Unsupervised Template Mining for Semantic Category Understanding

Lei Shi[1,2*], Shuming Shi[3], Chin-Yew Lin[3], Yi-Dong Shen[1], Yong Rui[3]
[1]State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
[3]Microsoft Research
{shilei,ydshen}@ios.ac.cn
{shumings,cyl,yongrui}@microsoft.com

Abstract

We propose an unsupervised approach to constructing templates from a large collection of semantic category names, and use the templates as the semantic representation of categories. The main challenge is that many terms have multiple meanings, resulting in a lot of wrong templates. Statistical data and semantic knowledge are extracted from a web corpus to improve template generation. A nonlinear scoring function is proposed and demonstrated to be effective. Experiments show that our approach achieves significantly better results than baseline methods. As an immediate application, we apply the extracted templates to the cleaning of a category collection and see promising results (precision improved from 81% to 89%).

1 Introduction

A semantic category is a collection of items sharing common semantic properties. For example, all cities in Germany form a semantic category named "city in Germany" or "German city". In Wikipedia, the category names of an entity are manually edited and displayed at the end of the page for the entity. There have been quite a lot of approaches (Hearst, 1992; Pantel and Ravichandran, 2004; Van Durme and Pasca, 2008; Zhang et al., 2011) in the literature to automatically extracting category names and instances (also called is-a or hypernymy relations) from the web.

Most existing work simply treats a category name as a text string containing one or multiple words, without caring about its internal structure. In this paper, we explore the *semantic* structure of category names (or simply called "categories").

[*]This work was performed when the first author was visiting Microsoft Research Asia.

For example, both "CEO of General Motors" and "CEO of Yahoo" have structure "CEO of [company]". We call such a structure a *category template*. Taking a large collection of open-domain categories as input, we construct a list of *category templates* and build a mapping from categories to templates. Figure 1 shows some example semantic categories and their corresponding templates.

Templates can be treated as additional features of semantic categories. The new features can be exploited to improve some upper-layer applications like web search and question answering. In addition, by linking categories to templates, it is possible (for a computer program) to infer the semantic meaning of the categories. For example in Figure 1, from the two templates linking to category "symptom of insulin deficiency", it is reasonable to interpret the category as: "a symptom of a medical condition called insulin deficiency which is about the deficiency of one type of hormone called insulin." In this way, our knowledge about a category can go beyond a simple string and its member entities. An immediate application of templates is removing invalid category names from a noisy category collection. Promising results are observed for this application in our experiments.

An intuitive approach to this task (i.e., extracting templates from a collection of category names)

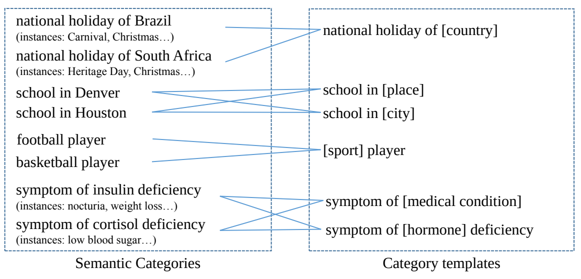
national holiday of Brazil
(instances: Carnival, Christmas…)
national holiday of South Africa
(instances: Heritage Day, Christmas…)
school in Denver
school in Houston
football player
basketball player
symptom of insulin deficiency
(instances: nocturia, weight loss…)
symptom of cortisol deficiency
(instances: low blood sugar…)
Semantic Categories
national holiday of [country]
school in [place]
school in [city]
[sport] player
symptom of [medical condition]
symptom of [hormone] deficiency
Category templates

Figure 1: Examples of semantic categories and their corresponding templates.

contains two stages: category labeling, and template scoring.

Category labeling: Divide a category name into multiple segments and replace some key segments with its hypernyms. As an example, assume “CEO of Delphinus” is divided to three segments “CEO + of + Delphinus”; and the last segment (Delphinus) has hypernyms “constellation”, “company”, etc. By replacing this segment with its hypernyms, we get candidate templates “CEO of [constellation]” (a wrong template), “CEO of [company]”, and the like.

Template scoring: Compute the score of each candidate template by aggregating the information obtained in the first phase.

A major challenge here is that many segments (like “Delphinus” in the above example) have multiple meanings. As a result, wrong hypernyms may be adopted to generate incorrect candidate templates (like “CEO of [constellation]”). In this paper, we focus on improving the template scoring stage, with the goal of assigning lower scores to bad templates and larger scores to high-quality ones.

There have been some research efforts (Third, 2012; Fernandez-Breis et al., 2010; Quesada-Martınez et al., 2012) on exploring the structure of category names by building patterns. However, we automatically assign semantic types to the pattern variables (or called arguments) while they do not. For example, our template has the form of “city in [country]” while their patterns are like “city in [X]”. More details are given in the related work section.

A similar task is query understanding, including query tagging and query template mining. Query tagging (Li et al., 2009; Reisinger and Pasca, 2011) corresponds to the category labeling stage described above. It is different from template generation because the results are for one query only, without merging the information of all queries to generate the final templates. Category template construction are slightly different from query template construction. First, some useful features such as query click-through is not available in category template construction. Second, categories should be valid natural language phrases, while queries need not. For example, “city Germany” is a query but not a valid category name. We discuss in more details in the related work section.

