Web Semantics: Science, Services and Agents on the World Wide Web xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Ontology-driven, unsupervised instance population

Luke K. McDowell [a,*], Michael Cafarella [b]

[a] Computer Science Department, U.S. Naval Academy, 572M Holloway Road Stop 9F, Annapolis, MD 21402, USA
[b] Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

ARTICLE INFO

Article history:
Received 14 September 2007
Received in revised form 27 March 2008
Accepted 14 April 2008
Available online xxx

Keywords:
Semantic Web
Ontology-driven
Instance population
Classification
Confidence assessment

ABSTRACT

The Semantic Web's need for machine understandable content has led researchers to attempt to automatically acquire such content from a number of sources, including the web. To date, such research has focused on "document-driven" systems that individually process a small set of documents, annotating each with respect to a given ontology. This article introduces OntoSyphon, an alternative that strives to more fully leverage existing ontological content while scaling to extract comparatively shallow content from millions of documents. OntoSyphon operates in an "ontology-driven" manner: taking any ontology as input, OntoSyphon uses the ontology to specify web searches that identify possible semantic instances, relations, and taxonomic information. Redundancy in the web, together with information from the ontology, is then used to automatically verify these candidate instances and relations, enabling OntoSyphon to operate in a fully automated, unsupervised manner. A prototype of OntoSyphon is fully implemented and we present experimental results that demonstrate substantial instance population in three domains based on independently constructed ontologies. We show that using the whole web as a corpus for verification yields the best results, but that using a much smaller web corpus can also yield strong performance. In addition, we consider the problem of selecting the best class for each candidate instance that is discovered, and the problem of ranking the final results. For both problems we introduce new solutions and demonstrate that, for both the small and large corpora, they consistently improve upon previously known techniques.

Published by Elsevier B.V.

1. Introduction

The success of the Semantic Web critically depends upon the existence of a sufficient amount of high-quality, relevant semantic content. But to date relatively little such content has emerged. In response, researchers have investigated systems to assist users with producing (or annotating) such content, as well as systems for automatically extracting semantic content from existing unstructured data sources such as web pages.

Most systems for automated content generation work as follows: the system sequentially processes a small to moderate size set of mostly relevant documents. For each document, the system tries to extract relevant information and encode it using the predicates and classes of a given ontology. This extraction might utilize a domain-specific wrapper, constructed by hand [30] or via machine learning techniques [10]. More recent domain-independent approaches have utilized a named entity recognizer to identify interesting terms, then used web searches to try to determine the term's class [13]. In either case, these are *d* ocument-driven systems whose workflow follows the documents.

This article describes OntoSyphon, an alternative *o* ntology-driven information extraction (IE) system. Instead of sequentially handling documents, OntoSyphon processes the ontology in some order. For each ontological class or property, OntoSyphon searches a large corpus for instances and relations that can be extracted. In the simplest case, for instance, a `Mammal` class in the ontology causes our system to search the web for phrases like "mammals such as" in order to identify instances (and subclasses) of `Mammal`. We then use redundancy in the web and information in the ontology to verify the candidate instances, subclasses, and relations that were found. In this article, we focus on learning instances to populate an input ontology.

Compared to more traditional document-driven IE, OntoSyphon's ontology-driven IE extracts relatively shallow information from a large corpus of documents, instead of performing more exhaustive (and expensive) processing of a small set of documents. Hence, the approaches are complementary, and real world systems may profitably utilize both. We note, however, several benefits of ontology-driven IE. First, driving the entire IE process directly from the ontology facilitates the direct use

* Corresponding author. Tel.: +1 410 293 6800; fax: +1 410 293 2686.
E-mail address: lmcdowel@usna.edu (L.K. McDowell).

1570-8268/$ – see front matter. Published by Elsevier B.V.
doi:10.1016/j.websem.2008.04.002

Table 1
A summary of work that attempts to (semi-)automatically extract instance-like content from the web or other text corpora

	Text-based	Ontology-based	
		Document-driven	Ontology-driven
Domain-specific	Crystal [58], Citeseer, Opine [51]	WebKB [18], TAP [30], OntoMiner [20], OntoSophie [9]	Cyc "web population" [39,56], van Hage et al. [62], Geleijnse et al. [29]
Domain-independent	MindNet [54], Snowball [1], Cederberg et al. [8], Google sets [48], KnowItAll [24], Pantel et al. [47], Qualia learning [16], Espresso [45], Pasca et al. [49], Pronto [5]	Hahn and Schnattinger [31], S-CREAM [32], SemTag [21], KIM [34], Armadillo [10], PANKOW [11], Li and Bontcheva [37]	**OntoSyphon**

There are many systems that produce output *based on* (in terms of) a given ontology, but little work has exploited the ontology to actually *drive* the extraction process. Note that an ontology-based system almost always utilizes a domain-specific ontology, but may still be a domain-independent *system* if it can easily exploit input ontologies from many different domains.

of multiple kinds of ontological data, e.g. utilizing class labels, synonyms, and sample instances for broader searching. Second, a search-based system enables us to consider a much larger set of documents than could be handled via individual, document-driven processing. Only a small fraction of the corpus will be used for any one system execution, but much more potentially relevant information is accessible. Finally, ontology-driven IE can be easily focused on the desired results. Rather than processing all content from some documents and then looking for the desired info, we can instruct the system to search directly for relevant classes.

Our contributions are as follows. First, we describe the ontology-driven paradigm for information extraction and explain its benefits compared to complementary approaches. Second, we explain how to apply this general paradigm to find instances from the web and demonstrate successful instance population for three different, independently created ontologies. Third, we propose new techniques for the problem of selecting the most likely class for each candidate instance that is found, and evaluate a large number of such techniques. We show that a new technique, by explicitly exploiting the ontology in its calculations, yields substantially improved precision. Fourth, we examine and evaluate several different techniques for improving the ranking of the extracted instances based upon automatic probability or confidence assessment. In particular, we introduce two simple but highly effective improvements to previously known assessment techniques for such extractions. These improvements relate to adding or improving upon frequency-based normalization, and can be used even in contexts without an explicit ontology. Fifth, we examine the impact of varying the corpus that is searched for instances, and show that its size affects different parts of a precision-recall curve in different ways. Sixth, we consider the impact of using two different verification corpora: the whole web ($Corpus_L$) or a smaller 60-million page subset ($Corpus_S$). We show that using $Corpus_L$ yields the best results on average, but that strong results can still be obtained with $Corpus_S$. In addition, we introduce improvements to an existing assessment technique (PMI-Bayes) that is appropriate for the larger corpus. Finally, we demonstrate that our other new techniques for class picking and instance ranking work consistently well across both corpora, but that the best metrics for class picking are not the same as the best metrics for instance ranking.

Our recent work [43] presented the general vision for OntoSyphon and preliminary results. This article, however, is the first to utilize the entire web as OntoSyphon's verification corpus (Section 5.1.3), to evaluate the overall impact of this choice compared to a smaller corpus (Section 6.2), to measure the effect of the size of the corpus that is searched for instances (Section 6.3), and to separately consider and evaluate the effect of the class picking algorithm (Section 6.1). In addition, the enhancements to PMI (Section 5.1.3), the development of class normalization for "Urns" and "Strength" (Sections 5.1.1 and 5.1.2), and the structure-based transformation of probabilities (Section 5.2) are new. Finally, results in this article are based on a larger gold standard (Section 4), and results for the first time are presented using both Learning Accuracy and precision (Section 6).

The next section summarizes related work in this area. Section 3 summarizes OntoSyphon's operation, while Section 4 describes our methodology and evaluation measures. Section 5 describes the existing and new techniques that we use for the key problem of assessing candidate instances. Finally, Section 6 presents experimental results, Section 7 discusses our findings, and Section 8 concludes.

2. Related work on information extraction from the Web

The general task we face is to extract information from some textual source, such as the WWW, and encode that information in a structured language such as RDF. Table 1 provides a general interpretation of the most relevant other work in this area, as discussed below. Section 5 provides further discussion on related work regarding the assessment of extracted knowledge.

The rows of Table 1 distinguish systems that are domain-independent from those that rely on domain-specific techniques or extraction patterns. The columns explain the extent to which each system utilizes an explicit ontology. In the leftmost column ("Text-based") are IE systems that are not explicitly based on an ontology. For instance, Citeseer automatically extracts metadata about research publications, Opine [51] focuses on product reviews, and Crystal [58] uses a domain-specific lexicon to learn text extraction rules by example. Amongst more domain-independent systems, MindNet [54] builds a semantic network based on dictionary and encyclopedia entries, while Snowball [1] learns relations (such as `headquartersOf`) based on an initial set of examples. KnowItAll [24] learns instances and other relations from the web, and Pasca [48] identifies and categorizes arbitrary named entities into logical sets. Pasca et al. [49] learn patterns to extract a large number of person-birthYear pairs, using the distributional similarity of each pair with respect to a dynamic set of seeds for validation. Many such systems [8,24,47,16,45,5] learn hyponym or is-a relationships based on searching for particular lexical patterns like "cities such as . . .," inspired by Hearst's original use of such patterns [33]. Our work uses these same patterns as building blocks, but exploits an ontology to guide the extraction and assessment, and to formally structure the results.

Some of these text-based systems, such as MindNet, use their input corpus to derive an ontology-like structured output. In contrast, we call a system *ontology-based* if it specifies its output in

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

terms of a pre-existing, formal ontology.[1] These systems almost always use a domain-specific ontology in their operation, but we consider a system to be domain-independent if it can operate without modification on ontologies covering a wide range of domains.

The majority of these ontology-based systems are *d* ocument-driven: starting from a particular document (or set of documents), they try to annotate all of the entities in that document relative to the target ontology. For instance, TAP [30] exploits a variety of wrappers to extract information about authors, actors, movies, etc. from specifically identified websites such as Amazon.com. WebKB [18] uses supervised techniques to extract information from computer science department websites, and Armadillo [10] has been applied to the same task. It uses a variety of structured and unstructured data sources, some particular to its department website domain, but it can also be adapted to other tasks. Amongst more strictly domain-independent systems, SemTag [21] and KIM [34] scan documents looking for entities corresponding to instances in their input ontology. Likewise, S-CREAM [32] uses machine learning techniques to annotate a particular document with respect to its ontology, given a set of annotated examples. PANKOW [11,13] annotates a specified document by extracting named entities from the document and querying Google with ontology-based Hearst phrases. For instance, if the entity "South Africa" is found in a document, PANKOW would issue multiples queries like "South Africa is a river" and use hit count results to determine which ontology term (river, country, etc.) was the best match. These systems all use an ontology to specify their output, but make limited use of information that is contained in the ontology beyond the names of classes and properties that may be relevant. A recent exception is the system of Li and Bontcheva [37], which annotates documents using a hierarchy of supervised classifiers that have been informed by the structure of the ontology.

OntoSyphon offers a complementary approach of being ontology-based and *ontology-driven*. Instead of trying to learn all possible information about a particular document, we focus on particular parts of an ontology and try to learn all possible information about those ontological concepts from the web. In addition, we seek to use ontological data and structure to enhance our assessment of the content that is found (see Section 5).

The only work of which we are aware that adopts a somewhat similar approach is that of Matuszek et al. [39,56], van Hage et al. [62], and Geleijnse et al. [29]. All three systems use an ontology to generate web search terms, though none identifies this ontology-driven approach or examines its merits. van Hage et al. use the searches to find mappings between two given ontologies, whereas Matuszek et al. use the searches to identify instances and relations that could be inserted into the (large) Cyc ontology. Geleijnse et al. use queries based on an initial set of instances to identify additional instances and relations to populate a hand-crafted ontology. Matuszek et al. use more sophisticated natural language processing than we do, and use the existing Cyc ontology to perform more kinds of reasoning. Compared to OntoSyphon, however, these three systems perform much less sophisticated verification of content learned from the web, either assuming that a human will perform the final verification [39], assuming that all web candidates are correct [62], or using an unnormalized threshold test to accept instances [29]. In addition, van Hage et al. and Matuszek et al. only discover information about instances or classes that are already present in their ontology. Finally, all three systems are at least partially domain-specific. Matuszek's system, for instance, depends upon manually generated search phrases for a few hundred carefully chosen properties, and Geleijnse's system depends heavily on domain-customized named entity recognition and search patterns, though recent work has begun to address the latter limitation [27]. van Hage describes some domain-independent approaches and others that rely upon a domain-specific dictionary.

Ontology learning systems seek to learn or extend an ontology based on examination of a particular relevant corpus [12,38,2,15,63,53,44]. Some such systems [38,2,15,44] use Hearst-like patterns to identify possible subclass relations, similar to OntoSyphon's use of such patterns to identify candidate instances. Ontology learning systems, however, presume a particularly relevant corpus and do not focus on learning instances (with some limited document-driven exceptions, e.g., Text2Onto [15]). In addition, the goal of producing a very accurate ontology leads to very different verification techniques, usually including human guidance and/or final verification. OntoSyphon instead operates in a fully automatic, unsupervised manner, and uses the web rather than require that a domain-specific corpus be identified. We note, however, that OntoSyphon's ability to operate in this unsupervised manner depends upon the provision of a suitable starting ontology, which ontology learning systems may not require.

This article demonstrates how a domain-independent, ontology-driven system can reliably extract instances using a few simple patterns. We focus particularly on describing existing and new assessment techniques that improve reliability with these patterns. Overall performance could be increased even more by incorporating other techniques such as the automatic learning of additional subclasses for the ontology [24] or the learning of domain-specific extraction patterns [1,57,24,45,5]. Likewise, non-pattern based techniques that instead classify or cluster terms based on their common syntactic or document contexts could assist with the verification of instances found by OntoSyphon [31,38,2,12]. Cimiano et al. have considered how to combine such heterogeneous sources of evidence for the classification of ontological terms [14]. In addition, learning from more structured sources such as HTML tables or formatted pages [36,26], Wikipedia [55,59,50], or online dictionaries [54,66] can yield good results and easier verification for some domains, though these sources often cannot provide the broad and timely coverage of the general web.

