Using Instance Texts to Improve Keyword-based Class Retrieval

Xingjian Zhang
Department of Computer Science and Engineering
Lehigh University, Bethlehem, PA 18015
Email: xiz307@lehigh.edu

Jeff Heflin
Department of Computer Science and Engineering
Lehigh University, Bethlehem, PA 18015
Email: heflin@cse.lehigh.edu

***Abstract*—In this paper we investigate the keyword based class retrieval problem, which we define as how to identify ontological classes that best match a keyword based query. Most previous applications use simple syntactic matching approaches on the class labels and/or comments, or expand the keyword query by using lexicons such as WordNet, but fail to retrieve relevant resources in many scenarios. Instead of relying on external sources, we investigate this problem by using the annotations of instances associated with classes in the knowledge base. We propose a general framework of this approach, which consists of two phases: the keyword query is first used to locate relevant instances; then we induce the classes given this list of weighted matched instances. If we identify sufficient text for the instances, then the first phase can be solved by a traditional information retrieval (IR) query, however the second phase might be cast in different ways: as an additive value function, as an IR problem with instance as queries, or as an instance-based ontology alignment problem. With many applicable strategies initiated from different viewpoints, we find that some of them are mathematically equivalent or very similar. In the experiments we compare our proposed framework to simple syntactic approaches and evaluate different strategies.**

I. Introduction

Ontologies and Knowledge Bases (KBs) have become widely used and are increasingly populated with real world data. Particularly, Semantic Web techniques have been applied to build KBs in many applications. While the ontology or schema provides useful structuring of the data and often includes semantics that can improve query answering, it also serves as a barrier to casual users who may not know what ontological terms to use or how they should be connected in a KB. Thus an important problem in using KBs is how to translate natural language queries to the appropriate ontological terms. In this paper we address the **resource retrieval** problem, which is the task to find the best matched resources (classes, properties, or instance) in the KB given a keyword-based query. Resource retrieval can be used in scenarios such as: (1) a portal that helps users locate a KB of user's interests; (2) a keyword search bar to specify a resource when a user explores a KB; and (3) a Question Answering (QA) system that needs to translate natural language queries into resources in the KB and later links these resources to form a valid query (e.g. in SPARQL).

In most of these scenarios, existing tools typically use simple string matching, although some expand the matching by using lexicons like WordNet. We believe that leveraging usage information from the KB can improve retrieval quality better than referencing external lexicons. Our intuition comes from the observation that in many scenarios, humans learn what a named class refers to by looking into some of its instances. For example, when we see a class named "Person", if after examining several random instances of it we find out all of them are scientists, we have an idea that this class *Person* may mainly refer to researchers. Now consider another example: a class named "Cat". If the instances include species of tigers, leopards, etc. then we know that it refers to felines in general, and not just the typical house cat. Similarly, a resource retrieval component can also obtain more information about a class by identifying patterns in the textual properties of an instance, and using this information to improve retrieval quality. Now consider the last example with an improved retrieval component. When users query "tiger", although there is no class named "Tiger", the retrieval component knows the class *Cat* covers the query topic best.

In this paper, we focus on the problem of **class retrieval** using instance texts. One of the interesting things about the problem is that it can be viewed from many different perspectives: we can solve it with an additive value function that combines the matched instances; we can rewrite the class retrieval query as an information retrieval (IR) query over the matched instances; or we can apply approaches from the instance-based ontology alignment problem. Our contributions include: (1) a proposal of a two-phase framework that solves the class retrieval problem using texts from instances; and (2) modeling the problem from different viewpoints and comparing these approaches with regard to the class retrieval problem. The paper is organized as follows. In Section II, we briefly introduce background knowledge related to our work. In Section III, we propose our two-phase framework and define instance texts. In Section IV, we discuss the problem from the three aspects. In Section V, we show experimental results for comparison and lastly in Section VI, we conclude.