Our major contributions are as follows.

1) To the best of our knowledge, this is the first work of template generation specifically for categories in unsupervised manner.

2) We extract semantic knowledge and statistical information from a web corpus for improving template generation. Significant performance improvement is obtained in our experiments.

3) We study the characteristics of the scoring function from the viewpoint of probabilistic evidence combination and demonstrate that nonlinear functions are more effective in this task.

4) We employ the output templates to clean our category collection mined from the web, and get apparent quality improvement (precision improved from 81% to 89%).

After discussing related work in Section 2, we define the problem and describe one baseline approach in Section 3. Then we introduce our approach in Section 4. Experimental results are reported and analyzed in Section 5. We conclude the paper in Section 6.

2 Related work

Several kinds of work are related to ours.

Hypernymy relation extraction: Hypernymy relation extraction is an important task in text mining. There have been a lot of efforts (Hearst, 1992; Pantel and Ravichandran, 2004; Van Durme and Pasca, 2008; Zhang et al., 2011) in the literature to extract hypernymy (or is-a) relations from the web. Our target here is not hypernymy extraction, but discovering the semantic structure of hypernyms (or category names).

Category name exploration: Category name patterns are explored and built in some existing research work. Third (2012) proposed to find axiom patterns among category names on an existing ontology. For example, infer axiom pattern “SubClassOf(AB, B)” from “SubClassOf(junior_school school)” and “SubClassOf(domestic_mammal mammal)”. Fernandez-Breis et al. (2010) and Quesada-Martınez et al. (2012) proposed to find lexical patterns in category names to define axioms (in medical domain). One example pattern mentioned in their papers is “[X] binding”. They need manual intervention to determine what X means. The main difference between the above work and ours is that we automatically assign semantic types to the pattern variables (or called arguments) while they do not.

Template mining for IE: Some research work in information extraction (IE) involves patterns. Yangarber (2003) and Stevenson and Greenwood (2005) proposed to learn patterns which were in the form of [subject, verb, object]. The category names and learned templates in our work are not in this form. Another difference between our work and their work is that, their methods need a supervised name classifer to generate the candidate patterns while our approach is unsupervised. Chambers and Jurafsky (2011) leverage templates to describe an event while the templates in our work are for understanding category names (a kind of short text).

Query tagging/labeling: Some research work in recent years focuses on segmenting web search queries and assigning semantic tags to key segments. Li et al. (2009) and Li (2010) employed CRF (Conditional Random Field) or semi-CRF models for query tagging. A crowdsourcing-assisted method was proposed by Han et al. (2013) for query structure interpretation. These supervised or semi-supervised approaches require much manual annotation effort. Unsupervised methods were proposed by Sarkas et al. (2010) and Reisinger and Pasca (2011). As been discussed in the introduction section, query tagging is only one of the two stages of template generation. The tagging results are for one query only, without aggregating the global information of all queries to generate the final templates.

Query template construction: Some existing work leveraged query templates or patterns for query understanding. A semi-supervised random walk based method was proposed by Agarwal et al. (2010) to generate a ranked templates list which are relevant to a domain of interest. A predefined domain schema and seed information is needed for this method. Pandey and Punera (2012) proposed an unsupervised method based on graphical models to mine query templates. The above methods are either domain-specific (i.e., generating templates for a specific domain), or have some degree of supervision (supervised or semi-supervised). Cheung and Li (2012) proposed an unsupervised method to generate query templates by the aid of knowledge bases. An approach was proposed in (Szpektor et al., 2011) to improve query recommendation via query templates. Query session information (which is not available in our task) is needed in this approach for templates generation. Li et al. (2013) proposed an clustering algorithm to group existing query templates by search intents of users.

Compared to the open-domain unsupervised methods for query template construction, our approach improves on two aspects. First, we propose to incorporate multiple types of semantic knowledge (e.g., term peer similarity and term clusters) to improve template generation. Second, we propose a nonlinear template scoring function which is demonstrated to be more effective.

3 Problem Definition and Analysis

3.1 Problem definition

The goal of this paper is to construct a list of category templates from a collection of open-domain category names.

Input: The input is a collection of category names, which can either be manually compiled (like Wikipedia categories) or be automatically extracted. The categories used in our experiments were automatically mined from the web, by following existing work (Hearst, 1992, Pantel and Ravichandran 2004; Snow et al., 2005; Talukdar et al., 2008; Zhang et al., 2011). Specifically, we applied Hearst patterns (e.g., "NP [,] (such as | including) $\{NP,\}^*$ {and|or} NP") and is-a patterns ("NP (is|are|was|were|being) (a|an|the) NP") to a large corpus containing 3 billion English web pages. As a result, we obtained a term→hypernym bi-partite graph containing 40 million terms, 74 million hypernyms (i.e., category names), and 321 million edges (e.g., one example edge is "Berlin"→"city in Germany", where "Berlin" is a term and "city in Germany" is the corresponding hypernym). Then all the multi-word hypernyms are used as the input category collection.