This article focuses on learning instances (*is-a* relationships), not learning more general relationships for the instances [65,52,59]. However, we've recently shown that our general approach of exploiting web-based redundancy can also be applied to learn textual relations [3], and will demonstrate its effectiveness for semantic relations in future work.

Finally, OntoSyphon currently uses very lightweight natural language processing (primarily a simple part-of-speech tagger) to extract information. Other researchers have examined much more sophisticated NLP processing (e.g., [19]), including Matuszek et al. [39] and Li and Bontcheva [37] discussed above. These approaches are somewhat complementary. Stronger NLP systems can extract more complex information from each document, but their processing time limits the number of documents that can be considered. Our current focus is to determine how much information we can leverage from web-scale, lightweight techniques, leveraging the redundancy of the web and the precomputed indexes of popular search engines to our advantage.

3. Overview of OntoSyphon's operation

Fig. 1 gives pseudocode for OntoSyphon's operation. The input to OntoSyphon includes an ontology *O*, a root class *R* such as `Animal`, and two corpora CorpusSearch and CorpusVerify (which may be the

[1] This is a weaker definition than the ontology-oriented vs. ontology-based distinction recently proposed by Li and Bontcheva [37], but serves here to distinguish systems that do not have an ontology in view.

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

Algorithm (given an input ontology O and root class R)	**Sample Output** (of each step)
Initialization: searchSet = {R} + O.subclassesOf(R)	searchSet = {Animal, Amphibian, Bird, Fish, ...}
Operation: 1. c = PickAndRemoveClass (searchSet)	c = Bird
2. phrases = ApplyPatterns(c)	phrases = {"birds such as ...", "birds including ...", "birds especially ...", "... and other birds", "... or other birds"}
3. candidates += FindInstances (phrases, CorpusSearch)	candidates = {..., (kookaburra, Bird), (oriole, Bird), ... }
4. If MoreUsefulWork(searchSet, candidates), goto Step 1	
5. rawStats = GetHitCounts (candidates, CorpusVerify) *(the notation (oriole, Bird, 20) describes a candidate instance with 20 hits. For simplicity, example statistics are shown aggregated across all patterns; some metrics (e.g., "PMI") actually use per-pattern statistics (see Section 5).)*	rawStats = {(kookaburra, Bird, 20), (kookaburra, Mammal, 1), (wildebeest, Animal, 56), (wildebeest, Mammal, 6), (leather, Animal, 1), (oriole, Bird, 37) }
6. metrics = CalcMetrics (rawStats, candidates, $P_{pick-metric}$) *(computes scores or probabilities using one of the techniques described in Section 5.1. Results at right show probabilities computed with $P_{pick-metric}$ = Urns.)*	metrics = {(kookaburra, Bird, 0.93), (kookaburra, Mammal, 0.36), (wildebeest, Animal, 0.91), (wildebeest, Mammal, 0.85), (leather, Animal, 0.01), (oriole, Bird, 0.93) }
7. candidates = PickBestClass (metrics, O, $P_{pick-method}$) *(picks best class for each instance using one of the techniques from Section 5.2. Here $P_{pick-method}$ = PickMax.)*	candidates = {(kookaburra, Bird), (wildebeest, Animal), (leather, Animal), (oriole, Bird) }
8. metrics = CalcMetrics (rawStats, candidates, $P_{rank-metric}$) *(recomputes metric values. Results at right show confidence values computed with $P_{rank-metric}$ = Str-INorm.)*	metrics = {(kookaburra, Bird, 0.0342), *LA: 1.00* (wildebeest, Animal, 0.0017), *LA: 0.67* (leather, Animal, 0.000007), *LA: 0.00* (oriole, Bird, 0.0077) } *LA: 1.00*
9. results = SortAndFilter (metrics, $P_{rank-threshold}$) *(here $P_{rank-threshold}$ = 0.0001. LA is the "Learning Accuracy" of each pair (1.0 is fully correct, see Section 4).)*	results = {(kookaburra, Bird, 0.0342), *LA: 1.00* (oriole, Bird, 0.0077), *LA: 1.00* (wildebeest, Animal, 0.0017) }, *LA: 0.67*

Fig. 1. OntoSyphon's algorithm for populating an ontology O with instances, along with partial sample output.

same) that are used to, respectively, identify an initial set of candidate instances and to verify those candidates. The last inputs are the parameters $P_{\text{pick-metric}}$, $P_{\text{pick-method}}$, $P_{\text{rank-metric}}$, and $P_{\text{rank-threshold}}$, which control what techniques are used to select the most likely class for each candidate term and to rank the final results. The output is a set of instances of R and its subclasses, along with associated probabilities or confidence scores for each.

In summary, the algorithm proceeds as follows. The search set is initialized to hold the root term R and all subclasses of R. OntoSyphon then performs the following steps: pick a "promising" class c from the ontology (step 1), instantiate several lexical phrases to extract candidate instances of that class from CorpusSearch (steps 2 and 3), then repeat until a termination condition is met (step 4). Step 5 obtains raw statistics on each candidate instance from CorpusVerify. OntoSyphon then uses these statistics and the ontology O to pick the most likely class for each candidate term (steps 6 and 7) and to then assess the probability of each candidate instance (steps 8 and 9). Below we explain in more detail.

(1) *Identify a promising class*: OntoSyphon must decide where to focus its limited resources. For this article's experiments, we pragmatically chose to completely explore all subclasses of the user-provided root class. Future work should consider how best to use OntoSyphon's limited resources when broader explorations are desired. For instance, we could chose the class that we know the least about (fewest instances), or we could instead focus attention on classes that are similar to those that yielded good results in the past. Finally, note that some classes (e.g., zip codes) may produce very large amounts of data that is accurate but uninteresting.

(2) *Generate phrases*: Given a class c, we search for lexico-syntactic phrases that indicate likely instances of c. For instance, phrases like "birds such as" are likely to be followed by instances of the class `Bird`. We use the 5 Hearst phrase templates [33] listed in the sample output of Fig. 1. To instantiate the phrase, we must convert the class c to some textual description. This is trivial when the ontology provides a natural language string for each class (e.g., via `rdfs:label`), but very difficult if the most distinguishing property is an automatically generated label such as `ae54b0123983:a34bc124`. Fortunately, ontology designers frequently use meaningful and consistent notation for classes, and we can use heuristic processing to convert IDs such as `Sweet-Dessert`to the search label "sweet desserts." In particular, we split terms based on capitalization, spacing, and underscores, then convert the last word found to plural form as needed. This does limit OntoSyphon's applicability to ontologies with lexicalized class names or labels. However, where present we also exploit alternative class labels that can be inferred from the ontology, e.g., through the definition of an equivalent class.

(3) *Search and extract*: Next, we search CorpusSearch for occurrences of these phrases and extract candidate instances. CorpusSearch could be the entire web, where relevant pages are identified with a search engine, downloaded, and processed locally for extractions (as with KnowItAll [24] and C-PANKOW [13]). Alternatively, CorpusSearch could be a local subset of the web, or any other collection of documents. For efficiency, we use a 60-million page fragment of the web (henceforth, $Corpus_S$), and employ the Binding Engine (BE) [7] over this corpus. BE accepts queries like "birds such as < NounPhrase >" and returns all possible fillers for the < NounPhrase > term in about a minute. The overall algorithm, however, could be used

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

Table 2
The domains and ontologies used for our experiments

Domain	Ontology used	# Subs.	Avg. Depth	# Graded	Sample subclasses
Animals	sweet.jpl.nasa.gov/ontology/biosphere.owl	28	1.04 (max 2)	607 (10%)	Arthropod, Bird, Reptile
Artists	www.kanzaki.com/ns/music.rdf	39	2.63 (max 4)	952 (5%)	Composer, Flutist, Singer
Food	www.w3.org/TR/owl-guide/food.rdf	31	2.13 (max 3)	344 (10%)	BlandFish, Meat, Pasta

The third column gives the number of subclasses of the chosen root term, followed by the average (and maximum) depth of these subclasses relative to the root term.

with any corpus that enables us to search for instances using the pattern phrases.

(4) *Repeat* (for this article, until *SearchSet* is empty).

(5) *Compute hit counts*: To enable later steps to perform assessment, step 5 queries CorpusVerify for hit count information about each candidate instance (Section 5.1.4 examines the time/query complexity of this step). In this work, we consider two representative sizes for CorpusVerify: the smaller $Corpus_S$ (the 60 million page collection discussed above) and the larger $Corpus_L$ (the whole web). For $Corpus_S$, BE actually provides all of the necessary statistics when instances are extracted in step 3, so no additional work is needed. For $Corpus_L$, we use the web API of a popular commercial search engine to query for the needed counts.

(6) *Compute confidence metrics*: Given the statistics from step 5, OntoSyphon computes a confidence score or probability for each candidate pair such as (*tiger*, *Mammal*) (see Section 5.1).

(7) *Choose the best class for each instance*: In many cases, more than one candidate pair is found for a given instance *i*, e.g., (*wildebeest*, *Animal*) and (*wildebeest*, *Mammal*). In some cases, more than one answer may indeed be fully correct, but for simplicity and to avoid producing an ontology with redundant information, this step chooses the single best class for each term (see Section 5.2).

(8) *Re-compute confidence metrics*: Scores are re-computed in this step, since the best metric for choosing the most likely class in step 7 may not be ideal for the ranking and filtering of step 9.

(9) *Rank and filter the final results*: Since not all instances will be correct, providing the end user with some indication of the more likely instances is critical. This step uses the scores from step 8 to sort the final results. If desired, it will also expunge any instance pairs below a given confidence or probability value.

We focus in this article on basic instance learning, but this algorithm naturally lends itself to several future enhancements. For instance, in step 3, the candidate instances that are discovered will also discover subclasses. Such subclasses might be added to the *S* earchSet and/or might be used to extend the ontology itself. Our initial experiments have shown that, as is to be expected, such steps will increase recall but at some cost of precision. The next section discusses how we grade discovered subclasses for this work; future work will more fully investigate the benefits of exploiting these candidate subclasses. This will also necessitate more work on picking the best class to pursue (step 1), and deciding when to terminate the extraction process (step 4).

4. Methodology

We ran OntoSyphon over the three ontologies shown in Table 2. All three ontologies were created by individuals not associated with OntoSyphon, and are freely available on the web. For each, we selected a prominent class to be the "root class," thereby defining three different domains for evaluation: Animals, Food, and Artists. Note that instances for the first two domains are dominated by common nouns (dog, broccoli), whereas the latter yields mostly proper nouns (Michelangelo). These choices encompass different domains and ontology types. For instance, the Animal ontology was fairly complete but shallow, while the Artist ontology covers artists in general but most classes focus on musical artists. The Food ontology has been used for demonstrating OWL concepts; it contains some more complex constructions and classes such as `NonSpicyRedMeat`.

OntoSyphon operates in a totally unsupervised manner and outputs a ranked list of candidate instances for the ontology. Because there is no accurate, authoritative source for determining the full, correct set of instances for our three domains, we cannot report recall as an absolute percentage, and instead report just the number of distinct, correct instances found (as done by systems such as KnowItAll [24]). In addition, we must evaluate correctness by hand. To do this, we created a "gold standard" as follows: all system configurations produce the same basic set of candidate instances for a given ontology. A human evaluator (one of the authors) classified a random sample of 10% of this set (5% for Artists due to the much larger number of candidates found, see Table 2). For the random selection, each candidate was equally likely to be chosen, regardless of how many times it was extracted from the corpus. For each candidate, the evaluator chose the best, most specific ontology class available, while allowing for multiple senses. So a "Yo-Yo Ma" would be marked as a `Cellist`rather than the less specific `Musician`, while "Franz Liszt" would be marked as a `Composer`, `Pianist`, and `Conductor`. Two classes, `ExoticSpecies` and `IndigenousSpecies`, were removed from Animals because they were too subjective for a human to evaluate. To reduce bias, the evaluator had no knowledge of what class OntoSyphon assigned to each candidate nor OntoSyphon's assigned probability for that candidate.

Candidates with no correct class for that domain (e.g., "truck") were marked as incorrect, as were misspelled or incomplete terms (e.g., the artist "John"). To decide whether a candidate was a proper instance or a subclass, we assumed that the ontology was fairly complete and tried to follow the intent of the ontology. Candidates that could be properly classified as an instance of a leaf node in the ontology were treated as instances. Candidates that were already present in the ontology as a class or that seemed to be siblings of an existing class (e.g., the discovered "fiddler" and the existing class `Pianist`) were counted as incorrect. Finally, candidates that did not fit the intent of the ontology were marked incorrect. For instance, we considered the `Animal` ontology to be about types of animals (dogs and cats), so specific animals like "King Kong" or "Fido" were incorrect. Clearly, a knowledge base intended to model a set of specific animals would make different choices and instead treat discovered terms such as "dog" and "cat" as subclasses of `Mammal`. For simplicity of explication in this article, we adopt the completeness and intent conventions described above, enabling us to focus only on instances. Section 7 discusses this issue further.

This evaluation produced the function $gold_O(\,)$, where $gold_O(i)$ is the set of classes assigned to candidate instance *i* by the evaluator for ontology *O*. Then, given a candidate instance *i* and a class *c*, we define the pair (i, c) to be:

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

- **correct** if $c \in gold_O(i)$,
- **sensible** if $\exists c' \in gold_O(i)$ s.t. $c' \in subclasses(c)$,
- or **incorrect** otherwise.

