Domain Cartridge: Unsupervised Framework for Shallow Domain Ontology Construction from Corpus

Subhabrata Mukherjee[†] Jitendra Ajmera[‡] Sachindra Joshi[‡]
[†]Max Planck Institute for Informatics, [‡]IBM India Research Lab
smukherjee@mpi-inf.mpg.de, jajmera1@in.ibm.com, jsachind@in.ibm.com

ABSTRACT

In this work we propose an *unsupervised* framework to construct a shallow domain ontology from corpus. It is essential for Information Retrieval systems, Question-Answering systems, Dialogue etc. to identify important concepts in the domain and the relationship between them. We identify important *domain terms* of which *multi-words* form an important component. We show that the incorporation of multi-words improves parser performance, resulting in better parser output, which improves the performance of an existing Question-Answering system by upto 7%. On manually annotated smartphone dataset, the proposed system identifies 40.87% of the domain terms, compared to 22% recall obtained using WordNet, 43.77% by Yago and 53.74% by BabelNet respectively. However, it does not use any manually annotated resource like the compared systems. Thereafter, we propose a framework to construct a shallow ontology from the discovered domain terms by identifying four domain relations namely, *Synonyms* ('similar-to'), *Type-Of* ('is-a'), *Action-On* ('methods') and *Feature-Of* ('attributes'), where we achieve *significant* performance improvement over WordNet, BabelNet and Yago without using any mode of supervision or manual annotation.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General

General Terms

Knowledge Extraction, Ontologies

Keywords

Ontology; Unsupervised Framework; Relation Extraction; Distributional Similarity

1. INTRODUCTION

An ontology can be viewed as a data structure that specifies terms, their properties and relations among them for a richer knowledge representation [47]. Such a knowledge representation makes an information retrieval system aware of a domain, so that it can capture domain concepts and associations well, to perform tasks like answering queries, conducting dialogue more efficiently. We show that a domain-aware Question-Answering system can have a 7% boost in recall by just knowing the multi-word domain terms (like 'Samsung Galaxy S IV', 'Sony Experia').

In this work, we present a framework to automatically construct a shallow domain ontology from a set of knowledge articles and pdf manuals in a given domain. We view this domain ontology as a graph, where the nodes are domain concepts and the edges depict relations among these concepts. Figure 1 shows a snapshot of a smartphone domain ontology, with the important domain concepts as nodes and a set of relations among these concepts as directed edges. The relations include the most commonly used *Type-Of* or hyponymy relation (denoted as the edge label T), *Feature-Of* (depicted as the label F, e.g. 'operating-system' is a feature of 'device'), *Action-On* (depicted as the label A, e.g. 'install' is an action on 'operating-system') and *Synonym* (depicted as the label S, e.g. 'device' is synonymous to 'handset'). Although, there may be more fine-grained relations possible in a given domain (e.g. temporal, spatial), we believe that these four relations are common across domains and can be used to represent domain knowledge efficiently, as in an object oriented programming methodology (e.g. JAVA) where 'objects' are defined to have a set of 'attributes' (features F), 'methods' (actions A) and 'superclasses' (type T).

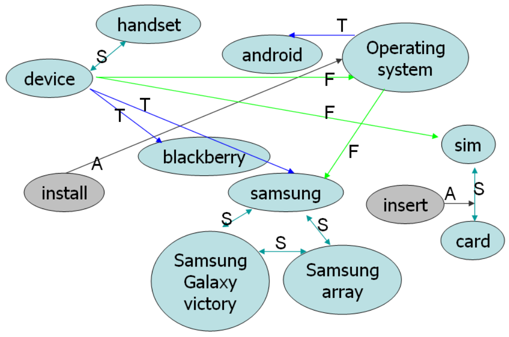

Figure 1: An example ontology from the smartphone domain.

At first we are interested in discovering the important concepts in a given domain, of which multi-word terms form an important component (a task referred to as *Domain Term Discovery (DTD)* in this work). Incorporation of such domain terms can improve a parser's performance. The following example depicts the utility of the DTD process for a parser. Consider the string "transfer files via USB cable". First, the DTD process allows the tokenizer to recognize 'USB-cable' as a single token. Secondly, the domain knowledge that 'files' is typically a noun in the smartphone domain and

CIKM '14, November 03 - 07 2014, Shanghai, China
Copyright 2014 ACM 978-1-4503-2598-1/14/11$15.00.
http://dx.doi.org/10.1145/2661829.2662087.

not a verb, leads to a correct and complete parse of the string. Since these steps are the basic building blocks of a Question-Answering system [10, 13], we show that its performance can be improved by improving the parser performance.