Output: The output is a list of templates, each having a score indicating how likely it is valid. A template is a multi-word string with one headword and at least one argument. For example, in template "national holiday of [country]", "holiday" is the headword, and "[country]" is the argument. We only consider one-argument templates in this paper, and the case of multiple arguments is left as future work. A template is valid if it is syntactically and semantically correct. "CEO of [constellation]" (wrongly generated from "CEO of Delphinus", "CEO of Aquila", etc.) is not valid because it is semantically unreasonable.

3.2 Baseline approach

An intuitive approach to this task contains two stages: category labeling and template scoring. Figure 2 shows its workflow with simple examples.

3.2.1 Phase-1: Category labeling

At this stage, each category name is automatically segmented and labeled; and some *candidate template tuples* (CTTs) are derived based on the labeling results. This can be done in the following steps.

Category segmentation: Divide each category name into multiple segments (e.g., "holiday of South Africa" to "holiday + of + South Africa"). Each segment is one word or a phrase appearing in an entity dictionary. The dictionary used in this paper is comprised of all Freebase (www.freebase.com) entities.

Segment to hypernym: Find hypernyms for every segment (except for the headword and some trivial segments like prepositions and articles), by referring to a term→hypernym mapping graph. Following most existing query labeling work, we derive the term→hypernym graph from a dump of Freebase. Below are some examples of Freebase types (hypernyms),

German city (id: /location/de_city)
Italian province (id: /location/it_province)
Poem character (id: /book/poem_character)
Book (id: /book/book)

To avoid generating too fine-grained templates like "mayor of [Germany city]" and "mayor of [Italian city]" (semantically "mayor of [city]" is more desirable), we discard type modifiers and map terms to the headwords of Freebase types. For example, "Berlin" is mapped to "city". In this way, we build our basic version of term→hypernym mapping which contains 16.13 million terms and 696 hypernyms. Since "South Africa" is both a country and a book name in Freebase, hypernyms "country", "book", and others are assigned to the segment "South Africa" in this step.

CTT generation: Construct CTTs by choosing one segment (called the *target segment*) each time and replacing the segment with its hypernyms. An CTT is formed by the candidate template (with one argument), the target segment (as an *argument value*), and the tuple score (indicating tuple quality). Below are example CTTs obtained after the last segment of "holiday + of + South Africa" is processed,

U_1: (holiday of [country], South Africa, w_1)
U_2: (holiday of [book], South Africa, w_2)

3.2.2 Phase-2: Template scoring

The main objective of this stage is to merge all the CTTs obtained from the previous stage and to compute a final score for each template. In this stage, the CTTs are first grouped by the first element (i.e., the template string). For example, tuples for "holiday of [country]" may include,

U_1: (holiday of [country], South Africa, w_1)
U_2: (holiday of [country], Brazil, w_2)
U_3: (holiday of [country], Germany, w_3)
...

Then a scoring function is employed to calculate the template score from the tuple scores. Formally, given n tuples $\vec{U}$=(U_1, U_2..., U_n) for a template, the goal is to find a score fusion function $F(\vec{U})$ which yields large values for high-quality templates and small (or zero) values for invalid ones.

Borrowing the idea of TF-IDF from information retrieval, a reasonable scoring function is,

$$F(\vec{U}) = \sum_{i=1}^{n} w_i \cdot IDF(h) \tag{1}$$

where h is the argument type (i.e., the hypernym of the argument value) of each tuple. TF means the "term frequency" and IDF means the "inverse document frequency". An IDF function assigns lower scores to common hypernyms (like person and music track which contain a lot of entities). Let $DF(h)$ be the number of entities having hypernym h, we test two IDF functions in our experiments,

$$\begin{aligned} IDF_1(h) &= log\frac{1+N}{1+DF(h)} \\ IDF_2(h) &= 1/sqrt(DF(h)) \end{aligned} \tag{2}$$

where N is total number of entities in the entity dictionary.

The next problem is estimating tuple score w_i. Please note that there is no weight or score information in the term→hypernym mapping of Freebase. So we have to set w_i to be constant in the baseline,

$$w_i = 1 \tag{3}$$

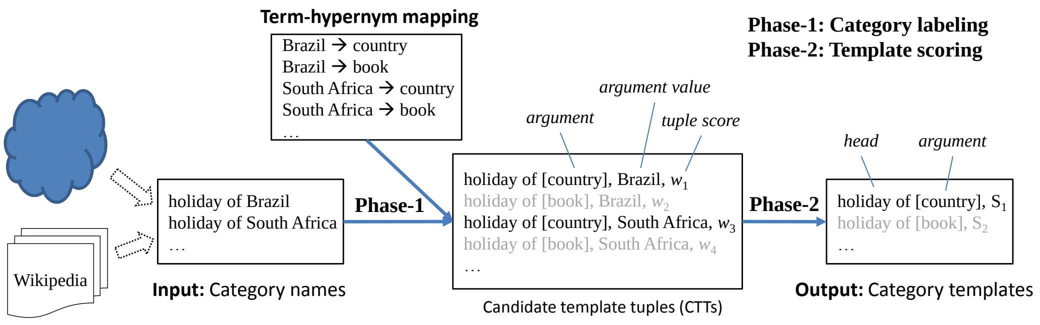

Figure 2: Problem definition and baseline approach.