II. Background

Although many systems perform some form of resource retrieval, to the best of our knowledge, none of them use instances to improve the retrieval of classes. For example, Sindice [1] is a state-of-the-art Semantic Web search engine that has an inverted index over crawled resources, and allows users to retrieve documents with statements about particular resources. Inverse functional property values (e.g. email) are indexed for instances retrieval. However, if there are many similar matching instances, it is up to the user to determine if any common classes might serve as an abstraction of the

queried concept. We find that although various strategies are applied in different (controlled) natural language QA systems [2][3][4], all of them implement and integrate some kind of resource retrieval components. Most of these systems only use the straightforward strategy for resource retrieval, i.e. exact string match on the *rdfs:label* values, while a few QA systems enhance their retrieval component by expanding queries with WordNet. For example, Aqualog [3] uses synonyms, and Tran et al. [4] extract synonyms, hyponyms (subclasses) and hypernyms (superclasses).

While exact string match usually misses alternative expressions, there are problems with synonym match. First, in some domain specific KBs, people might use query terms that are not in the lexicon. Second, a synonym might have other meanings, and retrieving all occurrences of it can reduce precision. Finally, the ontology creators and KB users may sometimes use words that are not synonyms to refer to the same concept [5] under different circumstances. For example the creator of an academic ontology may use Person to name the concept of people at an academic institution; but this concept only consists of "Professors" and "Students". Meanwhile in many cases a partial match is useful. For the keyword query "professor", the *Person* class from the academic ontology may be suitable as a partial match, even though it is a super class. This is especially true if other constraints in the query restrict results to someone teaching a course. Thus alternative approaches for resource retrieval need to be investigated.

III. Class Retrieval Framework

Let $\mathcal{C}$ be the set of all the classes defined in the KB. A class $C \in \mathcal{C}$ can be interpreted as a set of instances. A subclass relationship $C_1 \sqsubseteq C_2$ means the instance set of C_1 is a subset of the instance set of C_2. The collection of subclass relationships establish a graph of class hierarchy. Classes and instances can have properties that relate them to other things, including literal values as well as other classes and instances. Among various properties, *rdfs:label* provides a human-readable name of the instance or class, and thus is frequently used for the class retrieval problem.

Formally, we define the class retrieval problem as: given a natural language query q, return a set of $\langle$class, score$\rangle$ pairs $\{\langle C_i, scr_i\rangle\}$, where $C_i \in \mathcal{C}$, and the score scr_i is how well C_i matches q. We propose a two-phase class retrieval framework.In the first phase, the query q is matched to instances' texts (which we shall soon define), instead of directly matched to classes' labels. This step can be done with a standard IR query; and a set of $\langle$instance, IR score$\rangle$ pairs $RS = \{\langle I_j, r_j\rangle\}$ are returned. Given a set of weighted matched instances, the problem in the second phase is then how to induce the class represented by these instances. Then scr_i is defined as a function that may take C_i, RS and the *rdf:type* relations (i.e. a property that denotes the class of an instance) in the KB as arguments. We shall further discuss different implementations of this function in the next section.

Intuitively, if we have collected sufficient instance texts, the common terms among these instances are very likely to be indicative of the class. In practice, instance texts can be generalized as a function on an instance, as well as the given KB, such as: (1) **annotation properties** such as *rdfs:label* (names) and *rdfs:comment* (despriptions). (2) **properties with high discriminability/coverage**. A list of properties such as job title/name/address can be automatically generated [6] and further manually selected. (3) **external links**. External knowledge, e.g. WWW search, or *owl:sameAs* links, can be used as instance texts as well. (4) **refining existing texts**, e.g. to extract key terms from any of the above instance texts. Both lexicon expansions and our instance-based proposal try to find more texts connected to class C_i in the graph (shown in Figure 1), but they have different research challenges: In contrast with thesaurus-based extraction, where ambiguous syntactic forms can potentially decrease relevance, our approach mostly depends on how well the instance texts can represent the class; and thus the important task is to specify useful texts and extract representative keywords from them.