For instance, if $gold_O(Yo-Yo\ Ma) = \{Cellist\}$, then (*Yo-Yo Ma,Cellist*) is correct, (*Yo-Yo Ma,Musician*) is sensible, and (*Yo-Yo Ma,Pianist*) is incorrect. Later, we will also need the following definition to identify instances for which there is a chance of picking an appropriate class in the ontology:

- An instance i is **plausible** iff $gold_O(i) \neq \emptyset$.

Let X be the output of the system for some experimental condition, where X consists of a set of pairs of the form (i, c), and where each candidate instance i appears in only one pair.[2] Then the *recall* is the number of pairs in X that are correct, and the *precision* is the fraction of pairs in X that are correct. These metrics are useful, but count only instances that were assigned to the most correct class possible, and thus do not fully reflect the informational content of the output. Consequently, we primarily report our results using the *sensible-recall*, which is the number of pairs in X that are sensible. In addition, we follow the example of several others in using *learning accuracy* (LA) in addition to exact precision. The LA measures how close each candidate pair (i, c) was to the gold standard $(i, gold_O(i))$. This measurement is averaged over all pairs to yield a precision-like number ranging from zero to one where $LA(X) = 1$ indicates that all candidate pairs were completely correct.

We follow the general definition of Learning Accuracy from Cimiano et al. [13], which requires the least common superconcept (lcs) of two classes a and b for ontology O [3]:

$$\text{lcs}_O(a, b) = \underset{c \in O}{\text{argmin}}(\delta(a, c) + \delta(b, c) + \delta(\text{top}, c)) \quad (1)$$

where $\delta(a, b)$ is the number of edges on the shortest path between a and b. Given this definition, the taxonomic similarity *Tsim* between two classes is

$$Tsim_O(d, e) = \frac{\delta(\text{top}, f) + 1}{\delta(\text{top}, f) + 1 + \delta(d, f) + \delta(e, f)} \quad (2)$$

where $f = \text{lcs}_O(d, e)$. We then define the average Learning Accuracy for ontology O of a set X of candidate pairs as

$$LA(X) = \frac{1}{|X|} \sum_{(i,c) \in X} \max(0, \max_{c' \in gold(i)} Tsim_O(c, c')) \quad (3)$$

5. Assessing candidate instances

Extracting candidate instances from the web is a noisy process. Incorrect instances may be extracted for many reasons including noun phrase segmentation errors, incorrect or incomplete sentence parsing, or factual errors in the web corpus. Because OntoSyphon operates in an unsupervised manner, it is thus critical to be able to automatically assign a probability or confidence value to the instances that are produced. These values can then be used to expunge instances that are below some confidence threshold and/or to provide reliability estimates to applications that later make use of the output data.

[2] OntoSyphon assigns only a single class to each instance, which is often sufficient but restrictive for domains like Artists. Future work should consider more general techniques.

[3] Maynard et al. [40] criticized Learning Accuracy (as presented by Hahn and Schnattinger [31]) in part because $LA(a, b)$ does not in general equal $LA(b, a)$. However, the revised formulation of Cimiano et al. [13], which we use, does not have this limitation. We also adopt Cimiano et al.'s formulation in order to be able to compare later with their results.

Table 3
Different metrics used for picking the best class and/or ranking candidate instances

Metric	Type	Description
Generic methods applied to either corpus		
Strength	Conf.	Total number of pattern hits for (i, c)
Str-INorm	Conf.	Strength normalized by instance frequency ($count(i)$)
Str-INorm-Thresh	Conf.	Like Str-INorm, but enforce minimum on $count(i)$
Str-CNorm	Conf.	Strength normalized by class frequency ($count(c)$)
Str-ICNorm-Thresh	Conf.	Str-INorm-Thresh plus class normalization
Methods applied only to $Corpus_S$		
Urns	Prob.	Balls and urns model learned via Expectation Maximization
Urns-INorm	Prob.	Urns with z parameter normalized by instance frequency
Urns-CNorm	Prob.	Urns with z parameter normalized by class frequency
Urns-ICNorm	Prob.	Urns with instance and class normalization
Methods applied only to $Corpus_L$		
PMI-Base	Prob.	Naive Bayes combination of binary PMI features, one model per class
PMI-Unify	Prob.	PMI-Base that constructs a single model based on class normalization
PMI-Unify-Resample	Prob.	PMI-Unify with seed re-sampling and threshold averaging

The second column indicates whether the metric produces a probability or a confidence value.

This section describes OntoSyphon's approach to automatic assessment. First, a confidence metric is computed for each candidate pair (i, c) using either $Corpus_S$ or $Corpus_L$ (Section 5.1). Second, the confidence metrics are used to select the best class c for each instance i where evidence for more than one class was found (Section 5.2). Finally, a new confidence metric is computed (using a technique from Section 5.1), and that metric is used to rank the final pairs. While some of the metrics of Section 5.1 are appropriate to both the class picking and instance ranking tasks, Section 6 shows that the best metrics are task-specific.

5.1. Computing confidence metrics

Below we describe the various assessment metrics that we consider in this article (Table 3 provides a summary). Each is used to assign a confidence score or probability to a candidate pair (i, c). We divide them into three categories: (1) "generic" metrics that are equally applicable with $Corpus_S$ or $Corpus_L$, (2) metrics only applicable with $Corpus_S$, and (3) metrics only applicable when using $Corpus_L$. After explaining the metrics, Section 5.1.4 examines their time/query complexity, and Section 5.1.5 discusses their novelty and why some are useful only in relation to one of the corpora.

In what follows, $count(i, c, p)$ is the number of hits for the pair (i, c) in the corpus with pattern p, and $count(y)$ is the number of hits for the term y alone in the corpus.

5.1.1. Generic assessment metrics

The following metrics can be usefully computed using any size corpus:

1. *Strength*: Intuitively, if the pair (*dog*, *Mammal*) is extracted many times from our corpus, this redundancy gives more confidence that that pair is correct. The Strength metric thus counts the number of times a candidate pair was observed across all extraction patterns P:

$$Score_{Strength}(i, c) = \sum_{p \in P} count(i, c, p) \quad (4)$$

Many systems (e.g., [13,60,50]) are based on this simple metric, though the counts are often obtained in a different manner. An interesting variation is that of Ponzetto and Strube [50], which contrasts the strength value with a parallel metric computed from patterns which indicate that the `is-a`relationship does *not* hold.

2. *Str-INorm*: The Strength metric is biased towards instances that appear very frequently in the corpus. To compensate, Str-INorm normalizes the pattern count by the number of hits for the instance alone:

$$Score_{Str-INorm}(i, c) = \frac{\sum_{p \in P} count(i, c, p)}{count(i)} \quad (5)$$

Similar normalization techniques are found in many systems (e.g., [24,14]).

3. *Str-INorm-Thresh*: The normalization performed by Str-INorm can be misleading when the candidate instance is a very rare term or a misspelling. Consequently, we created a modified Str-INorm where the normalization factor is constrained to have at least some minimum value. We found that a variety of such thresholds worked well. For this work we sort the instances by $count(i)$ and then select $Count_{25}$, the hit count that occurs at the 25th percentile:

$$Score_{Str-INorm-Thresh}(i, c) = \frac{\sum_{p \in P} count(i, c, p)}{\max(count(i), Count_{25})} \quad (6)$$

Pantel and Ravichandran [46] use, but do not separately evaluate, a much more complex "discounting factor" to account for such infrequent terms.

4. *Str-CNorm*: The raw Strength metric for a pair (i, c) is also biased towards pairs where the class c is more common in the corpus. To compensate, Str-CNorm normalizes the pattern count by the number of hits for the class alone:

$$Score_{Str-CNorm}(i, c) = \frac{\sum_{p \in P} count(i, c, p)}{count(c)} \quad (7)$$

Such normalization has been infrequently used in related work. One exception is Geleijnse et al. [28], who utilize a similar class-based normalization with their "PCM" metric, but do not specifically evaluate its performance.

5. *Str-ICNorm-Thresh*: Finally, it is natural to combine normalization for both the instance and the class together:

$$Score_{Str-ICNorm-Thresh}(i, c) = \frac{\sum_{p \in P} count(i, c, p)}{\max(count(i), Count_{25}) \cdot count(c)} \quad (8)$$

Note that thresholding on $count(c)$ is not as critical as with $count(i)$ because each class c is from the ontology, whereas each instance i is extracted from a more general (and possibly noisy) web document.

5.1.2. Metrics for use with $Corpus_S$

This section describes metrics that we only use in conjunction with the smaller $Corpus_S$.

6. *Urns*: OntoSyphon, like some other systems, extracts candidate facts by examining a large number of web pages (though we use the aforementioned Binding Engine to perform this process very efficiently over the smaller corpus). Prior work has developed the Urns model to apply in such cases [22,6] and has shown it to produce significantly more accurate probabilities than previous methods such as PMI (pointwise mutual information) or noisy-or. The Urns model treats each extraction event as a draw of a single labeled ball from one or more urns, with replacement. Each urn contains both correct labels (from the set C), and incorrect labels (from the set E); where each label may be repeated on a different number of balls. For instance, $num(C)$ is the multi-set giving the number of balls for each label $i \in C$. Urns is designed to answer the following question: given that a candidate i was extracted k times in a set of n draws from the urn (i.e., in n extractions from the corpus), what is the probability that i is a correct instance? For a single urn, if s is the total number of balls in the urn, then this probability is computed as follows:

$$P(i \in C | NumExtractions_i(n) = k) = \frac{\sum_{r \in num(C)} (r/s)^k (1 - (r/s))^{n-k}}{\sum_{r' \in num(C \cup E)} (r'/s)^k (1 - (r'/s))^{n-k}} \quad (9)$$

Urns operates in two phases. In the first phase, the set of all candidate extractions is used to estimate the needed model parameters ($num(C)$ and $num(C \cup E)$), using expectation maximization (EM). In particular, $num(C)$ is estimated by assuming that the frequency of correct extractions is Zipf-distributed, and then estimating the exponent z which parameterizes this distribution. In the second phase, a single pass is made over the extractions and each is assigned a probability using the estimated model parameters and an integral approximation to Eq. (9)[22].

Urns was designed to assign probabilities to a set of extractions that were targeting a single class, and the EM phase relies upon having a sufficient number of samples (roughly 500) for estimation. In our context, relatively few classes yield this many extractions on their own. We experimented with using Urns on these classes anyway, learning a separate model for each class but using a prior on the Urns parameters when the number of samples was low. We obtained better results, however, by learning a single model via combining the candidates from all of the classes of a single ontology together for the EM estimation. This approach yields more data to fit the model and ensures that any outliers influence the parameters for all models equally.

7. *Urns-INorm*: The Urns model, like Strength, does not exploit the frequency with which a candidate appears in the corpus. Instead, each instance is assumed to occur with equal probability anywhere along the aforementioned Zipf distribution. Introducing an appropriate prior probability of an input candidate's location along this curve would improve accuracy, but would significantly complicate the model and computation.

Fortunately, we can approximate the benefit of such a change with a much simpler approach. We begin by sorting the input data set X by $count(i)$. We then run EM both on the "lower" half of this data (which contains the less frequent terms), obtaining parameter z_L and on the whole data set, obtaining the aforementioned z. We then compute a parameter z_i for each instance i as

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

follows:

$$z_i = \max\left(z_L, z + \log(count(i)) - \sum_{(i',c')\in X} \frac{\log(count(i'))}{|X|}\right) \quad (10)$$

The probability for candidate (i, c) is then computed using z_i. Intuitively, the log functions increase z_i when a candidate i is more frequent than average, thus forcing i to have more pattern matches to obtain a high probability. On the other hand, the max function ensures that very infrequent words (particularly misspellings) do not obtain an artificially high probability, by insisting that the minimum z_i is a value appropriate for the "lower" half of the inputs (z_L).

8. *Urns-CNorm*: For a pair (i, c), Urns-INorm compensates for the frequency of i. We can adopt a similar strategy to compensate for the frequency of c. In particular, we can compute a parameter z_c for each class c as follows:

$$z_c = \max\left(z_L, z + \log(count(c)) - \sum_{(i',c')\in X} \frac{\log(count(c'))}{|X|}\right) \quad (11)$$

The probability for candidate (i, c) is then computed using z_c. We use the same z_L as above to ensure the final parameter is reasonable, and average over all pairs in set X (rather than over all classes) so that the average reflects the frequency of classes actually seen by the EM process.

9. *Urns-ICNorm*: Finally, we can combine instance and class normalization as follows:

$$z_{i,c} = \max\left(z_L, z + \log(count(i)) + \log(count(c)) - \sum_{(i',c')\in X} \frac{\log(count(i')) + \log(count(c'))}{|X|}\right) \quad (12)$$

The probability for candidate (i, c) is then computed using $z_{i,c}$.

5.1.3. Metrics for use with $Corpus_L$

This section describes metrics that we only use in conjunction with $Corpus_L$ (the whole web via a search engine).

10. *PMI-Base*: We start with the PMI-Bayes approach of KnowItAll [24], which was derived from Turney's PMI-IR algorithm [61]. Several papers [24,6] have found it to produce strong precision-recall curves, even if the Naive Bayes assumption used leads to polarized probability estimates. We summarize the algorithm below; see Etzioni et al. [24] for more details.