4 Approach: Enhancing Template Scoring

In our approach, we follow the same framework as in the above baseline approach, and focus on improving the template scoring phase (i.e., phase-2).

We try three techniques: First, a better tuple score w_i is calculated in Section 4.1 by performing statistics on a large corpus. The corpus is a collection of 3 billion web pages crawled in early 2013 by ourselves. During this paper, we use "our web corpus" or "our corpus" to refer to this corpus.

Second, a nonlinear function is adopted in Section 4.2 to replace the baseline tuple fusion function (Formula 1). Third, we extract term peer similarity and term clusters from our corpus and use them as additional semantic knowledge to refine template scores.

4.1 Enhancing tuple scoring

Let's examine the following two template tuples,

U_1: (holiday of [country], South Africa, w_1)

U_2: (holiday of [book], South Africa, w_2)

Intuitively, "South Africa" is more likely to be a country than a book when it appears in text. So for a reasonable tuple scoring formula, we should have $w_1 > w_2$.

The main idea is to automatically calculate the popularity of a hypernym given a term, by referring to a large corpus. Then by adding the popularity information to (the edges of) the term→hypernym graph of Freebase, we obtain a *weighted* term→hypernym graph. The weighted graph is then employed to enhance the estimation of w_i.

For popularity calculation, we apply Hearst patterns (Hearst, 1992) and is-a patterns ("NP (is|are|was|were|being) (a|an|the) NP") to every sentence of our web corpus. For a (term, hypernym) pair, its popularity F is calculated as the number of sentences in which the term and the hypernym co-occur and also follow at least one of the patterns.

For a template tuple U_i with argument type h and argument value v, we test two ways of estimating the tuple score w_i,

$$w_i = \log\left(1 + F(v, h)\right) \tag{4}$$

$$w_i = \frac{F(v, h))}{\lambda + \sum_{h_j \in H} F(v, h_j)} \tag{5}$$

where $F(v, h)$ is the popularity of the (v, h) pair in our corpus, H is the set of all hypernyms for v in the weighted term→hypernym graph. Parameter λ (=1.0 in our experiments) is introduced for smoothing purpose. Note that the second formula is the conditional probability of hypernym h given term v.

Since it is intuitive to estimate tuple scores with their frequencies in a corpus, we treat the approach with the improved w_i as another baseline (our strong baseline).

4.2 Enhancing tuple combination function

Now we study the possibility of improving the tuple combination function (Formula 1), by examining the tuple fusion problem from the viewpoint of probabilistic evidence combination. We first demonstrate that the linear function in Formula 1 corresponds to the conditional independence assumption of the tuples. Then we propose to adopt a series of nonlinear functions for combining tuple scores.

We define the following events:

T: Template T is a valid template;

$\overline{T}$: T is an invalid template;

E_i: The observation of tuple U_i.

Let's compute the posterior odds of event T, given two tuples U_1 and U_2. Assuming E_1 and E_2 are conditionally independent given T or $\overline{T}$, according to the Bayes rule, we have,

$$\begin{aligned}\frac{P(T|E_1,E_2)}{P(\overline{T}|E_1,E_2)} &= \frac{P(E_1,E_2|T)\cdot P(T)}{P(E_1,E_2|\overline{T})\cdot P(\overline{T})}\\ &= \frac{P(E_1|T)}{P(E_1|\overline{T})}\cdot\frac{P(E_2|T)}{P(E_2|\overline{T})}\cdot\frac{P(T)}{P(\overline{T})}\\ = \frac{P(T|E_1)\cdot P(\overline{T})}{P(\overline{T}|E_1)\cdot P(T)}&\cdot\frac{P(T|E_2)\cdot P(\overline{T})}{P(\overline{T}|E_2)\cdot P(T)}\cdot\frac{P(T)}{P(\overline{T})}\end{aligned} \tag{6}$$

Define the log-odds-gain of T given E as,

$$G(T|E) = \log\frac{P(T|E)}{P(\overline{T}|E)} - \log\frac{P(T)}{P(\overline{T})} \tag{7}$$

Here G means the gain of the log-odds of T after E occurs. By combining formulas 6 and 7, we get

$$G(T|E_1,E_2) = G(T|E_1) + G(T|E_2) \tag{8}$$

It is easy to prove that the above conclusion holds true when $n > 2$, i.e.,

$$G(T|E_1,...,E_n) = \sum_{i=1}^{n} G(T|E_i) \tag{9}$$

If we treat $G(T|E_i)$ as the score of template T when only U_i is observed, and $G(T|E_1,...,E_n)$ as the template score after the n tuples are observed, then the above equation means that the combined template score should be the sum of $w_i \cdot IDF(h)$, which is exactly Formula 1. Please keep in mind that Equation 9 is based on the assumption that the tuples are conditional independent. This assumption, however, may not hold in reality. The case of conditional dependence was studied in (Zhang et al., 2011), where a group of nonlinear combination functions were proposed and achieved good performance in their task of hypernymy extraction. We choose p-Norm as our nonlinear fusion functions, as below,

$$F(\vec{U}) = \sqrt[p]{\sum_{i=1}^{n} w_i^p \cdot IDF(h)} \qquad (p > 1) \tag{10}$$

where p (=2 in experiments) is a parameter.