Thereafter, we want to determine the relations between the discovered domain terms (referred to as *Domain Relation Discovery (DRD)* in this work). We show that once the parser becomes aware of the domain concepts in the first phase, it generates refined output, in terms of better parser relations, which helps in the DRD process. The discovered relations can be used for query expansion (e.g. by considering *Synonyms* along with the original query), interactive dialogue systems (e.g. for the user query "the battery of my device depletes very fast", a DRD inference that 'battery' is a *Feature-Of* 'phone' as well as a 'tablet' device - enables the system to clarify about the type of device). Similarly, *Type-Of* relations can help in query re-formulation. E.g. for the user query "screen freezes E5150", the DRD inference that 'E5150' is a type-of 'Error' can result in the query re-formulation "screen freezes error E5150". Query re-formulation improves the performance of Question- Answering system by increasing its recall.

Overall we focus on discovering all the domain concepts and relations *efficiently* in an automated way, and construct a shallow domain ontology. Our work differs in the ontology construction process from related works [49, 16] in the usage of an unsupervised framework for ontology generation, domain term and relation discovery, where we do not make use of any manually created resource such as WordNet [32], Wikipedia [50], ConceptNet [18] or other readily available ontologies [49].

An issue with such manually created resources with annotations is that they mostly contain generic concepts and miss out on many domain specific concepts and associated relations. In addition to the labor cost involved in constructing such resources, they are not updated frequently and as a result lack the newly coined concepts in the domain (e.g. 'HTC Desire', 'Nokia Lumia 1020'). On the other hand our system extracts domain concepts and relations from tutorials and manuals in a given domain, which are readily available, that creates a richer knowledge representation. As the approach is unsupervised, requiring no manual annotation, human labor in creating and maintaining the ontology is markedly reduced.

We compare our approach to other state-of-the-art systems like BabelNet [36] and Yago [48]. These systems harvest knowledge from manually created resources like WordNet and Wikipedia. Yago and BabelNet perform better than our system in domain-term discovery with a recall of 43.77% and 53.74% respectively. Although our system obtains 40.87% recall in discovering domain terms, it does not make use of any world knowledge or human annotated resource. On the other hand, we perform much better than BabelNet, WordNet and Yago in discovering domain relations. For instance, WordNet, BabelNet and Yago do not give Feature-Of or Action-On relations. Our system *significantly* outperforms them on relations available to all the systems.

2. OVERVIEW OF OUR FRAMEWORK

In this section we give a high level overview of the proposed Domain Cartridge framework (refer to Figure 2). The input to the system is a corpus consisting of knowledge articles, manuals, tutorials etc. in a variety of formats in a given domain. Suitable adaptors are then used to bring the information in plain text form such that the documents can be parsed by a slot grammar parser (ESG) [30].

The ESG parser is used due to its speed. It is $50 - 100$ times faster than the Charniak parser [7]. Simple transformations over the ESG parser output and encoding alternate variants of common expressions provide *Shallow Semantic Relationship (SSR)* (referred

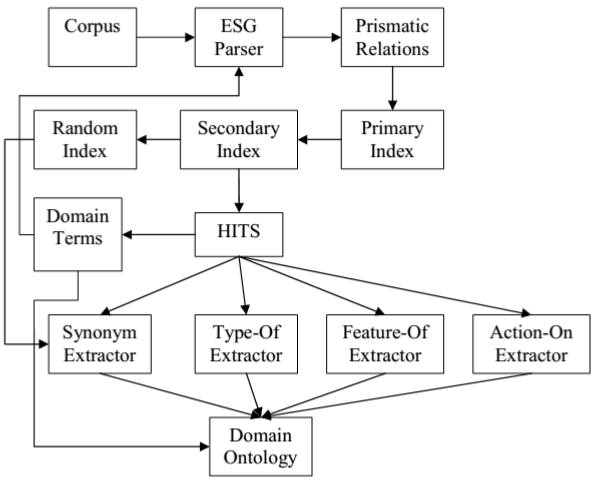

Figure 2: Our framework: Domain Cartridge.

to as *Prismatic* in Figure 2) annotations. A detailed discussion on the types and constituents of these relations is presented in [10].