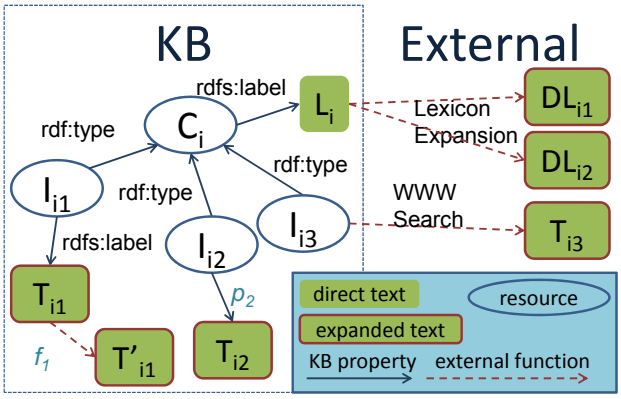

Fig. 1. Expanded Texts of a Class C_i

We use DBPedia 3.7 [7] for our experiments, because it includes various kinds of classes (cross-domain), and contains many *rdfs:label* and *rdfs:comment* (various texts). We find that for some classes, the name of each instance usually contains the category name, which can be an alternative to the label of (a subset of) its class. e.g. the label "Chesterton Community College" of an instance of class *d:EducationalInstitution* indicates the category name "Community College". However, the labels of other classes like *d:Person*, *d:Film*, and *d:Song* are less informative because the titles and names seldom contain relevant terms to the class. On the other hand, *rdfs:comment* (basically the content of the Wikipedia article) often contains useful terms for class retrieval, but are usually too long and contain many irrelevant terms too. However, the oft-repeated terms in these values are often closely **related** to the class. To enhance the chance that the selected texts accurately reflect the class, we introduce a third text type by refining the comments with simple string manipulations on the first sentance of it. Thus in total we index three types of texts: labels, comments, and fragments of comments with three indexing fields. Given a query q and a specified text feature (a single field or any weighted combination of them), a set of $\langle$instance, IR score$\rangle$ pairs can be retrieved via standard IR means; and the first phase in our framework is done.

IV. Inducing Classes from Instances

From the first phase we have a set of $\langle$instance, IR score$\rangle$ pairs $RS = \{\langle I_j, r_j\rangle\}$ as the results. For convenience, we define $\mathcal{I}_q = \{I_j | < I_j, r_j > \in RS\}$ the set of all the instances

in RS. The task of the second phase is to assign an appropriate score scr_i for each C_i in the KB. While the first phase can rely on a standard IR approach, we have more choices in how to induce a class in the second phase. In this section, we cast it in three different ways as discussed below.

A. Additive Value Function

We start from the most straightforward intuition: if an instance I_j of a class C_i is returned, the IR score r_j associated with I_j should somehow contribute to scr_i, the score of C_i. In utility theory, the influence of multiple attributes can be represented by an additive value function as long as we assume mutual preferential independence holds between the attributes. An additive value function is simply a multiatrribute function that is the sum of a set of single attribute value functions. Inspired by this idea, we define the additive value function (AVF) score of a particular class C_i as:

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \tag{1}$$

In this formula, we take the score of each instance in the class C_i, apply a normalization/transformation function T, and then simply apply a naive summation of each transformed value.

We can define T with simple functions. For example, define $T_0(r_j) = 1$ if $r_j > 0$, otherwise 0, which means every instance "votes" for its classes. Or set a threshold ϵ and let $T_\epsilon(r_j) = r_j$ if $r_j \geq \epsilon$ or 0 otherwise. If ϵ is set as the n-th greatest value in $\{r_j\}$, it means we only consider the top n matching instances.

Note that in a KB, $I_j \in C_i$ can be either explicit or entailed by ontological axioms. An instance thus can belong to multiple classes even in single-inheritance ontologies. This suggests that the score (or vote) $T(r_j)$ from each instance I_j should be apportioned among every class it belongs to. Thus a modified version is

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot f(I_j, C_i) \tag{2}$$

where $f(I_j, C_i)$ factors how the vote from I_j is divided. Naively, let nc_j be the total number of classes I_j belongs to, we can equally divide $T(r_j)$, i.e.

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T(r_j)}{nc_j} \tag{3}$$

However, this does not exactly reflect a general intuition that the more specific classes should be more favored. Instead, let $f(I_j, C_i) = \frac{1}{|C_i|}$, where $|C_i|$ denotes the size of instances entailed in C_i, we have

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T(r_j)}{|C_i|} \tag{4}$$

B. Instances as IR Queries

Another option is to consider the problem as another IR problem by treating each class as an IR document, and indexing each class with its instances' IDs as its content. In the index, each instance has a posting list of classes it belongs to. Then the problem in the second phase of our framework is cast as Boolean retrieval given a long query with query terms $\mathcal{I}_q$ to this index. A basic tf-idf approach is

$$scr_i = \sum_{\forall I_j \in \mathcal{I}_q} tf(C_i, I_j) \cdot idf(I_j) = \sum_{I_j \in C_i \cap \mathcal{I}_q} idf(I_j) \tag{5}$$

