, pages 141–155. Springer, 2012.

[Wu *et al.*, 2012] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 481–492. ACM, 2012.