Experiments show that the above nonlinear function performs better than the linear function of Formula 1. Let's use an example to show the intuition. Consider a good template "city of [country]" corresponding to CTTs $\vec{U}_A$ and a wrong template "city of [book]" having tuples $\vec{U}_B$. Suppose $|\vec{U}_A| = 200$ (including most countries in the world) and $|\vec{U}_B| = 1000$ (considering that many place names have already been used as book names). We observe that each tuple score corresponding to "city of [country]" is larger than the tuple score corresponding to "city of [book]". For simplicity, we assume each tuple in $\vec{U}_A$ has score 1.0 and each tuple in $\vec{U}_B$ has score 0.2. With the linear and nonlinear (p=2) fusion functions, we can get,

Linear:

$$\begin{aligned}F(\vec{U}_A) &= 200 * 1.0 = 200\\ F(\vec{U}_B) &= 1000 * 0.2 = 200\end{aligned} \tag{11}$$

Nonlinear:

$$\begin{aligned}F(\vec{U}_A) &= 14.1\\ F(\vec{U}_B) &= 6.32\end{aligned} \tag{12}$$

In the above settings the nonlinear function yields a much higher score for the good template (than for the invalid template), while the linear one does not.

4.3 Refinement with term similarity and term clusters

The above techniques neglect the similarity among terms, which has a high potential to improve the template scoring process. Intuitively, for a toy set {"city in Brazil", "city in South Africa","city in China", "city in Japan"}, since "Brazil", "South Africa", "China" and "Japan" are very similar to each other and they all have a large probability to be a "country", so we have more confidence that "city in [country]" is a good template. In this section, we propose to leverage the term similarity information to improve the template scoring process.

We start with building a large group of *small* and *overlapped* clusters from our web corpus.

4.3.1 Building term clusters

Term clusters are built in three steps.

Mining term peer similarity: Two terms are peers if they share a common hypernym and they are semantically correlated. For example, "dog" and "cat" should have a high peer similarity score. Following existing work (Hearst, 1992; Kozareva et al., 2008; Shi et al., 2010; Agirre et al., 2009; Pantel et al., 2009), we built a peer similarity graph containing about 40.5 million nodes and 1.33 billion edges.

Clustering: For each term, choose its top-30 neighbors from the peer similarity graph and run a hierarchical clustering algorithm, resulting in one or multiple clusters. Then we merge highly duplicated clusters. The algorithm is similar to the first part of CBC (Pantel and Lin, 2002), with the difference that a very high merging threshold is adopted here in order to generate small and overlapped clusters. Please note that one term may be included in many clusters.

Assigning top hypernyms: Up to two hypernyms are assigned for each term cluster by majority voting of its member terms, with the aid of the weighted term→hypernym graph of Section 4.1. To be an eligible hypernym for the cluster, it has to be the hypernym of at least 70% of terms in the cluster. The score of each hypernym is the average of the term→hypernym weights over all the member terms.

4.3.2 Template score refinement

With term clusters at hand, now we describe the score refinement procedure for a template T having argument type h and supporting tuples $\vec{U}$=(U_1, U_2..., U_n). Denote $V = \{V_1, V_2, ..., V_n\}$ to be the set of argument values for the tuples (where V_i is the argument value of U_i).

By computing the intersection of V and every term cluster, we can get a distribution of the argument values in the clusters. We find that for a good template like "holiday in [country]", we can often find at least one cluster (one of the country clusters in this example) which has hypernym h and also contains many elements in V. However, for invalid templates like "holiday of [book]", every cluster having hypernym h (="book" here) only contains a few elements in V. Inspired by such an observation, our score refinement algorithm for template T is as follows,

Step-1. Calculating supporting scores: For each term cluster C having hypernym h, compute its supporting score to T as follows:

$$S(C,T) = k(C,V) \cdot w(C,h) \qquad (13)$$

where $k(C,V)$ is the number of elements shared by C and V, and $w(C,h)$ is hypernym score of h to C (computed in the last step of building clusters).

Step-2. Calculating the final template score: Let term cluster C^* has the maximal supporting score to T, the final template score is computed as,

$$S(T) = F(\vec{U}) \cdot S(C^*,T) \qquad (14)$$

where $F(\vec{U})$ is the template score before refinement.

5 Experiments

5.1 Experimental setup

5.1.1 Methods for comparison

We make a comparison among 10 methods.

SC: The method is proposed in (Cheung and Li, 2012) to construct templates from queries. The method firstly represents a query as a matrix based on Freebase data. Then a hierarchical clustering algorithm is employed to group queries having the same structure and meaning. Then an intent summarization algorithm is employed to create templates for each query group.