An instance is either a member of a class ($tf = 1$) or not ($tf = 0$), thus the tf merely denotes whether $I_j \in C_i$ is true (explicitly or by entailment) for this KB. Furthermore if we consider that query terms are not equally weighted, again we apply the transformed scores $T(r_j)$ associated with each I_j as the boost factor, and then we have

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot idf(I_j) \tag{6}$$

The idf is usually defined as $idf(I_j) = \log \frac{N_C}{df(I_j)}$, where N_C is the total number of documents (in our case classes). The number of classes is often relatively small in Linked Data datasets, for example, in DBPedia, $N_C = 319$. The log function is taken to scale the huge difference among document frequencies of terms ($df(I_j)$). However in our case, $df(I_j) = nc_j$, which we defined previously as the total number of classes I_j belongs to. In DBPedia, for most of the instances $nc_j = 3 \sim 5$. Since nc_j does not change in orders of magnitude, for most of the time we get $idf(I_j) = 1.80 \sim 2.02$, thus we can approximately treat $\log N_C / nc_j \approx \alpha$ as a constant. Then

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot idf(I_j) \approx \sum_{I_j \in C_i \cap \mathcal{I}_q} \alpha \cdot T(r_j) \tag{7}$$

is mathematically approximate to AVF in Eq. 1. To emphasize the difference, we define idf' without the log, then

$$scr_i = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot idf'(I_j) = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{N_C \cdot T(r_j)}{nc_j} \tag{8}$$

which is in proportion to Eq. 3, and thus is equivalent to it with regard to the output of rankings of classes. In practice, the basic tf-idf approach above is usually tuned with various normalization factors, and we will use a state-of-the-art IR tool in the experiment to evaluate this casting perspective.

C. Ontology Alignment Problem

We define a virtual class D_q, which is the concept that directly corresponds to the query need represented by the term q; and our task is to find the class best aligned to D_q. Note that not all the instances that contain keyword q are necessarily instances of D_q, and not all the instances of D_q necessarily contain keyword q in their text.

From the first phase, the result set $\langle I_j, r_j \rangle$ actually returns a set of instances that are likely to be instances of D_q, where the relevance score r_j indicates such likelihood. So we can first apply a function $T_p(r_j)$ which is the probability that I_j is an instance of D_q. In addition we use a factor α to compensate for the estimation errors that arise because D_q and the instances with keyword q are not perfectly aligned; eventually we can define a transform function $T_\alpha(r_j) = \alpha \cdot T_p(r_j)$. Then $T_\alpha(r_j)$ can be interpreted as the expected number of instances of D_q that I_j represents. The total size of D_q is estimated as

$$|D_q| \approx \sum_{\forall I_j \in \mathcal{I}_q} T_\alpha(r_j) \tag{9}$$

Based on the uniform assumption, we can estimate:

$$|D_q \sqcap C_i| \approx \sum_{I_j \in C_i \cap \mathcal{I}_q} T_\alpha(r_j) \tag{10}$$

With these estimated sizes, many existing approaches in instance-based ontology alignment can be applied. e.g. the the

most commonly used Jaccard (**Jcd**) approach is defined as:

$$scr_i = \frac{|D_q \sqcap C_i|}{|D_q \sqcup C_i|} = \frac{|D_q \sqcap C_i|}{|D_q| + |C_i| - |D_q \sqcap C_i|} \tag{11}$$

There are also measures for ontology matching based on information theory. e.g. The Pointwise Mutual Information approach (**PMI**) measures the reduction of uncertainty that the annotation of one concept yields for the other. Mathematically, it is the log of the ratio between the probability of their coincidence given their joint distribution and the probability of their coincidence given only their individual distributions:

$$scr_i = \log \frac{P(D_q \sqcap C_i)}{P(D_q) \cdot P(C_i)} = \log \frac{\frac{|D_q \sqcap C_i|}{N}}{\frac{|D_q|}{N} \frac{|C_i|}{N}} = \log \frac{|D_q \sqcap C_i| \cdot N}{|C_i| \cdot |D_q|} \tag{12}$$

where N is the total number of instances in the KB. PMI maximizes when $D_q \sqsubseteq C_i$ or $D_q \sqsupseteq C_i$ is true. Other measures in this category include log likelihood ratio, information gain, etc. A comprehensive comparison of measures in instance-based ontology alignment can be found in [8].