Let ϕ be the event that instance i is an instance of class c. The algorithm will compute $P(\phi)$ based on the value of k PMI scores, which are based on k different *discriminator* phrases (we use the same 5 patterns shown in Step 2 of Fig. 1). The jth score uses pattern p_j and is computed as follows:

$$\mathrm{PMI}(i, c, p_j) = \frac{count(i, c, p_j)}{count(i)} \quad (13)$$

$\mathrm{PMI}(i, c, p_j)$ is not a probability, but as with Strength, larger values should indicate that i *isa* c is more likely. We treat each score $\mathrm{PMI}(i, c, p_j)$ as a *feature* f_j that provides evidence for ϕ. By assuming independence of features, we can then compute the following probability:

$$P(\phi|f_1, f_2, \ldots f_k) = \frac{P(\phi)\prod_j P(f_j|\phi)}{P(\phi)\prod_j P(f_j|\phi) + P(\neg\phi)\prod_j P(f_j|\neg\phi)} \quad (14)$$

Here $P(\phi)$ is an estimated prior probability, $P(f_j|\phi)$ is the probability of observing feature j given ϕ is true, and $P(f_j|\neg\phi)$ is the probability of observing feature j given that ϕ is false. The key challenges with applying this formula are (1) estimating these probabilities from a small set of labeled examples and (2) producing such labeled examples in this unsupervised setting. Fortunately, Etzioni et al. found the general algorithm to be robust in the presence of some incorrect examples.

First, to make the probability estimation tractable, the algorithm transforms the continuous PMI scores into binary features, where f_j is true iff $\mathrm{PMI}(i, c, p_j) > T_j$ and T_j is a discriminator-specific threshold. For each discriminator j, the algorithm selects the T_j that that best separates a set of 10 positive and 10 negative examples ("seeds"). Then, it uses a second set of 10 positive and 10 negative seeds to estimate $P(f_j|\phi)$ by counting the number of positive seeds (plus a smoothing term) that are above the threshold. Likewise, counting the number of negative seeds yields $P(f_j|\neg\phi)$. Given these thresholds and conditional probabilities, Eq. (14) can be used to directly estimate the desired probability that i *isa* c.

Second, to automatically identify the needed positive and negative seeds, the algorithm randomly selects 100 candidate instances of class c. It ranks them by raw, average PMI score and selects the best 20 as positive seeds. To obtain negative seeds, we run OntoSyphon on all 3 ontologies simultaneously, and use the positive seeds of one domain as negative seeds for the others (Etzioni et al. use a similar approach; we likewise found it to be effective). The selected seeds are used to estimate $P(f_j|\phi)$ and $P(f_j|\neg\phi)$, then all 100 seeds are re-ranked using Eq. (14), and the best 20 seeds become the positive seeds for the next round.

With PMI-Base, we perform two such rounds of seed selection and probability estimation. For each class, we train a separate PMI model. In the original work by Etzioni et al., this bootstrap process also selected the five best discriminators among a larger set of choices. We found, however, that the discriminator selection was highly variable with some of the less frequent classes and thus adopted a fixed set of five discriminators. In our actual implementation, we adopt the suggestion of Etzioni et al. in using two thresholds per discriminator; this yields three different conditional probabilities for each feature and we found better handled low-frequency instances.

PMI-Base assigns higher probabilities to instances that satisfy (i.e., are above threshold for) more than one discriminator, and those that satisfy more reliable discriminators (i.e., those where $P(f_j|\phi)/P(f_j|\neg\phi)$ is greater). In this sense, it is somewhat similar to the instance ranking techniques used by Espresso [45] and Pronto [5], which incorporate per-pattern confidence scores. These systems, however, require manually provided training seeds.

11. *PMI-Unify*: PMI, as originally developed by KnowItAll, was intended to be used for ranking named entities that all belonged to the same class. In our multi-class context, this approach presents several problems. First, PMI needs a significant number (e.g., 100) possible instances of each class for its training, but many of the classes in our ontology do not produce that many. Second, the Naive Bayes classifier used by PMI is well known for producing very polarized probabilities (artificially close to zero or one). This has little practical impact when the same model is used to simply produce a relative ranking of entities, but may

produce mistakes when we must compare the probabilities of two different classes that were produced by two different PMI models.

PMI-Unify partially addresses both of these challenges. First, instead of training multiple models, it trains one model for the root class of the ontology. Then, to estimate probabilities for a single instance, we apply the model repeatedly, once for each class in the ontology, but where the PMI scores used have been normalized by the frequency with which each class/pattern combination appears alone. More specifically,

$$\mathrm{PMI}'(i, c, p) = \frac{count(i, c, p)}{count(i) \cdot count(*, c, p)} \quad (15)$$

where $count(*, c, p)$ represents the number of hits for class c with pattern p for any instance. This new normalization term is what enables us to meaningfully apply a model learned for one class to other classes as well (cf., the similar PMI normalization of Espresso [45]). Using a single model provides more consistency across the predictions for different classes, and eliminates the need to have sufficient seeds from each class for training.

12. *PMI-Unify-Resample*: While PMI-Unify partially addresses the challenges of applying PMI to a multi-class ontology, challenges remain. In particular, PMI's bootstrap process is somewhat problematic, because it specifically tries to identify the *most likely* terms as seeds. This goal is important, since the probability learning requires meaningful seeds to operate. However, this goal exacerbates the problem of biased probabilities, since the final set of seeds tends to be terms that have evidence from many different patterns, causing the model to sharply reduce the probability of any tested term that does *not* have such strong evidence, even if it is in fact correct.

PMI-Unify-Resample addresses this challenge with two enhancements. First, we add a third and fourth bootstrapping round. During these rounds, instead of always picking the N most probable terms as the next seeds, the process chooses $N/2$ seeds from amongst the top ranked terms, and the remaining seeds from among the terms with estimated probabilities in the range of 0.5–0.7. In this way, general parameters are established in rounds one and two, but rounds three and four ensure that the final probability estimates are based on a more realistic distribution of instances.

Second, we found that even with these changes PMI was very sensitive to the particular set of seeds that were used. Since PMI uses thresholds to enable tractable learning, small changes in the threshold can cause the probability of particular instances to fluctuate substantially. To address this problem, PMI-Unify-Resample also modifies all rounds of the bootstrap phase so that, after each discriminator estimates its optimal threshold value, the algorithm replaces each threshold with the average threshold across all discriminators. Essentially, this change limits the power of the model in order to reduce the impact of the particular set of seeds on the final results.

5.1.4. Time/query complexity

The dominant cost for all of the above metrics is the time to query the corpus for the raw statistics, e.g., $count(i, c, p)$. To analyze this cost, we assume below that there are N_I instances found, N_C classes in the ontology, N_P patterns used (in this article, $N_\mathrm{P} = 5$), and N_D documents in the corpus.

The statistics can be computed in two different manners. First, we can make a single pass over the entire corpus, in time $\mathcal{O}(N_\mathrm{D})$. We use this approach for all $Corpus_S$-based metrics, and with the aforementioned Binding Engine [7] this process requires about a minute. Second, the statistics can be computed by querying a search engine for each instantiated pattern string, and obtaining the resulting counts. This is our approach for all $Corpus_L$-based metrics, and it requires $\mathcal{O}(N_\mathrm{I} N_\mathrm{C} N_\mathrm{K})$ queries.

Many optimizations, however, are possible for reducing the actual number of queries needed. For instance, for Strength (but not for PMI), per-pattern statistics are not needed (only $\sum_{p \in P} count(i, c, p)$), so multiple queries for the same (i, c) pair can be combined with a boolean OR query. Second, many queries can be eliminated by observing that the query "pianists such as Beethoven" is redundant if the queries "pianists such as" or "such as Beethoven" return no hits. Thus with minimal effort, we reduced the actual number of queries needed for PMI with our ontologies to about 60–70 queries per instance, with further gains possible. This represents an improvement over the 140–195 queries that were theoretically required, but remains a substantial cost. The cost is especially an issue because commercial search engines impose hard limits on the number of queries that can be programmatically issued per day. More drastic reductions could be obtained by issuing generic queries like "such as Beethoven," then examining the limited number of search snippets returned to find possible classes for "Beethoven" (similar to C-PANKOW [13]). We instead compute the full statistics in order to gauge the potential improvement of using the larger $Corpus_L$ vs. $Corpus_S$. Some related work [6,4] has also considered the relative impact of learning structured data from a larger corpus.

5.1.5. Discussion

Urns, Strength, Str-INorm, and (at least parts of) Str-CNorm have been used in some form in other work, though Urns required some adaptation for our multi-class problem. However, Str-INorm-Thresh, Urns-INorm, and Urns-CNorm are novel contributions of this work that we found to be very effective compared to their simpler variants (e.g., vs. Strength and Urns, see Section 6), and that should also provide significant improvements in other systems. Likewise, PMI-Base has been used extensively by KnowItAll, but later sections will show that our enhancements with PMI-Unify and PMI-Unify-Resample can substantially improve performance for the more challenging multi-class problem that we face.

Not all metrics are profitable to use with either $Corpus_S$ or $Corpus_L$. For instance, we could also apply PMI to $Corpus_S$. However, to work well PMI requires that true instances have strong (above threshold) evidence from multiple patterns. We found that even with $Corpus_L$ this was a challenge, because many correct instances appear predominantly in just one or two of the considered patterns. Using a smaller corpus would greatly exacerbate this problem and make training the classifier probabilities much less reliable. Hence, we only evaluated PMI with the larger corpus.

Likewise, we experimentally verified that applying Urns to our results, with counts computed from $Corpus_L$, produced very poor results. The problem with this approach is that it violates a key assumption used by the Urns EM process to learn the model. In particular, the EM process assumes that its input is the set of *a* ll candidates that were found in *n* extraction events. This requirement is satisfied when the input statistics are computed by repeatedly issuing web queries that return instances of a particular class (e.g., query for "animals such as . . .", as in the original application of Urns [22]), or by extracting all such matches from a set of documents (as we do for $Corpus_S$). However, this requirement is *not* satisfied by finding new counts for an existing set of candidates (e.g., counting hits for "animals such as tigers"). The problem with the latter approach is that the resulting input to Urns does not accurately

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

reflect the distribution of instances in the larger corpus. In particular, true but infrequent instances are very underrepresented. Consequently, the probabilities produced by Urns heavily penalize any instance that does not have many hits, resulting in many false negatives. In theory, a model learned over $Corpus_S$ could, with suitable parameter adjustments, be applied over $Corpus_L$. We found that straight-forward parameter adjustments did not work well, but believe this is a tractable direction for future research.

5.2. Picking the best class

Given a set X of candidate instances, and a metric value computed for each via one of the metrics described above, OntoSyphon must next select the best class c for each instance i for which evidence for more than one class was found. This is shown in step 7 of Fig. 1. That example includes the following evidence (where we start to substitute i for the instance *wildebeest*):

$$Score(wildebeest, Animal) = Score(i, Animal) = 0.91$$
$$Score(wildebeest, Mammal) = Score(i, Mammal) = 0.85$$

Recall that OntoSyphon always chooses a single class for each instance i. We consider several techniques for making this choice:

(1) PickMax: The simplest case is to pick the class with the maximal confidence score:

$$Class_{PickMax}(i) = argmax_{c \in O} Score(i, c) \quad (16)$$

In the above example, this selects *Animal*—a "sensible" choice, but not fully correct (the LA is 0.67). This approach has been the most commonly used in previous work [11,13,60,28].

(2) TreeAscent: Given a new concept t and scores representing how similar t is to each concept in a hierarchy, Maedche et al. [38] previously proposed the use of the TreeAscent algorithm for ontology learning. Adapted to instance classification, this algorithm would compute, for every class c, the sum of the instance's score with respect to every class c', weighted by the taxonomic similarity (see Section 4) between c and c':

$$Class_{TreeAscent}(i) = argmax_{c \in O} \sum_{c' \in O} Score(i, c') Tsim_O(c, c') \quad (17)$$

The idea is that significant evidence in the classes "nearby" class c might argue for choosing c, even if another class had a higher raw score. Frommholz [25] utilized a very similar idea, but where estimated class subsumption probabilities were used instead of taxonomic similarity, with some limited success.

In the above example, the TreeAscent metric would still choose *Animal* for *wildebeest*. Suppose, however, that there was also the (in this case incorrect) evidence $Score(wildebeest, Human) = 0.52$. This new evidence for *Human* (as a subclass of Mammal) would provide enough evidence such that TreeAscent would choose *Mammal* (an LA of 1.0). Such a technique may not maximize precision overall, but aims to maximize the LA of the final result.

(3) PickMax-StructAdjust: In the first example, PickMax's naive choice of the maximal score produced a sub-optimal result. However, the structure of the ontology (in particular, the subclass hierarchy) suggests a better approach. To simplify the discussion, let us introduce the events A and M as follows:

$$Score(i, Animal) = P(i \, isa \, Animal) = P(A) = 0.91$$
$$Score(i, Mammal) = P(i \, isa \, Mammal) = P(M) = 0.85$$

The ontology, however, states that $Mammal \subset Animal$, which implies that event A must be true if event M is true. Thus, to pick the best, most specific class, we really must compare $P(M)$ vs. $P(A \wedge \neg M)$. This can be computed as follows:

$$P(A) = P(A \wedge M) + P(A \wedge \neg M),$$
$$P(A) = P(M) + P(A \wedge \neg M),$$
$$P(A \wedge \neg M) = P(A) - P(M),$$
$$P(A \wedge \neg M) = 0.91 - 0.85,$$
$$P(A \wedge \neg M) = 0.06$$

Since $P(A \wedge \neg M) = 0.06 < P(M) = 0.85$, we should choose *Mammal* as the most specific class for this instance. Intuitively, since the probabilities for *Animal* and *M* ammal are very close, it is probable that the subclass (*M* ammal) is the most correct choice. We can generalize the above approach to compute a modified score for any class c with a single child c':

$$Score'(i, c) = P(i \, isa \, c \wedge \neg(i \, isa \, c')) \quad (18)$$

$$Score'(i, c) = Score(i, c) - Score(i, c') \quad (19)$$

Moreover, if we assume that siblings in the subclass hierarchy are disjoint classes, then similar algebra can be used to generalize this approach to deal with situations where there is evidence for more than one child of c:

$$Score''(i, c) = P(i \, isa \, c \wedge \neg(i \, isa \, c' | c' \in child(c))) \quad (20)$$

$$Score''(i, c) = Score(i, c) - \sum_{c' \in child(c)} Score(i, c') \quad (21)$$

This equation, however, has several practical flaws. First, if the sum of the children's scores is greater than $Score(i, c)$ (either because the estimated scores are biased or because the children are not in fact disjoint), then $Score''(i, c)$ will be less than zero. Second, while in theory it should hold $P(i, c') < P(i, c)$ if c' is a child of c, this will not always be true for the estimated scores, leading to incorrect conclusions. For instance, consider the following evidence, where each class is a child of the one above it:

$$Score(black \, widow \, spider, Animal) = 0.97,$$
$$Score(black \, widow \, spider, Arthropod) = 0.01,$$
$$Score(black \, widow \, spider, Arachnid) = 0.95$$

Using Eq. (21) on this data would yield adjusted probabilities of (0.96, −0.94, and 0.95) respectively. Even if the scores are constrained to be positive, clearly the probability related to *Animal* needs to be adjusted based not just on the score at *Arthropod*, but with the score for *Arachnid* (a descendant of *Animal*, but not a child).