Base: The linear function in Formula 1 is adopted to combine the tuple scores. We use IDF_2 here because it achieves higher precision than IDF_1 in this setting.

LW: The linear function in Formula 1 is adopted to combine the tuple scores generated by Formula 4. IDF_1 is used rather than IDF_2 for better performance.

LP: The linear function in Formula 1 is adopted to combine the tuple scores generated by Formula 5. IDF_2 is used rather than IDF_1 for better performance.

NLW: The nonlinear fusion function in Formula 10 is used. Other settings are the same as LW.

NLP: The nonlinear fusion function in Formula 10 is used. Other settings are the same as LP.

LW+C, LP+C, NLW+C, NLP+C: All the settings of LW, LP, NLW, NLP respectively, with the refinement technology in Section 4.3 applied.

5.1.2 Data sets, annotation and evaluation metrics

The input category names for experiments are automatically extracted from a web corpus (Section 3.1). Two test-sets are built for evaluation from the output templates of various methods.

Subsets: In order to conveniently compare the performance of different methods, we create 20 sub-collections (called subsets) from the whole input category collection. Each subset contains all the categories having the same headword (e.g., "symptom of insulin deficiency" and "depression symptom" are in the same subset because they share the same headword "symptom"). To choose the 20 headwords, we first sample 100 at random from the set of all headwords; then manually choose 20 for diversity. The headwords include symptom, school, food, gem, hero, weapon, model, etc. We run the 10 methods on these subsets and sort the output templates by their scores. Top-30 templates from each method on each subset are selected and mixed together for annotation.

Fullset: We run method NLP+C (which has the best performance according to our subsets experiments) on the input categories and sort the output templates by their scores. Then we split the templates into 9 sections according to their ranking position. The sections are: [1∼100], (100∼1K], (1K∼10K], (10K∼100K], (100K,120K], (120K∼140K], (140K∼160K], (160K∼180K], (180K∼200K]. Then 40 templates are randomly chosen from each section and mixed together for annotation.

The selected templates (from subsets and the fullset) are annotated by six annotators, with each template assigned to two annotators. A template is assigned a label of "good", "fair", or "bad" by an annotator. The percentage agreement between the annotators is 80.2%, with kappa 0.624.

For the subset experiments, we adopt Precision@k (k=10,20,30) to evaluate the top templates generated by each method. The scores for "good", "fair", and "bad" are 1, 0.5, and 0. The score of each template is the average annotation score over two annotators (e.g., if a template is annotated "good" by one annotator and "fair" by another, its score is (1.0+0.5)/2=0.75). The evaluation score of a method is the average over the 20 subsets. For the fullset experiments, we report the precision for each section.

5.2 Experimental results

5.2.1 Results for subsets

The results of each method on the 20 subsets are presented in Table 1. A few observations can be made. First, by comparing the performance of baseline-1 (Base) and the methods adopting term→hypernym weight (LW and LP), we can see big performance improvement. The bad performance of baseline-1 is mainly due to the lack of weight (or frequency) information on

Method		**P@10**	**P@20**	**P@30**
Base (baseline-1)		0.359	0.361	0.358
SC (Cheung and Li, 2012)		0.382	0.366	0.371
Weighted (baseline-2)	LW	0.633	0.582	0.559
	LP	0.771	0.734	0.707
Nonlinear	NLW	0.711	0.671	0.638
	NLP	0.818	0.791	0.765
Term cluster	LW+C	0.813	0.786	0.754
	NLW+C	0.854	0.833	0.808
	LP+C	0.818	0.788	0.778
	NLP+C	0.868	0.839	0.788

Table 1: Performance comparison among the methods on subset.

term→hypernym edges. The results demonstrate that edge scores are critical for generating high quality templates. Manually built semantic resources typically lack such kinds of scores. Therefore, it is very important to enhance them by deriving statistical data from a large corpus. Since it is relatively easy to have the idea of adopting a weighted term→hypernym graph, we treat LW and LP as another (stronger) baseline named baseline-2.

As the second observation, the results show that the nonlinear methods (NLP and NLW) achieve performance improvement over their linear versions (LW and LP).

Third, let's examine the methods with template scores refined by term similarity and term clusters (LW+C, NLW+C, LP+C, NLP+C). It is shown that the refine-by-cluster technology brings additional performance gains on all the four settings (linear and nonlinear, two different ways of calculating tuple scores). So we can conclude that the peer similarity and term clusters are quite effective in improving template generation.

Fourth, the best performance is achieved when the three techniques (i.e., term→hypernym weight, nonlinear fusion function, and refine-by-cluster) are combined together. For instance, by comparing the P@20 scores of baseline-2 and NLP+C, we see a performance improvement of 14.3% (from 0.734 to 0.839). Therefore every technique studied in this paper has its own merit in template generation.

Finally, by comparing the method SC (Cheung and Li, 2012) with other methods, we can see that SC is slightly better than baseline-1, but has much lower performance than others. The major reason may be that this method did not employ a weighted term→hypernym graph or term peer similarity information in template construction.