We also propose an alternative approach. First we can consider $P(C_i|D_q)$ which represents the probability that an instance has type C_i if we already know it has type D_q.

$$P(C_i|D_q) = \frac{|D_q \sqcap C_i|}{|D_q|} = \frac{\sum_{I_j \in C_i \cap \mathcal{I}_q} T_\alpha(r_j)}{|D_q|} \tag{13}$$

Note that scr_i is calculated for each class C_i given query q, and $|D_q|$ is just a normalization factor. So the output rankings of C_i of this approach is the same as that of AVF in Eq. 1. From one perspective, we evaluate each $C_i \in \mathcal{C}$ with some metric on how well C_i matches virtual class D_q. Then Eq. 13 evaluates each candidate C_i with the *virtual* recall of D_q. We can also use the *virtual* precision: the number of instances of D_q in C_i, i.e.

$$precision(C_i, D_q) = \frac{|D_q \sqcap C_i|}{|C_i|} = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T_\alpha(r_j)}{|C_i|} \tag{14}$$

which is identical to Eq. 4. In combination, we propose the *virtual* F-Measure approach (**FM**):

$$scr_i = \frac{2 \cdot \frac{|D_q \sqcap C_i|}{|C_i|} \cdot \frac{|D_q \sqcap C_i|}{|D_q|}}{\frac{|D_q \sqcap C_i|}{|C_i|} + \frac{|D_q \sqcap C_i|}{|D_q|}} = \frac{2 \cdot |D_q \sqcap C_i|}{|D_q| + |C_i|} \tag{15}$$

V. Evaluation

In this section, we first introduce our experiment setup, and then discuss the results of our proposed approaches.

A. Experiment Setup

In traditional IR, a document is either relevant or not, but classes can be generalizations of each other, and intuitively more specific classes should be better matches than very generic classes. Assuming the query term q represents a virtual concept D_q, we discuss different categories of matches to D_q that can be specified in the ground truth. Within each case, we define a function $rel(C_i, q)$ that indicates the degree of relevance of a retrieved class C_i to the query q.

1. **Equivalence Match**: We define $rel(C_i, q) = 1$ if and only if $C_i = D_q$ and 0 otherwise. In addition, we call an Equivalence Match a **Syntactic** match if the matched class has the same label as q , otherwise a **Synonym** match because the label, as a synonym of q, indicates the same concept.

2. **Partial Match**: Sometimes there is no equivalent class for q, but we may find classes very close to the virtual query class D_q. e.g. given "composer" return *d:MusicalArtist*; or given "physicist" return *d:Scientist*. Both examples are the best results the system can return, because they are the *upper bounds* of D_q. Similarly, there are *lower bounds* of D_q. Note there can be multiple upper bounds or lower bounds. Suppose the total number of upper bounds and lower bounds of a query are u and l respectively. We define three types of partial matches:

- **Superclass Match** if $u \geq 1$ and $l = 0$. We assign $rel(C_i, q) = 0.8/u$ if C_i is one of u upper bounds of D_q; otherwise it is 0.
- **Subclass Match** if $u = 0$ and $l \geq 1$. We assign $rel(C_i, q) = 0.8/l$ if C_i is one of l lower bounds of D_q; otherwise it is 0.
- **Bounded Match** if $u \geq 1$ and $l \geq 1$. We assign $rel(C_i, q) = 0.4/u$ if C_i is one of u upper bounds of D_q; assign $rel(C_i, q) = 0.4/l$ if C_i is one of l lower bounds of D_q; otherwise it is 0.

The $rel(C, q)$ values can be viewed as the expected utility of translating q into C for subsequent usage. We set these values based on rough estimations of the likelihood that such a match can be a satisfactory search result for a user, or the likelihood that such a match can be used to formulate a SPARQL query to get useful answers. Note that we actually define very strict goals for the retrieval task. In all three cases, we are asking what the best possible match could be for the query. Thus the superclasses of the upper bounds or subclasses of the lower bounds do not get a partial relevance score. We also debated whether we should give partial relevance scores to the overlapped classes. The benefit from retrieving an overlapped class C_x is really determined by the ratio of the common part of C_x and D_q. Since we need human judgment to produce ground truth for the evaluation, in order to increase the intra-human agreement as well as reduce human effort, we simply treat all the overlapped class as non-relevant.