We thus introduce the notion of "structure adjusted" scores, which can be computed as follows:

$$M(i, c) = \max\left(Score(i, c), \max_{c' \in child(c)} M(i, c')\right) \quad (22)$$

$$Score_{StructAdjust}(i, c) = \max\left(0, Score(i, c) - \sum_{c' \in child(c)} M(i, c')\right) \quad (23)$$

The first equation adjusts for children that may have higher scores than their parents, while the second equation performs the basic transformation of Eq. (21) while pragmatically adjusting negative probabilities that can arise in practice. Note that the first equation requires traversing the tree in postfix order

Table 4
Class picking results for Animals domain

Basic method	PickMax				TreeAscent			
	Without StructAdjust		With StructAdjust		Without StructAdjust		With StructAdjust	
	LA	Prec	LA	Prec	LA	Prec	LA	Prec
Sml-Strength	0.85	57.9%	0.89	69.4%	0.85	59.5%	0.88	67.8%
Sml-Str-CNorm	0.91	76.0%	0.91	76.9%	0.91	76.0%	0.91	76.9%
Sml-Urns	0.85	57.9%	0.91	76.9%	0.83	52.9%	0.89	71.9%
Sml-Urns-INorm	0.85	57.9%	0.90	73.6%	0.83	52.9%	0.89	70.2%
Sml-Urns-CNorm	**0.92**	**80.2%**	**0.92**	**80.2%**	0.89	72.7%	0.90	75.2%
Lrg-Strength	0.85	60.3%	0.86	65.3%	0.85	58.7%	0.86	64.5%
Lrg-Str-CNorm	0.89	72.7%	0.89	75.2%	0.89	73.6%	0.89	75.2%
Lrg-PMI-Base	0.85	60.3%	0.86	62.0%	0.81	47.9%	0.85	57.0%
Lrg-PMI-Unify	0.85	58.7%	0.91	77.7%	0.80	43.0%	0.88	67.8%
Lrg-PMI-Unify-Resample	0.84	54.5%	**0.93**	**81.8%**	0.79	42.1%	0.89	71.1%

(children before parents). Using these formulas, the preceding example would yield adjusted scores of (0.02, 0.00, and 0.95) respectively. This is to be interpreted as "the estimated probability that *i* is an Animal and *not* an *A* rthropod or *Arachnid* is 0.02." Picking the maximal such score yields *Arachnid*, the most correct choice of the three.

Thus we arrive at the final form for PickMax-StructAdjust:

$$Class(i) = arg\max_{c \in O} Score_{StructAdjust}(i, c) \quad (24)$$

(4) TreeAscent-StructAdjust: To optimize LA instead of precision, it makes sense to utilize the above-mentioned adjusted probabilities with TreeAscent:

$$Class_{TreeAscent-StructAdjust}(i) = arg\max_{c \in O} \sum_{c' \in O} Score_{StructAdjust}(i, c') Tsim_O(c, c') \quad (25)$$

In hindsight, the transformation of probabilities based on StructAdjust seems to be a clear advantage provided that (1) the probabilities are not too biased and (2) the sibling classes are mostly disjoint. However, we are not aware of any previous system that has employed such an approach for instance or concept classification. In fact, many systems that classify entities of some kind into a class hierarchy actually treat the possible classes as a flat set for classification purposes [11,17,60]. Among systems that do attempt to use the hierarchical structure for classification, many systems (e.g., Dumais and Chen [23], the "tree descent" of Maedche et al. [38]) follow the early example of Koller and Sahami [35] and use a top-down approach that only classifies entities into "leaf" classes. Such an approach is not appropriate for many domains, including the ones we investigate, where the best class for a particular instance may be an internal node. One recent exception is that of Li and Bontcheva [37], which extends the Hieron algorithm to classify terms into an ontology. Notably, Hieron does not make decisions based on per-class prototypes, but on the difference between the prototype of a node and its parent. While this technique is motivated very differently, it resembles some of the parent/child adjustments made by StructAdjust.

Note that even when the disjointness assumption is violated, StructAdjustmay nonetheless work well in practice. Furthermore, while we have derived StructAdjust with probabilities, the derivation of Eq. (21) and hence the use of StructAdjust applies equally well to linear confidence scores. Section 6.1 evaluates these issues experimentally.

6. Experimental evaluation

In this section we experimentally evaluate OntoSyphon. We first consider the impact of the class picking technique, then evaluate instance ranking. Finally, we evaluate the impact of using a smaller search corpus.

6.1. Instance classification results

After extracting candidate instances, OntoSyphon should pick the best class for each candidate. For instance, should the final output include (*tiger*, *Animal*) or (*tiger*, *Mammal*)?

Tables 4–6 contain the results for our three ontologies. The values in each cell report the LA and the exact precision for the set produced by some instance classification technique. Each row corresponds to a different confidence metric, with the $Corpus_S$-based

Table 5
Class picking results for Food domain

Basic method	PickMax				TreeAscent			
	Without StructAdjust		With StructAdjust		Without StructAdjust		With StructAdjust	
	LA	Prec	LA	Prec	LA	Prec	LA	Prec
Sml-Strength	0.86	55.1%	0.87	55.1%	0.86	55.1%	0.88	57.1%
Sml-Str-CNorm	**0.89**	**63.3%**	**0.89**	**63.3%**	**0.89**	**63.3%**	**0.89**	**63.3%**
Sml-Urns	0.86	55.1%	0.87	57.1%	0.86	53.1%	0.87	59.2%
Sml-Urns-INorm	0.86	55.1%	0.87	59.2%	0.86	55.1%	0.87	59.2%
Sml-Urns-CNorm	0.88	57.1%	0.87	59.2%	0.87	55.1%	0.88	61.2%
Lrg-Strength	0.86	55.1%	0.86	55.1%	0.86	55.1%	0.86	55.1%
Lrg-Str-CNorm	0.84	49.0%	0.80	38.8%	0.84	49.0%	0.81	40.8%
Lrg-PMI-Base	**0.85**	**57.1%**	0.83	51.0%	**0.85**	**57.1%**	0.83	51.0%
Lrg-PMI-Unify	0.85	55.1%	0.81	44.9%	0.84	51.0%	0.81	44.9%
Lrg-PMI-Unify-Resample	0.84	55.1%	0.82	51.0%	0.85	55.1%	0.84	53.1%

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

Table 6
Class picking results for Artists domain

Basic method	PickMax				TreeAscent			
	Without StructAdjust		With StructAdjust		Without StructAdjust		With StructAdjust	
	LA	Prec	LA	Prec	LA	Prec	LA	Prec
Sml-Strength	0.73	30.2%	0.74	30.7%	0.73	30.2%	0.73	30.2%
Sml-Str-CNorm	0.75	33.3%	0.75	33.3%	0.75	32.8%	0.75	33.3%
Sml-Urns	0.73	30.2%	**0.76**	**35.4%**	0.74	30.2%	**0.76**	**35.9%**
Sml-Urns-INorm	0.72	29.2%	0.74	30.7%	0.73	29.7%	0.74	30.7%
Sml-Urns-CNorm	0.75	32.3%	**0.76**	**34.9%**	0.75	31.3%	**0.76**	**34.9%**
Lrg-Strength	0.73	30.7%	0.74	33.3%	0.74	30.7%	0.74	33.9%
Lrg-Str-CNorm	0.79	43.8%	0.80	46.9%	0.79	43.8%	0.80	46.9%
Lrg-PMI-Base	0.73	31.8%	0.78	44.3%	0.75	28.6%	0.79	43.8%
Lrg-PMI-Unify	0.73	31.3%	**0.80**	**50.5%**	0.75	27.1%	0.81	49.5%
Lrg-PMI-Unify-Resample	0.73	31.3%	0.79	49.5%	0.74	25.5%	**0.81**	**51.0%**

metrics up top and the $Corpus_L$-based metrics below.[4] Each column corresponds to a different strategy for selecting the best class based upon that metric. The first two columns use PickMax, with and without StructAdjust, while the last two columns likewise use variants of TreeAscent. For instance, for Animals when the best class is selected via PickMax over structure-adjusted Sml-Urns statistics, the resulting set has an LA of 0.91 and a precision of 76.9% (e.g., 76.9% of the instances are assigned the same class as in the gold standard). Bold entries show results where the precision was within 1% of the maximum found within the same $Corpus_S$ or $Corpus_L$ parts of the table. Note that class picking techniques are written in small caps (e.g., PickMax), while metric types are written in a regular font (e.g., Str-INorm).

In order to show the differences more clearly, the results in this section evaluate only those instances that are plausible (see Section 4) and where there was some ambiguity regarding which class to pick, i.e., where considering all metrics across both corpora, there was evidence for more than one class for a particular instance. This represents 20% of the total candidates found for Food and Artists, and 14% for Food (a large fraction of candidates are not plausible because even a single incorrect extraction produces a candidate that is considered). The next section instead evaluates overall performance. Thus, this section examines the important problem of maximizing the precision of plausible candidates (our ultimate goal for instance population), while the next section considers all candidate results together in order to evaluate overall accuracy.

Table 7 shows the results averaged across the three domains. This table highlights four general results:

1. *Using* StructAdjust*was almost always helpful*. Averaging across all rows of Table 7, this transformation improves precision by 4.9 percentage points for PickMaxand 6.5 points for TreeAscent. For Animals and Artists, StructAdjustalways improves upon or equals the performance of not using StructAdjust.[5] In some cases, StructAdjust has a large impact. For instance, Lrg-PMI-Unify-Resample with PickMax improves from 54.5% precision to 81.8% for Animals and from 31.3% to 49.5% for Artists. Large gains are also found with Urns. Likewise, Food with $Corpus_S$ shows consistent but small gains. The only exception is for Food when using $Corpus_L$. In general, for Food using $Corpus_S$ is superior to using $Corpus_L$ (see Section 7), and in this latter case the transformation is less useful.

   In contrast, the choice of PickMax vs. TreeAscent mattered less. In general, using PickMax almost always improved precision, without decreasing LA. This effect is to be expected, though notably in some cases precision improves by as much as 10 points. Averaging across all rows of Table 7, using PickMaximproved over TreeAscentby 2.7 points when StructAdjust was not used but by just 1.0 points when StructAdjust was applied.
2. *Class normalization showed a small but highly consistent gain*. For instance, averaging across the columns of Table 7, Sml-Str-CNorm outperforms Sml-Strength by 7.8 points, Sml-Urns-CNorm outperforms Sml-Urns by 4.9 points, Lrg-Str-CNorm outperforms Lrg-Strength by 4.8 points, and Lrg-PMI-Unify outperforms Lrg-PMI-Base by 0.8 points. These gains are consistent with our intuition: when trying to pick the best class, it pays to normalize by the class frequency. Table 7 shows one exception to these results: with PMI and non-StructAdjust, class normalization decreases precision by 1.4 points with PickMax and 4.2 points with TreeAscent. However, when StructAdjust is applied, normalization instead increases precision by 5.3 points with PickMax and 3.5 points with TreeAscent. This illustrates the general pattern that StructAdjust and class normalization are synergistic: normalization yields more representative per-class scores, which StructAdjust then uses to more accurately pick the most precise class. Indeed, for each domain using normalization and transformation together almost always yields maximal results compared to using neither or just one (again, Food with $Corpus_L$ is an exception).
3. *The proposed enhancements to PMI demonstrated gains, especially when combined with* StructAdjust. In that case, normalization with PMI-Unify improves precision vs. PMI-Base, as discussed above. Furthermore, the addition of seed re-sampling and threshold averaging with PMI-Unify-Resample further improves average precision by 3.1 points for PickMax and 4.4 points for TreeAscent. These two enhancements show consistent improvement for Animals and Artists, with gains as large as 19.8 points and 7.2 points, respectively. For Food, PMI-Unify somewhat decreases performance compared to PMI-Base (since class normalization was not helpful for Food with $Corpus_L$), though the enhancements of PMI-Unify-Resample recovers most of the losses.
4. *For every domain there is substantial opportunity to improve precision over the less sophisticated methods*. The best technique depends upon the domain, but to gauge the magnitude of the possible improvement, we compare the precision of the best

[4] We omit results for Str-INorm and variants because, while instance normalization affects other results, it has no effect on the relative strength-based scores of different classes for the same instance *i*.

[5] Note that this is true for Artists even though this domain does not strictly satisfy the sibling class disjointness assumption. This is likely due both to the practical adjustments of Eqs. (22) and (23) to handle imperfect probabilities, as well as the fact that we seek to identify just the single best class, not all appropriate ones.