		Base	**SC**	**LP**	**NLP**	**LP+C**
	SC	$\sim$				
	LP	$> **$	$> **$			
P@10	**NLP**	$> **$	$> **$	$>$		
	LP+C	$> **$	$> **$	$> **$	$\sim$	
	NLP+C	$> **$	$> **$	$> **$	$> **$	$>$
		Base	**SC**	**LP**	**NLP**	**LP+C**
	SC	$\sim$				
	LP	$> **$	$> **$			
P@20	**NLP**	$> **$	$> **$	$> **$		
	LP+C	$> **$	$> **$	$> **$	$\sim$	
	NLP+C	$> **$	$> **$	$> **$	$> **$	$> **$
		Base	**SC**	**LP**	**NLP**	**LP+C**
	SC	$\sim$				
	LP	$> **$	$> **$			
P@30	**NLP**	$> **$	$> **$	$> **$		
	LP+C	$> **$	$> **$	$> **$	$\sim$	
	NLP+C	$> **$	$> **$	$> **$	$>$	$\sim$

Table 2: Paired t-test results on subsets.

		Base	**SC**	**LW**	**NLW**	**LW+C**
	SC	$\sim$				
	LW	$> **$	$> **$			
P@10	**NLW**	$> **$	$> **$	$> *$		
	LW+C	$> **$	$> **$	$> **$	$> **$	
	NLW+C	$> **$	$> **$	$> **$	$> **$	$> *$
		Base	**SC**	**LW**	**NLW**	**LW+C**
	SC	$\sim$				
	LW	$> **$	$> **$			
P@20	**NLW**	$> **$	$> **$	$> **$		
	LW+C	$> **$	$> **$	$> **$	$> **$	
	NLW+C	$> **$	$> **$	$> **$	$> **$	$> **$
		Base	**SC**	**LW**	**NLW**	**LW+C**
	SC	$\sim$				
	LW	$> **$	$> **$			
P@30	**NLW**	$> **$	$> **$	$> **$		
	LW+C	$> **$	$> **$	$> **$	$> **$	
	NLW+C	$> **$	$> **$	$> **$	$> **$	$> **$

Table 3: Paired t-test results on subsets.

Are the performance differences between methods significant enough for us to say that one is better than the other? To answer this question, we run paired two-tailed t-test on every pair of methods. We report the t-test values among methods in tables 2, 3 and 4.

The meaning of the symbols in the tables are,

$\sim$: The method on the row and the one on the column have similar performance.

$>$: The method on the row outperforms the method on the column, but the performance difference is not statistically significant ($0.05 \leq P < 0.1$ in two-tailed t-test).

$> *$: The performance difference is statistically significant ($P < 0.05$ in two-tailed t-test).

$> **$: The performance difference is statistically highly significant ($P < 0.01$ in two-tailed t-test).

	P@10	**P@20**	**P@30**
LP V.S. **LW**	$> **$	$> **$	$> **$
NLP V.S. **NLW**	$> **$	$> **$	$> **$
LP+C V.S. **LW+C**	$\sim$	$\sim$	$\sim$
NLP+C V.S. **NLW+C**	$\sim$	$\sim$	$\sim$

Table 4: Paired t-test results on subsets.

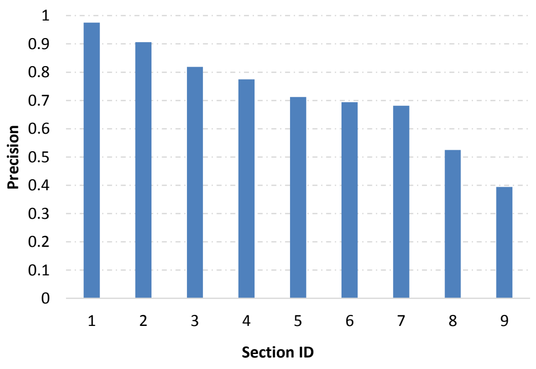

Figure 3: Precision by section in the fullset.

5.2.2 Fullset results

As described in the Section 5.1.2, for the *fullset* experiments, we conduct a section-wise evaluation, selecting 40 templates from each of the 9 sections of the NLP+C results. The results are shown in Figure 3. It can be observed that the precision for each section decreases when the section ID increases. The results indicate the effectiveness of our approach, since it can rank good templates in top sections and bad templates in bottom sections. According to the section-wise precision data, we are able to determine the template score threshold for choosing different numbers of top templates in different applications.

5.2.3 Templates for category collection cleaning

Since our input category collection is automatically constructed from the web, some wrong or invalid category names is inevitably contained. In this subsection, we apply our category templates to clean the category collection. The basic idea is that if a category can match a template, it is more likely to be correct. We compute a new score for every category name H as follows,

$$S_{new}(H) = \log(1 + S(H)) \cdot S(T^*) \qquad (15)$$

where $S(H)$ is the existing category score, determined by its frequency in the corpus. Here $S(T^*)$ is the score of template T^*, the best template (i.e., the template with the highest score) for the category.