For our evaluation, we call a query q a qualified query if and only if $\exists C_x$ in the KB, s.t. $rel(C_x, q) > 0$. We extract terms from the DBPedia links to WordNet, and expand the query set by the adding the synonyms (**e.S**) and hypernyms (superclasses of terms) of the original terms from WordNet. For hypernyms, we use not only the direct (level 1, **e.H1**) hypernyms, but also hypernyms of level 2 (**e.H2**) and level 3 (**e.H3**). By expansion, we ended up with 184 candidate queries.

We implemented an interface that provides the evaluators with a hierarchy, aided with reasoning on disjointness and subsumptions. Three native English speakers provided their judgment on whether an ontological class is a super/equivalent/sub/overlapped/disjoint concept to the query. After using majority vote, there were only 4 queries that lacked intra-judge agreement and we dropped them from the query set. A summary of the data set is presented in Table I. This table categorizes the queries by two dimensions: how a query is generated (the columns) and the ground truth of the query (the rows).

TABLE I. SUMMARY OF QUERY DIMENSIONS

	All	e.O	e.S	e.H1	e.H2	e.H3
All	180	106	13	28	17	16
Syntactic	68	53	3	7	1	4
Synonym	26	15	2	5	2	2
Super	42	26	6	7	3	0
Sub	12	0	0	3	5	4
Bounded	32	12	2	6	6	6

To best evaluate the top-k retrieved classes, we use Discounted Cumulative Gain (DCG) [9] as our metric. i.e.

$$\mathrm{DCG}_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i} \tag{16}$$

When a query is issued, the top $p = 10$ matched classes are returned, and by using the average of relevance scores from human judgment, we get the DCG score for this query.

B. Experiment Results

We use Additive Value Function (AVF) from Eq. 1, , an IR method (Luc) based on Eq. 6, Jaccard (Jcd) from Eq. 11, F-Measure (FM) from Eq. 15, and Pointwise Mutual Information (PMI) from Eq. 12 introduced in Section 4 as our test systems. The IR algorithm uses the state-of-the-art IR system Lucene 3.5, which uses a combined Boolean model and Vector Space Model scoring method. We say the Luc approach is "based on Eq. 6" because although it is similar, the weighting and normalization factors are much better tuned for standard IR tasks (detailed in javadoc of *org.apache.lucene.search.Similarity*). All of the above systems use instance texts. As we introduced in Section 3, the texts we chose are labels (L), comments (C) and fragments of comments (F). To compare with our proposed approach, we also implemented two baseline systems that only uses class labels: the string match method **SL** with Scaled Levenstein (edit distance similarity), and the lexicon method **WN** to match synonyms of queries provided by WordNet. To avoid extra factors complicating the analysis, we define the transforming function as $T(r_j) = r_j$, i.e. we directly use the IR scores from results in the first phase, and we set a constant ratio $\alpha = 1$. Our future work includes experiments with different T functions and variable α on different queries.

Figure 2 shows the overall comparison of average DCG scores among the combination of the systems and text fields, in contrast with the baselines and the ideal DCG score. Among different systems, we find that Jcd, FM, and Luc have a better performance than the others, and it suggests that if we use Jcd, FM, or Luc, our proposed idea of using instance texts can provide better class results than the syntactic matching approaches on class labels. For different text fields, we can see that F is the best feature in general, which is sometimes as good as C and sometimes slightly better than C. Using L seems to be less helpful than the other two fields. That is because, as we discussed in Section 3, labels of instances do not usually provide useful terms that refer to the class of that instance. However, it is still useful if we manipulate it with a right approach. In this experiment, we did not try to use multiple fields as a text feature. However we have noticed the fact that in some cases one field provides more useful information but in other cases introduces more noise than the other. We believe by combining these different fields, we can expect improvement for the class retrieval problem. We will