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

Table 7
Class picking results averaged over the three domains

Basic method	PickMax				TreeAscent			
	Without StructAdjust		With StructAdjust		Without StructAdjust		With StructAdjust	
	LA	Prec	LA	Prec	LA	Prec	LA	Prec
Sml-Strength	0.81	47.7%	0.83	51.8%	0.82	48.3%	0.83	51.7%
Sml-Str-CNorm	**0.85**	**57.5%**	**0.85**	**57.8%**	**0.85**	**57.4%**	**0.85**	**57.8%**
Sml-Urns	0.81	47.7%	0.84	56.5%	0.81	45.4%	0.84	55.7%
Sml-Urns-INorm	0.81	47.4%	0.84	54.5%	0.81	45.9%	0.83	53.4%
Sml-Urns-CNorm	0.85	56.5%	**0.85**	**58.1%**	0.84	53.0%	**0.85**	**57.1%**
Lrg-Strength	0.81	48.7%	0.82	51.2%	0.81	48.2%	0.82	51.1%
Lrg-Str-CNorm	0.84	55.2%	0.83	53.6%	0.84	55.4%	0.84	54.3%
Lrg-PMI-Base	0.81	49.7%	0.83	52.4%	0.80	44.6%	0.82	50.6%
Lrg-PMI-Unify	0.81	48.3%	0.84	57.7%	0.80	40.4%	0.83	54.0%
Lrg-PMI-Unify-Resample	0.80	47.0%	**0.85**	**60.8%**	0.79	40.9%	0.84	58.4%

simple technique (one that does no class normalization and does not use StructAdjust) vs. the best method enabled by the new techniques described in this article (using class normalization and StructAdjust). We measure the statistical significance of the gain using a one-tailed Chi-square test, as also used in Maedche et al.'s [38] original examination of TreeAscent, at the 5% significance level. Both Animals and Artists show statistically significant gains: for Animals, 60.3% vs. 81.8% (a gain of 21.5 points, $p = 0.0002$) and for Artists, 31.8% vs. 49.5% (a gain of 17.7 points, $p = 0.0004$). For Food, there are also gains, but they are not statistically significant: 57.1% vs. 63.3% (a gain of 6.2 points, $p = 0.535$). Averaging all of the "simpler" techniques across the 3 domains vs. the average of the techniques with class normalization and StructAdjust shows a gain from 47.5% to 57.2% precision and from 0.811 to 0.844 LA. These results demonstrate that different class picking strategies have very different performance, particularly if precision is the primary concern.

Since OntoSyphon is an unsupervised system, we would ideally like to identify a single class selection strategy that will perform well across many domains. Considering the average results, Sml-Urns-CNorm with PickMax+StructAdjust performed best amongst the $Corpus_S$-based statistics, and always had reasonable performance (Sml-Str-CNorm would also be a strong choice). Amongst the $Corpus_L$-based methods, Lrg-PMI-Unify-Resample with PickMax+StructAdjust performed the best on average and was also fairly consistent. Thus, in the remainder of this article we report results using these two class selection algorithms.

6.2. Instance ranking results

Fig. 2(a)–(f) shows the results of executing OntoSyphon on our sample ontologies. The left column of figures show results using $Corpus_S$-based metrics, while the right column show results using $Corpus_L$-based metrics. Class picking used PickMax+StructAdjust with Sml-Urns-CNorm for the figures on the left and Lrg-PMI-Unify-Resample for figures on the right.[6] Each line represents one output of the system using a particular assessment technique. To create one point on the line, we chose a threshold and then removed from the output set all candidate pairs whose assigned confidence values were below that threshold. The *x*-axis measures the sensible-recall of this modified set, and the *y*-axis shows the LA of this set. Varying the threshold thus produces a line with properties similar to a classical precision/recall tradeoff curve. For comparison, Fig. 3 shows similar graphs but where the *y*-axis is exact precision.

The data demonstrates that OntoSyphon was able to find a substantial number of sensible instances for all three domains. In addition, the data shows that some of our tested assessment techniques are quite effective at identifying the more reliable instances. In particular, the techniques that perform instance normalization (Str-INorm, Str-INorm-Thresh, and Urns-INorm) show consistent improvements over the techniques that do not (Strength and Urns). Consider, for instance, the "50% recall" point, where each technique has a sensible-recall equal to half of the maximum achieved under any situation (e.g., where sensible-recall equals 950 for Animals). At this point, Sml-Urns-INorm and Sml-Str-INorm-Thresh increase LA compared to the no-normalization techniques by 0.18–0.25 (41–59%) for Animals, 0.20–0.21 (47–48%) for Food, and 0.06–0.07 (15–17%) for Artists. Lrg-Str-INorm-Thresh shows similar gains vs. Lrg-Strength.

Overall, both Urns-INorm and Str-INorm-Thresh perform consistently well. Str-INorm also performs well, except that it has many false positives at low recall. With both corpora, it gets fooled by many incorrect terms that are very infrequent on the web, and thus have a high score after normalization, even though they were extracted only once or a few times. For instance, Str-INorm incorrectly gives a high score to the misspelled terms "mosquities" for Animals, "Andrea Cotez" for Artists, and "pototato" for Food.

For Animals and Food, the results were similar, with somewhat higher LA for Animals, particularly with $Corpus_L$. For Artists, OntoSyphon found substantially more instances. However, our assessment techniques worked least well on this domain, showing a fairly flat curve. One reason is that the assessment techniques are fooled by a large number of spurious candidates that come from one ambiguous class, `Players`. This class (intended for musical instrument players) finds mostly sports team participants (e.g. "Greg Maddux") and a few digital music software products ("WinAmp"). Fig. 2(g) and (h) shows the results if this class is removed from OntoSyphon's search process (Section 7 describes how this could be done automatically). With Str-INorm-Thresh, LA increases at 50%

[6] To make the figures more readable we omit results that combine a ranking metric from one corpus with a class picking metric from another. We found that switching between the best class picking methods from each corpus had minimal effect on the overall Learning Accuracy (recall that for most instances there is no class picking decision to be made), but a small impact on *precision* for Food and Artists. For instance, for Lrg-Str-INorm-Thresh, changing to the $Corpus_S$-based class picking reduced precision, at 50% recall, from 41% to 36% for Artists and increased precision from 44% to 48% for Food (consistent with the trends in the previous section). Likewise, we omit results for Strength and Urns with class normalization. Results showed that, while class normalization is a very significant factor for choosing the best class for a single instance, it had minimal impact on the results when used for ranking multiple instances.

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

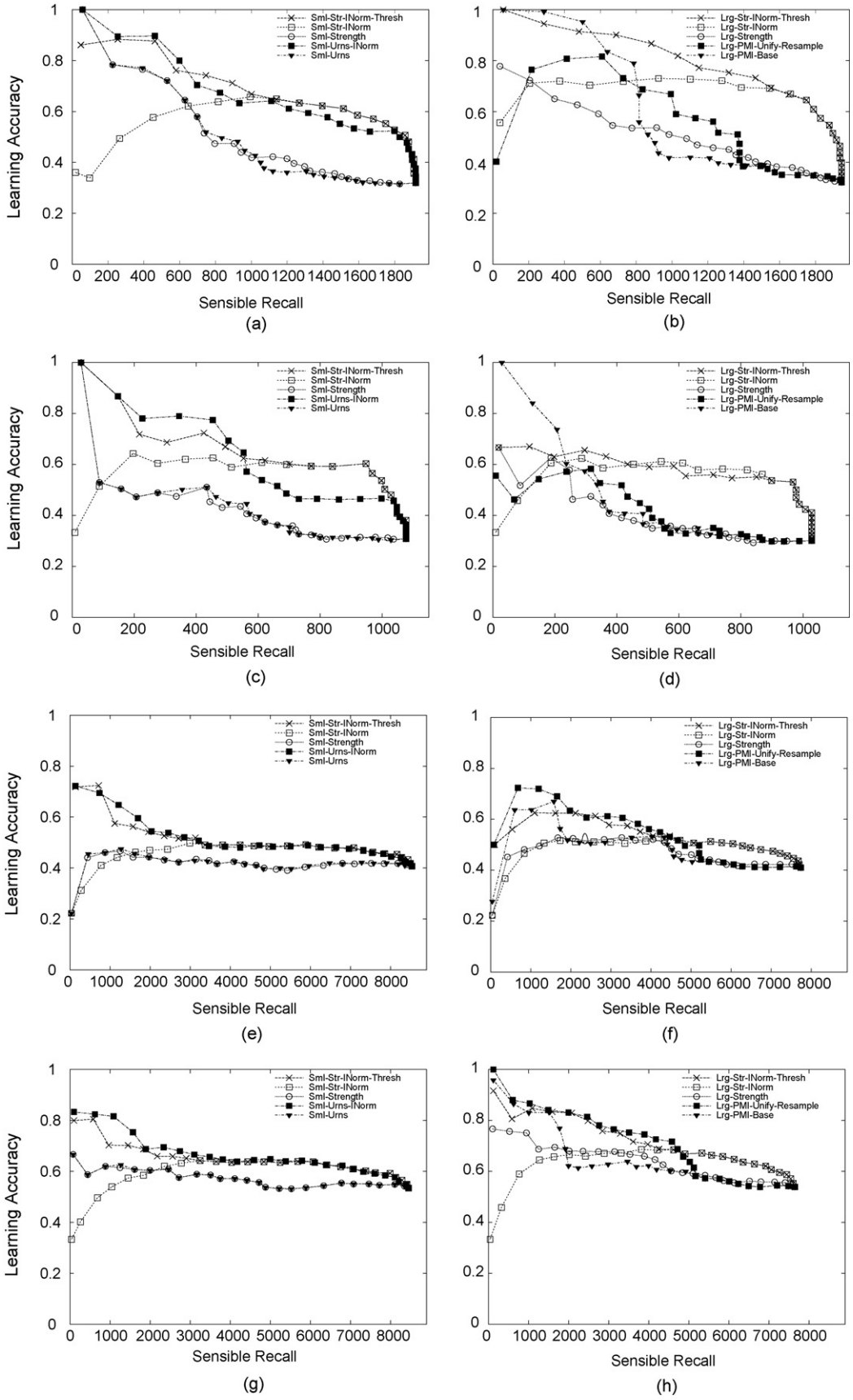

Fig. 2. Results (Learning Accuracy) for each domain. (a and b) Animal domain, (c and d) Food domain, (e and f) Music domain, and (g and h) Music (without `Player`).

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

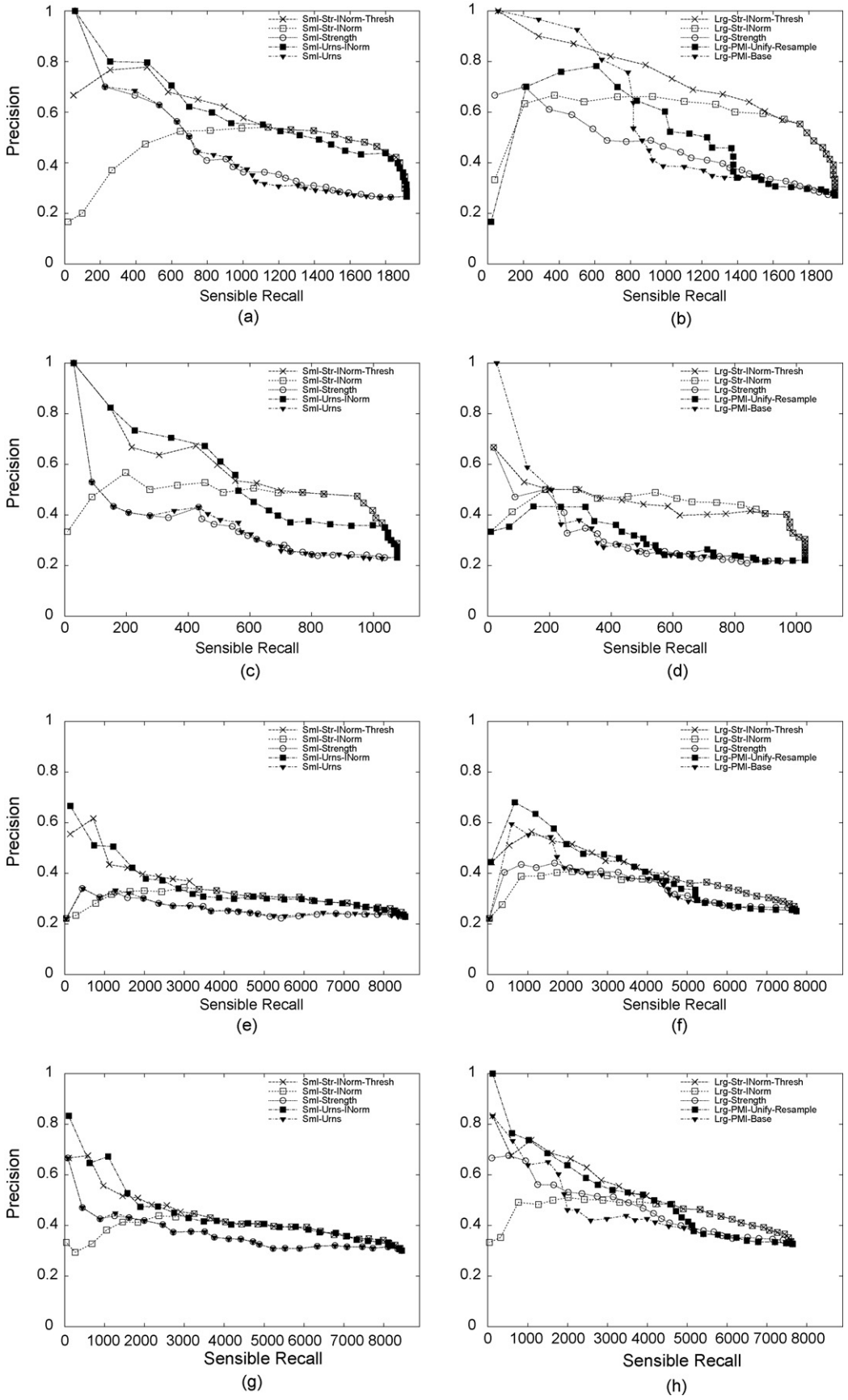

Fig. 3. Results (precision) for each domain. (a and b) Animal domain, (c and d) Food domain, (e and f) Music domain, and (g and h) Music (without `Player`).