Then we re-rank the categories according to their new scores to get a re-ranked category list. We randomly sampled 150 category names from the top 2 million categories of each list (the old list and the new list) and asked annotators to judge the quality of the categories. The annotation results show that, after re-ranking, the precision increases from 0.81 to 0.89 (i.e., the percent of invalid category names decreases from 19% to 11%).

6 Conclusion

In this paper, we studied the problem of building templates for a large collection of category names. We tested three techniques (tuple scoring by weighted term→hypernym mapping, non-linear score fusion, refinement by term clusters) and found that all of them are very effective and their combination achieves the best performance. By employing the output templates to clean our category collection mined from the web, we get apparent quality improvement. Future work includes supporting multi-argument templates, disambiguating headwords of category names and applying our approach to general short text template mining.

Acknowledgments

We would like to thank the annotators for their efforts in annotating the templates. Thanks to the anonymous reviewers for their helpful comments and suggestions. This work is supported in part by China National 973 program 2014CB340301 and NSFC grant 61379043.

References

Ganesh Agarwal, Govind Kabra, and Kevin Chen-Chuan Chang. 2010. Towards rich query interpretation: walking back and forth for mining query templates. In *Proceedings of the 19th international conference on World wide web*, pages 1–10. ACM.

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics.

Nathanael Chambers and Dan Jurafsky. 2011. Template-based information extraction without the templates. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 976–986. Association for Computational Linguistics.

Jackie Chi Kit Cheung and Xiao Li. 2012. Sequence clustering and labeling for unsupervised query intent discovery. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 383–392. ACM.

Jesualdo Tomas Fernandez-Breis, Luigi Iannone, Ignazio Palmisano, Alan L Rector, and Robert Stevens. 2010. Enriching the gene ontology via the dissection of labels using the ontology pre-processor language. In *Knowledge Engineering and Management by the Masses*, pages 59–73. Springer.

Jun Han, Ju Fan, and Lizhu Zhou. 2013. Crowdsourcing-assisted query structure interpretation. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2092–2098. AAAI Press.

Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zornitsa Kozareva, Ellen Riloff, and Eduard H Hovy. 2008. Semantic class learning from the web with hyponym pattern linkage graphs. In *ACL*, volume 8, pages 1048–1056.

Xiao Li, Ye-Yi Wang, and Alex Acero. 2009. Extracting structured information from user queries with semi-supervised conditional random fields. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 572–579. ACM.

Yanen Li, Bo-June Paul Hsu, and ChengXiang Zhai. 2013. Unsupervised identification of synonymous query intent templates for attribute intents. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2029–2038. ACM.

Xiao Li. 2010. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1337–1345. Association for Computational Linguistics.

Sandeep Pandey and Kunal Punera. 2012. Unsupervised extraction of template structure in web search queries. In *Proceedings of the 21st international conference on World Wide Web*, pages 409–418. ACM.

Patrick Pantel and Dekang Lin. 2002. Discovering word senses from text. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 613–619. ACM.

Patrick Pantel and Deepak Ravichandran. 2004. Automatically labeling semantic classes. In *HLT-NAACL*, volume 4, pages 321–328.

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. 2009. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 938–947. Association for Computational Linguistics.

Manuel Quesada-Martınez, Jesualdo Tomás Fernández-Breis, and Robert Stevens. 2012. Enrichment of owl ontologies: a method for defining axioms from labels. In *Proceedings of the First International Workshop on Capturing and Refining Knowledge in the Medical Domain (K-MED 2012), Galway, Ireland*, pages 1–10.

Joseph Reisinger and Marius Pasca. 2011. Fine-grained class label markup of search queries. In *ACL*, pages 1200–1209.

Nikos Sarkas, Stelios Paparizos, and Panayiotis Tsaparas. 2010. Structured annotations of web queries. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 771–782. ACM.

Shuming Shi, Huibin Zhang, Xiaojie Yuan, and Ji-Rong Wen. 2010. Corpus-based semantic class mining: distributional vs. pattern-based approaches. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 993–1001. Association for Computational Linguistics.

Mark Stevenson and Mark A Greenwood. 2005. A semantic approach to ie pattern induction. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 379–386. Association for Computational Linguistics.

Idan Szpektor, Aristides Gionis, and Yoelle Maarek. 2011. Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th international conference on World wide web*, pages 47–56. ACM.

Allan Third. 2012. Hidden semantics: what can we learn from the names in an ontology? In *Proceedings of the Seventh International Natural Language Generation Conference*, pages 67–75. Association for Computational Linguistics.

Benjamin Van Durme and Marius Pasca. 2008. Finding cars, goddesses and enzymes: Parametrizable acquisition of labeled instances for open-domain information extraction. In *AAAI*, volume 8, pages 1243–1248.

Roman Yangarber. 2003. Counter-training in discovery of semantic patterns. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 343–350. Association for Computational Linguistics.

Fan Zhang, Shuming Shi, Jing Liu, Shuqi Sun, and Chin-Yew Lin. 2011. Nonlinear evidence fusion and propagation for hyponymy relation mining. In *ACL*, volume 11, pages 1159–1168.