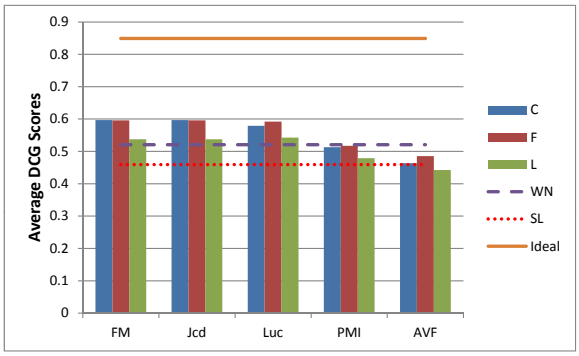

Fig. 2. Comparison on the Overall Query Set

study this in our future work. Also, interestingly, we find the coincidence that FM and Jcd always returns the same rankings of classes for all the queries in this experiment. After reflection, we realized that since we did not transform the IR score r_j, we always underestimated $|D_q \sqcap C_i|$ in Eq. 10; and thus in Eq. 11 $|D_q \sqcap C_i|$ is usually negligible when compared to $|D_q| + |C_i|$; the typical value of $\frac{|D_q \sqcap C_i|}{|D_q|+|C_i|} = 10^{-5} \sim 10^{-4}$ in our test set, thus it it makes Eq. 11 highly similar to Eq. 15. Since there is no significant difference between FM and Jcd, in the rest of the paper we only present the results of FM.

We also compare the systems in different dimensions of queries. Figure 3 shows the comparison on match types of queries. We compare the four systems on the same field F, to see how each system performs against queries with different matching types. There are several things we want to point out. First we want to explain the change of average ideal scores. There are two reasons why an ideal score becomes small: (1) there are many partial matches, which get partial scores; (2) a partial match has too many classes as its bounds. The upper bounds usually have only one class for each query, however there could be a lot for the lower bounds of a query. Although we set the sum of rel scores of these bounds to 0.8, DCG calculation results in any class suggested at rank 3 or less will be discounted, even if there are three or more matches in the ground truth. This is why we see exact matches have 1.0 and Superclass matches have 0.8 as their ideal scores, while Subclass matches have very a small ideal scores and Bounded match is between Superclass match and Subclass match. Secondly, we inspect the performance of baseline systems. As we can expect, SL works perfectly on Syntactic matches, and sometimes finds matches in Synonym match queries thanks to the partial string match. e.g. "character" matches to *d:FictionalCharacter* and "official" matches to *d:OfficeHolder*. However it has difficulty finding partial string matches in other match types, and the score drops dramatically. Similar to SL, WN is perfect at Syntactic matches, and is very good at Synonym matches because of the lexical expansion on queries to match to class labels. For Superclass matches and Bounded matches, we find that such lexical expansion continues benefiting WN; however for Subclass matches, it makes WN worse than SL: when we expand the query too much, we reduce the precision while increasing the recall. Also, we want to point out that the experiment is a little biased towards WN, mainly because we generate the query terms using WordNet. Lastly we compare the performances of