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

recall from 0.49 to 0.64 with $Corpus_S$ and from 0.54 to 0.71 with $Corpus_L$.

A more fundamental problem is the smaller amount of redundancy in the web for the Artists domain. For instance, with $Corpus_S$ a sensible instance for Animals is extracted by OntoSyphon's patterns on average 13.7 times vs. 2.3 times for an incorrect instance (16.3 vs. 3.7 hits for Food). However, Artists, even with `Player` removed, averages only 2.7 hits for sensible instances vs. 2.2 hits for incorrect instances. This smaller split yields a much more difficult assessment task, and additional work is needed to more fully exploit the potential of such domains.

The PMI-based metrics perform surprisingly poorly. In general, PMI-Base does better at low recall/high precision (since its multiple thresholds provide the most discriminative power), while PMI-Unify-Resample does better at moderate recall. At high recall, however, both PMI-based metrics perform very poorly. Examination of the results revealed that this poor performance resulted from PMI's inability to handle low-frequency instances. In particular, many such instances passed the threshold for only one or no discriminators, yielding many instances at a small number of probability levels. This was often adequate for picking the best class within a single instance, but not very helpful for separating out the most likely instances. Additional thresholds or discarding thresholds entirely in favor of continuous functions could provide additional fidelity but present significant training challenges. Alternatively, we could exploit the observation of Etzioni et al. [24] that an incorrect instance is much less likely to pass a discriminator test based on a particular class *and* a discriminator test based on a synonym of that class (e.g. "town" and "city"). They used hand-selected synonyms; for our more automated system future work is needed to learn synonymous words or phrases for the classes of an arbitrary ontology.

Finally, we compare $Corpus_S$-based metrics vs. $Corpus_L$-based metrics. The relative performance of Strength vs. Str-INorm vs. Str-INorm-Thresh is remarkably consistent across the two corpora. In both cases, Str-INorm-Thresh is a top overall performer. For Food and Artists, using $Corpus_L$ improves LA, e.g. by 0.15 for Animals and 0.05 for Artists at 50% recall. With Food, using $Corpus_L$ slightly decreases LA by 0.04, consistent with the trends in the previous section.

6.3. Results with varying corpus size

The previous section considered results where candidate instances were extracted by searching $Corpus_S$ and then ranked using either that same corpus or a much larger web corpus ($Corpus_L$). This section examines how such results vary with the size of the search corpus.

Fig. 4 displays the results for each domain, using Str-INorm-Thresh, a top performer from the previous section. A single line represents the results for a particular choice of the search corpus size. In each case, the same corpus is used for searching and for instance ranking; results that instead used $Corpus_L$ for instance ranking showed very similar trends. The corpus size is varied from about 60 million pages (the size of $Corpus_S$ in the previous experiments) down to 10 million pages. In all cases the results show that OntoSyphon is able to find a substantial number of sensible instances for even the smallest corpus size.

For the high recall end of the curves, using a larger corpus improves the results, though the effect of diminishing returns can be seen from the smaller difference between the 10 and 20 million page corpora vs. between the 50 and 60 million page corpora. Thus, we anticipate that, for a sufficiently large corpus, adding additional pages to the corpus will begin to have negligible effect. Using similar measurements, for instance, Cafarella et al. [6] estimated that a corpus of 400 million web pages would be sufficient to approximate the performance of a lengthy run of the KnowItAll system that searched for CEO-company pairs using the entire web as a corpus.

For the low recall end of the curves, the effect of corpus size is less clear. Interestingly, for the Animal and Food domains, decreasing the corpus size has little impact on the LA at low recall. Manual inspection of the results revealed that, for these domains, there is a substantial overlap of the highest-ranked instances (i.e., those included in the low recall results), independent of the corpus size used. This overlap occurs because the highest-ranked instances tend to occur very frequently, and thus can already be identified with confidence even from the smallest corpus considered. For the music domain, instances tend to be less frequent, and hence this effect is less pronounced. Thus overall we find that using a larger corpus is generally helpful, but that useful instances can still be found with a much smaller corpus, and that using a larger corpus may not improve results at low recall.

7. Discussion and future work

Overall, we conclude that OntoSyphon was highly effective at extracting instances from a web corpus, and that some of our new assessment techniques (especially StructAdjust, Str-INorm-Thresh, and Urns-INorm) significantly improved the accuracy of the results. In particular, using Str-INorm-Thresh with $Corpus_S$, OntoSyphon was able to achieve an LA of about 0.6 while extracting 1550 sensible Animal instances (80% of the total found), 950 Food instances (88% of the total), and (after removing `Player`) 7500 Artist instances (87% of the total). Even higher accuracy may be obtained at a cost of reduced recall.

A Learning Accuracy of 0.6 is on par with the results surveyed by Cimiano et al. [13]. They report LA of between 0.44 and 0.76 (with an average of 0.61 for independent systems) and recall ranging from 0.17 to 0.31 (with an average of 0.24). These results are not directly comparable with ours, since these systems perform a different task (annotating individual documents rather than populating a ontology from many documents), use different ontologies, and in some cases evaluate LA using only those terms marked by the evaluator as valid for some class.[7] Also, these systems generally define recall as the percentage of results from the complete gold standard that was found by the system. For our open-web system, however, recall is reported as a raw number or as a percentage of the set of all answers found by any execution of the system (as with [24]). Nonetheless, the magnitude of these previous results demonstrate that an LA of 0.6 is reasonable, and our results show that OntoSyphon can find many instances at this accuracy level.

Regarding the relative strengths of the class picking and instance classification techniques, four results stand out:

(1) *Explicitly leveraging the ontology's subclass hierarchy via* StructAdjust*was a consistent and very strong factor in improving the precision of class picking.* This method formalizes the notion that a subclass may be a better class choice than its parent, even if there is more raw evidence for the parent.

[7] For instance, C-PANKOW [13] appears to compute LA using only instances that were assigned a class by both the system and an evaluator. For our system (see Eq. (3)) it seemed more accurate to instead assign a value of zero to a pair (i, c) produced by the system but for which $gold_O(i) = \emptyset$ (e.g. for (*t* ruck, *Animal*)). If we likewise ignored instances with no valid class per the evaluator, our LA at 50% recall with Str-INorm-Thresh on $Corpus_S$ would increase from 0.69 to 0.92 for Animals, 0.64 to 0.81 for Artists, and 0.63 to 0.94 for Food. Thus, our LA results may be conservative in comparison, though note that document-driven systems that perform more careful parsing and/or use more reliable input documents (eliminating more implausible instances) may not be as strongly affected by this measurement decision.

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

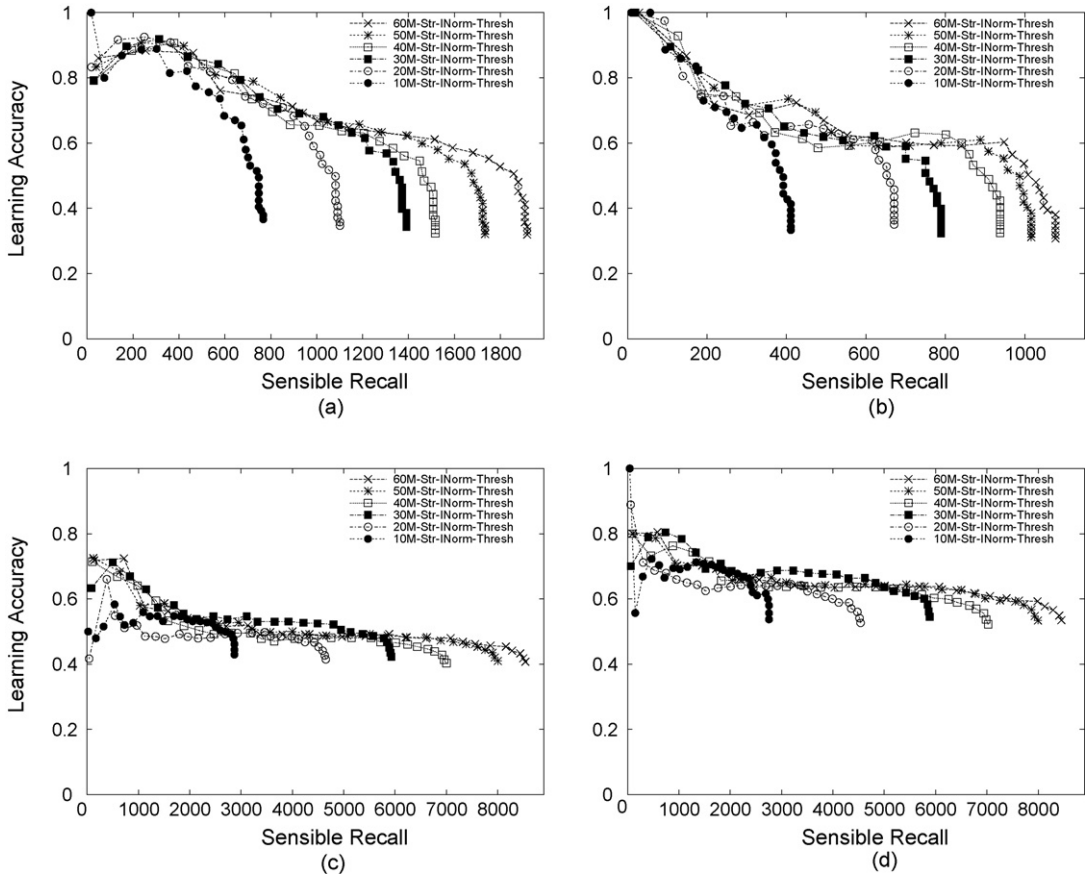

Fig. 4. Results for varying search corpus size. (a) Animal domain, (b) Food domain, (c) Music domain, and (d) Music (without `Player`).

(2) *The best metrics for class picking were not the best techniques for instance ranking*. In particular, class normalization, as well as variants of PMI, performed well for class picking but either had no impact (class normalization) or poor performance (PMI) for instance ranking.
(3) *Normalization was a common theme in the best techniques*. Class normalization gave small but consistent improvements for class picking. Instance normalization, however, was essential to achieving the best results for instance ranking.
(4) *Our two base metrics that performed instance normalization, Str-INorm-Thresh and Urns-INorm, both performed well and about comparably*. These findings are consistent with earlier, non-normalized findings: while Urns was found to be greatly superior to many other techniques in terms of producing accurate probabilities [22], simple Strength-like measures performed almost as well if only a relative confidence ranking, not a probability was required [6]. Because Urns and Urns-INorm are more complex to compute, many situations may thus call for using the simpler Str-INorm-Thresh. On the other hand, users of the final, populated ontology may find actual probabilities very helpful for further processing, in which case Urns-INorm may be best. Further work is needed, however, to empirically demonstrate that Urns-INorm maintains Urns' accuracy with probability assignment while also improving the tradeoff curves shown in Figs. 2 and 3.

The results listed above were highly consistent across the two corpora. In general, using the larger corpus yields more hits per instance, and thus should be expected to yield better discriminative power between true and false instances. The results confirmed this effect for Animals and Artists, but not for Food. Understanding these results requires closer examination of the corpus processing techniques. While $Corpus_L$ is larger, it is searched via public search engines that match any occurrence of the appropriate textual pattern. With $Corpus_S$, however, we used the Binding Engine, which reduces some errors by performing part-of-speech (POS) tagging before matching. For instance, $Corpus_L$ finds strong evidence for "finger" as a kind of food, probably because of many matches for "Vienna fingers," a kind of cookie. BE also finds this candidate, but its POS tagging results in a much smaller amount of evidence for the candidate. Thus, using a larger corpus provides more evidence, but of a slightly less reliable nature. This unreliability has a proportionally larger effect on the Food domain, where there are more multi-word instances that are prone to be misinterpreted. In general, systems must consider this tradeoff of corpus size vs. processing time available per page, while also factoring in the limitations imposed by commercial services vs. the costs of maintaining a private corpus search engine. Finally, OntoSyphon in its present form is clearly not suited for populating every kind of ontology. For instance, ontologies describing things or events that are mentioned only a handful of times on the web are not well suited to our current strategy of using simple pattern-based extractions followed by redundancy-based assessment. Likewise, classes that are either complex (`NonBlandFish`) or ambiguous (`Player`) will not yield good results. We intend to develop techniques to address these issues. For instance, ambiguous classes can be recognized by the small degree of overlap between a class and its parent (as is the case with `Player` and `Artist`, cf. [64]). Moreover, ambiguity may be reduced by adding additional search terms to disambiguate such classes during extraction (e.g., "music"), or by also searching

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

for instances based on synonyms of a class and favoring instances that thus have more than one source of evidence [24]. Such terms and synonyms could be automatically learned or provided in the ontology. Lastly, deciding whether a term such as "dog" should be a subclass or an instance can be challenging even for human ontologists. More work is needed to help OntoSyphon honor the intent of an ontology, e.g., by considering subclasses and instances already present in that ontology.