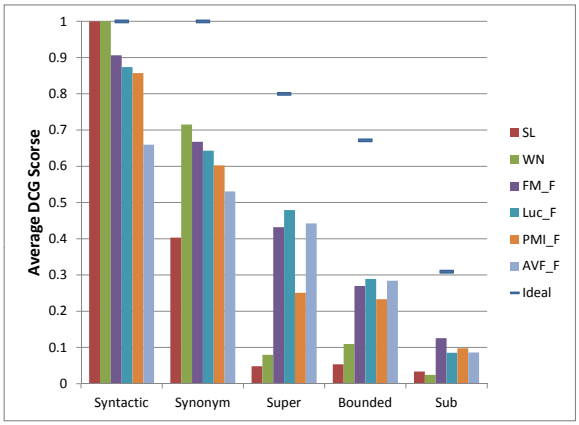

Fig. 3. Comparison on the Match Type Dimension

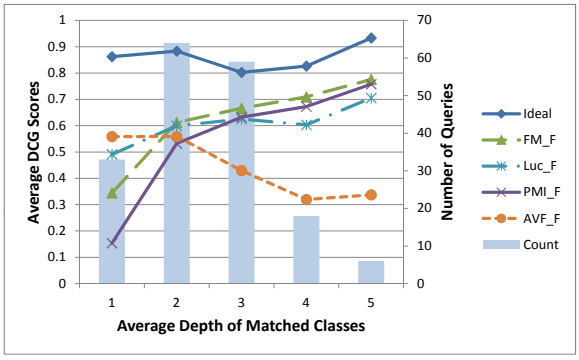

Fig. 4. Comparison on the Average Depth of Matched Classes

proposed systems on different match types. We can see that FM is the best on Exact and Subclass matches, however Luc and AVF become better for Superclass and Bounded matches. By examining Eq. 1 of AVF, we can see that AVF has no factoring for classes. While each instance adds some utility to each of its classes, the more general classes are more likely to get larger scores. Thus AVF always favors general classes, and effectively finds upper bounds (usually general classes) for Superclass and Bounded matches. On the other hand, FM has a factor of $|C_i|$ that penalizes general classes, thus it is less likely to get upper bounds in these cases. Luc, however has a moderate factoring on $|C_i|$, which is encoded in its normalization of document sizes (which in our case is $|C_i|$), and thus exhibits good performance over different match types.

To further inspect how the systems perform on queries that match to general or specific classes, we divide the query set in a third way. We define the depth of a class as the shortest path length from this class to the top class *owl:Thing* in the class hierarchy graph. For each query, we calculate the average depth for all the bound classes (or the exact class if it is an exact match), and round to the nearest integer. As a result, general queries have a small average depth, while specific queries have a larger average depth. We compare FM, Luc, PMI, and AVF on field F in Figure 4. We can see that each system has a clear trend of performance as the average depth increases. Most systems achieve better class retrieval results if the query becomes more specific, while only AVF favors the general queries. In other words, FM, Luc, and PMI are leaning towards returning specific classes in the KB because $|C_i|$ all appear in their formula as a factor discounting the general classes. While such discounting is desirable for specific queries, we wonder whether in some systems it is over-discounting. From the figure we can see that Luc is most robust with the change of generality of queries, while PMI is most sensitive to it.

VI. Conclusion and Future Work

In this paper, we addressed the keyword based class retrieval problem. Unlike traditional approaches that directly match the query to the labels of classes, we proposed a two-phase framework that utilizes the texts from instances to improve class retrieval. The main challenge for the first phase is to define which types of texts from instances we want to use for class retrieval, and then perform a standard IR search on this text. The second phase is to induce classes from the instances produced by the first phase. We took different perspectives to analyze the problem, showing that in the end some viewpoints lead to equivalent results with regard to ranking matching classes. A nearly 20% improvement in DCG scores is achieved when comments or comment fragments are used as the instance text for our proposed instance-based approach comparing to the baseline systems. We also showed that the F-measure, instances-as-queries, and PMI approaches all performed better when the best matching classes were deeper in the hierarchy, but that the additive value function performed best for very shallow classes. For future work, we will continue to study alternative approaches to the problem, including new approaches for inducing the class, alternative text fields (or combination of text fields), and the impact of various of transforming functions. Additionally, we will evaluate the methods on other datasets.

References

[1] G. Tummarello, R. Delbru, and E. Oren, "Sindice.com: weaving the open linked data," in *6th International Semantic Web Conference and 2nd Asian Semantic Web conference*, 2007, pp. 552–565.

[2] A. Bernstein, E. Kaufmann, A. Ghring, and C. Kiefer, "Querying ontologies: A controlled English interface for end-users," in *4th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, 2005, pp. 112–126.

[3] V. Lopez, M. Pasin, and E. Motta, "Aqualog: An ontology-portable question answering system for the Semantic Web," in *2nd European Semantic Web Conference*, 2005, pp. 546–562.

[4] T. Tran, H. Wang, S. Rudolph, and P. Cimiano, "Top-k exploration of query candidates for efficient keyword search on graph-shaped RDF data," in *25th International Conference on Data Engineering*, 2009, pp. 405–416.

[5] J. Fan and B. Porter, "Interpreting loosely encoded questions," in *19th National Conference on Artifical Intelligence*, 2004, pp. 399–405.

[6] D. Song and J. Heflin, "Automatically generating data linkages using a domain-independent candidate selection approach," in *10th International Semantic Web Conference*, 2011, pp. 649–664.

[7] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, 2007, pp. 722–735.

[8] A. Isaac, L. Van Der Meij, S. Schlobach, and S. Wang, "An empirical study of instance-based ontology matching," in *6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, 2007, pp. 253–266.

[9] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, pp. 422–446, October 2002.