8. Conclusion

The Semantic Web critically needs a base of structured content to power its applications. Because of the great variety of information, no one approach will provide everything that is needed. Instead, there are multiple complementary approaches that each tackle different knowledge situations. Much content can only be created by human annotators, and incentives are needed to motivate this work [42,41]. Other data is contained in documents that can be effectively leveraged via the document-driven approaches described in Section 2. This article has focused on an alternative ontology-driven method to extract large amounts of comparatively shallow information from millions of web pages. This approach lets us leverage the existing work of skilled ontology designers, extract information from a very large corpus, and focus extraction efforts where it is most valuable and relevant.

While additional work is needed to demonstrate that OntoSyphon is robust across an even wider range of ontologies and can extract non-instance information, our results have demonstrated the feasibility of OntoSyphon's ontology-driven, unsupervised, domain-independent approach. We successfully extracted a large number of instances from a variety of independently created ontologies. We developed novel techniques for selecting the best class for a candidate instance, and showed that the best techniques explicitly utilized the ontology structure to make this decision. In addition, we demonstrated how different instance ranking techniques affect the accuracy of the output, and introduced simple improvements to existing assessment techniques that significantly improved upon these results. Because these new techniques, StructAdjustfor class picking and Str-INorm-Thresh and Urns-INorm for instance ranking, are easy to implement modifications to techniques that have been used for other tasks, our improvements should carry over easily to many other systems (e.g. [13,24,6,39,47,62,60,50]). Future work will examine the many promising directions for further improvements in this area.

Acknowledgments

Thanks to Sebastian Blohm, Christopher Brown, Martin Carlisle, Frederick Crabbe, Oren Etzioni, Jeff Heflin, and the anonymous reviewers for their helpful comments on aspects of this work. This work was partially supported by the Naval Research Council, ONR grants N0001405WR20153 & N00014-02-1-0324, NSF grant IIS-0312988, DARPA contract NBCHD030010, as well as gifts from Google, and carried out in part at the University of Washington's Turing Center.

References

[1] E. Agichtein, L. Gravano, Snowball: Extracting relations from large plain-text collections, in: Proceedings of the 5th ACM International Conference on Digital Libraries (DL), 2000.
[2] E. Alfonseca, S. Manandhar, Improving an ontology refinement method with hyponymy patterns, in: Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC), 2002.
[3] M. Banko, M.J. Cafarella, S. Soderland, M. Broadhead, O. Etzioni, Open information extraction from the web, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 2007.
[4] S. Blohm, P. Cimiano, Using the web to reduce data sparseness in pattern-based information extraction, in: Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), 2007.
[5] S. Blohm, P. Cimiano, E. Stemle, Harvesting relations from the web-quantifying the impact of filtering functinons, in: Proceedings of the 22nd Conference on Artifical Intelligence (AAAI), 2007.
[6] M. Cafarella, D. Downey, S. Soderland, O. Etzioni, KnowItNow: fast, scalable information extraction from the web, in: Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), 2005.
[7] M. Cafarella, O. Etzioni, A search engine for natural language applications, in: Proceedings of the 14th International World Wide Web Conference (WWW), 2005.
[8] S. Cederberg, D. Widdows, Using LSA and noun coordination information to improve the precision and recall of automatic hyponymy extraction, in: Proceedings of the 7th Conference on Computational Natural Language Learning (CoNLL), 2003.
[9] D. Celjuska, M. Vargas-Vera, Ontosophie: A semi-automatic system for ontology population from text, in: Proceedings of the 3rd International Conference on Natural Language Processing (ICON), 2004.
[10] S. Chapman, A. Dingli, F. Ciravegna, Armadillo: harvesting information for the semantic web, in: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004.
[11] P. Cimiano, S. Handschuh, S. Staab, Towards the self-annotating web, in: Proceedings of the 13th International World Wide Web Conference (WWW), 2004.
[12] P. Cimiano, A. Hotho, S. Staab, Learning concept hierarchies from text corpora using formal concept analysis, Journal of Artificial Intelligence Research 24 (2005) 305–339.
[13] P. Cimiano, G. Ladwig, S. Staab, Gimme' the context: context-driven automatic semantic annotation with C-PANKOW, in: Proceedings of the 14th International World Wide Web Conference (WWW), 2005.
[14] P. Cimiano, A. Pivk, L. Schmidt-Thieme, S. Staab, Learning taxonomic relations from heterogeneous evidence, in: Proceedings of the ECAI-2004 Workshop on Ontology Learning and Population, 2004.
[15] P. Cimiano, J. Volker, Text2Onto - a framework for ontology learning and data-driven change discovery, in: Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), 2005.
[16] P. Cimiano, J. Wenderoth, Automatically learning qualia structures from the web, in: Proceedings of the ACL Workshop on Deep Lexical Acquisition, 2005.
[17] F. Ciravegna, Y. Wilks, Designing adaptive information extraction for the semantic web in amilcare, in: S. Handschuh, S. Staab (Eds.), Annotation for the Semantic Web, IOS Press, 2003.
[18] M. Craven, D. DiPasquo, D. Freitag, A.K. McCallum, T.M. Mitchell, K. Nigam, S. Slattery, Learning to construct knowledge bases from the World Wide Web, Artificial Intelligence 118 (1/2) (2000) 69–113.
[19] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, Gate: a framework and graphical development environment for robust NLP tools and applications, in: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL), 2002.
[20] H. Davalcu, S. Vadrevu, S. Nagarajan, OntoMiner: bootstrapping and populating ontologies from domain specific web sites, IEEE Intelligent Systems 18 (5) (2003) 24–33.
[21] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, Semtag and seeker: bootstrapping the semantic web via automated semantic annotation, in: Proceedings of the 12th International World Wide Web Conference (WWW), 2003.
[22] D. Downey, O. Etzioni, S. Soderland, A probabilistic model of redundancy in information extraction, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005.
[23] S. Dumais, H. Chen, Hierarchical classification of web content, in: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2000.
[24] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, A. Yates, Unsupervised named-entity extraction from the web: an experimental study, Artificial Intelligence 165 (1) (2005) 91–134.
[25] I. Frommholz, Categorizing web documents in hierarchical catalogues, in: Proceedings of the 23rd European Colloquium on Information Retrieval Research (ECIR), 2001.
[26] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krupl, B. Pollak, Towards domain-independent information extraction from web tables, in: Proceedings of the 16th International World Wide Web Conference (WWW), May 2007.
[27] G. Geleijnse, J. Korst, Learning effective surface text patterns for information extraction, in: Proceedings of the 18th Belgium-Dutch Conference on Artificial Intelligence (BNAIC), 2006.
[28] G. Geleijnse, J. Korst, V. de Boer, Instance classification using co-occurrences on the web, in: Proceedings of the ISWC 2006 Workshop on Web Content Mining with Human Language Technologies, 2006.
[29] G. Geleijnse, J. Korst, V. Pronk, Google-based information extraction, in: Proceedings of the 6th Dutch-Belgian Information Retrieval Workshop (DIR), 2006.
[30] R. Guha, R. McCool, E. Miller, Semantic search, in: Proceedings of the 12th International World Wide Web Conference (WWW), 2003.
[31] U. Hahn, K. Schnattinger, Towards text knowledge engineering, in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI), 1998.

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002

[32] S. Handschuh, S. Staab, F. Ciravegna, S-CREAM—semi-automatic creation of metadata, in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW), 2002.
[33] M. Hearst, Automatic acquisition of hyponyms from large text corpora, in: Proceedings of the 14th International Conference on Computational Linguistics (COLING), 1992.
[34] A. Kiryakov, B. Popov, I. Terziev, D. Manov, D. Ognyanoff, Semantic annotation, indexing, and retrieval, Journal of Web Semantics 2 (1) (2004) 49–79.
[35] D. Koller, M. Sahami, Hierarchically classifying documents using very few words, in: Proceedings of the 14th International Conference on Machine Learning (ICML), 1997.
[36] K. Lerman, C. Gazen, S. Minton, C.A. Knoblock, Populating the semantic web, in: Proceedings of the AAAI 2004 Workshop on Advances in Text Extraction and Mining, 2004.
[37] Y. Li, K. Bontcheva, Hierarchical, perceptron-like learning for ontology-based information extraction, in: Proceedings of the 16th International World Wide Web Conference (WWW), May 2007.
[38] A. Maedche, V. Pekar, S. Staab, Ontology learning part one—on discovering taxonomic relations from the web, in: N. Zhong, J. Liu, Y.Y. Yao (Eds.), Web Intelligence, Springer, 2002, pp. 301–322.
[39] C. Matuszek, M. Witbrock, R. Kahlert, J. Cabral, D. Schneider, P. Shah, D. Lenat, Searching for common sense: populating cyc from the web, in: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI), 2005.
[40] D. Maynard, W. Peters, Y. Li, Metrics for evaluation of ontology-based information extraction, in: Proceedings of the WWW2006 Workshop on Evaluation of Ontologies for the Web, 2006.
[41] B. McBride, Four steps towards the widespread adoption of a semantic web, in: Proceedings of the 1st International Semantic Web Conference (ISWC), 2002.
[42] L. McDowell, O. Etzioni, S.D. Gribble, A. Halevy, H. Levy, W. Pentney, D. Verma, S. Vlasseva, Mangrove: enticing ordinary people onto the semantic web via instant gratification, in: Proceedings of the 2nd International Semantic Web Conference (ISWC), October, 2003.
[43] L.K. McDowell, M. Cafarella, Ontology-driven information extraction with OntoSyphon, in: Proceedings of the 5th International Semantic Web Conference (ISWC), 2006.
[44] N. Ogata, N. Collier, Ontology express: non-monotonic learning of domain ontologies from text, in: Proceedings of the ECAI-2004 Workshop on Ontology Learning and Population, 2004.
[45] P. Pantel, M. Pennacchiotti, Espresso: leveraging generic patterns for automatically harvesting semantic relations, in: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL), 2006.
[46] P. Pantel, D. Ravichandran, Automatically labeling semantic classes, in: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: (HLT-NAACL), 2004.
[47] P. Pantel, D. Ravichandran, E. Hovy, Towards terascale knowledge acquisition, in: Proceedings of the 20th International Conference on Computational Linguistics (COLING), 2004.
[48] M. Pasca, Acquisition of categorized named entities for web search, in: Proceedings of the 13th ACM Conference on Information and Knowledge Management (CIKM), 2004.
[49] M. Pasca, D. Lin, J. Bigham, A. Lifchits, A. 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 Artifical Intelligence (AAAI), 2006.
[50] S.P. Ponzetto, M. Strube, Deriving a large scale taxonomy from wikipedia, in: Proceedings of the 22nd National Conference on Artifical Intelligence (AAAI), 2007.
[51] A.-M. Popescu, O. Etzioni, Extracting product features and opinions from reviews, in: Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), 2005.
[52] C. Ramakrishnan, K.J. Kochut, A.P. Sheth, A framework for schema-driven relationship discovery from unstructured text, in: Proceedings of the 5th International Semantic Web Conference (ISWC), 2006.
[53] M.-L. Reinberger, P. Spyns, A. Johannes Pretorius, W. Daelemans, Automatic initiation of an ontology, in: Proceedings of the Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE), 2004.
[54] S. Richardson, W. Dolan, L. Vanderwende, Mindnet: acquiring and structuring semantic information from text, in: Proceedings of the 17th International Conference on Computational Linguistics (COLING), 1998.
[55] M. Ruiz-Casado, E. Alfonseca, P. Castells, Automatic extraction of semantic relationships for wordnet by means of pattern learning from wikipedia, in: Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), 2005.
[56] D. Schneider, C. Matuszek, P. Shah, R. Kahlert, D. Baxter, J. Cabral, M. Witbrock, D. Lenat, Gathering and managing facts for intelligence analysis, in: Proceedings of the International Conference on Intelligence Analysis, 2005.
[57] R. Snow, D. Jurafsky, A.Y. Ng, Learning syntactic patterns for automatic hypernym discovery, in: Proceedings of the 18th Annual Conference on Neural Information Processing Systems (NIPS), 2004.
[58] S. Soderland, Learning to extract text-based information from the World Wide Web, in: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD), 1997, pp. 251–254.
[59] F.M. Suchanek, G. Kasneci, G. Weikum, Yago: a core of semantic knowledge unifying wordnet and wikipedia, in: Proceedings of the 16th International World Wide Web Conference (WWW), May 2007.
[60] H.T. Tanev, B. Magnini, Weakly supervised approaches for ontology population, in: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), 2006.
[61] P.D. Turney, Mining the web for synonyms: PMI-IR versus LSA on TOEFL, in: Proceedings of the 12th European Conference on Machine Learning (ECML), 2001.
[62] W. van Hage, S. Katrenko, G. Schreiber, A method to combine linguistic ontology-mapping techniques, in: Proceedings of the 4th International Semantic Web Conference (ISWC), 2005.
[63] P. Velardi, R. Navigli, A. Cucchiarelli, F. Neri, Evaluation of OntoLearn, a methodology for automatic learning of domain ontologies, in: P. Buitelaar, P. Cimmiano, B. Magnini (Eds.), Ontology Learning and Population, IOS press, 2005.
[64] J. Volker, D. Vrandecic, Y. Sure, A. Hotho, Learning disjointness, in: Proceedings of the 4th European Semantic Web Conference (ESWC), 2007.
[65] T. Wang, Y. Li, K. Bontcheva, H. Cunningham, J. Wang, Automatic extraction of hierarchical relations from text, in: Proceedings of the 3rd European Semantic Web Conference (ESWC), 2006.
[66] D. Widdows, Unsupervised methods for developing taxonomies by combining syntactic and statistical information, in: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL), 2003.

Please cite this article in press as: L.K. McDowell, M. Cafarella, Ontology-driven, unsupervised instance population, Web Semantics Sci Serv Agents World Wide Web (2008), doi:10.1016/j.websem.2008.04.002