The first phase consists of discovering important domain terms, referred to as Domain Term Discovery (DTD) in this work (refer to Section 3). In this phase, domain multi-words are discovered using *noun phrase chunking* on the parser output, which is used to *bootstrap* the parser. This results in the parser generating better SSR as it becomes domain-aware.

The Lucene index called the *Primary Index* contains all SSR and associated concepts in the corpus. The *Secondary Lucene Index* created from the primary index bears only unique SSR. The indices provide easy access to all SSR, concepts and documents in corpus to incrementally build the Domain Cartridge, as well as help in scalability. During domain migration, only *primary index* needs to be changed as other indices are derived from it. Section 4 discusses the index construction process in details.

To discover more fine-grained multi-word concepts and relations, HITS algorithm [23] is used over the now-refined parser output (refer to Section 4.4), after the noun phrase chunking process. The newly discovered domain terms by HITS are incorporated in the parser lexicon, further enriching it. The parser is run again to generate better relations, and previous steps are iterated till convergence. The HITS algorithm takes its input from the secondary index.

In order to identify similar domain concepts, a dimensionality reduction technique called *Random Indexing* [45] is used leveraging *relational distributional similarity* of the candidate concepts (refer to Section 4.3). The *Random Index* is created out of the secondary index containing similar neighbors for a concept.

The output of HITS and Random Index is used to discover relations between the domain terms (referred to as *Domain Relation Discovery* or *DRD* in this work; refer to Section 5). A classifier is built for each of the 4 relation types, which takes as input a word pair and uses SSR features to predict the relation type existing between them.

3. DOMAIN TERM DISCOVERY (DTD)

The first step in gathering insight about a new domain is to discover a list of important domain concepts. For example, in the smartphone domain, terms like 'Samsung-Galaxy-Tab', 'Call-log', 'Call-forwarding', '4g-connection' can be considered important concepts. Similarly in a finance domain, terms like 'Company', 'Transaction', 'Sales-Tax' etc. can be considered as domain terms.

An important characteristic feature of domain terms are the multi-word tokens. In one of the experiments, with a manually designed list of financial domain terms, we observed that as many as 50% of the important tokens were multi-words out of 7000 domain terms.

3.1 DTD using Noun Phrase Chunking

We make use of the parse tree output of the slot grammar parser (ESG) [30] to discover important domain terms. Figure 3 shows an example parse tree with various parser relations connecting different concepts.

First, all the knowledge sources such as knowledge articles (HTML), manuals (PDF) etc. are appropriately pre-processed to obtain simple text documents. The pre-processing stage itself involves a lot of issues related to formatting, extracting information from pdf etc.

The documents are parsed using the slot grammar parser. Noun phrases are extracted from the parses and processed to discover domain terms. Figure 3 shows the parse tree corresponding to the sentence *"Turn the Wi-Fi radio on or off"*. We consider all the terms which have part-of-speech (POS) tags as 'nouns'. These terms now become candidates for domain terms. Furthermore, we also consider the sub-tree of which this noun term is the root and make them candidates for domain terms. In the example shown in Figure 3, such a sub-tree is *"The Wi-Fi radio"*.

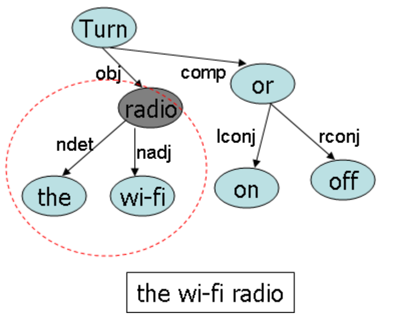

Figure 3: Parse tree for the title *"Turn the wi-fi radio on or off"*.

This approach results in long candidates like *"issues related to receiving calls"*. Although such patterns are valid noun-phrases, we want to recognize only atomic units in our list of domain terms. In this example, we would like to keep *"issues"* and *"receiving calls"* as separate tokens. Hence, we perform a subsequent post-processing where the candidates are split into atomic domain terms based on the presence of *verbs and stop words*. Finally, only those domain term candidates are retained for which the count is greater than a threshold. Table 1 shows a snapshot of the domain terms discovered using the noun phrase chunking approach.

In our analysis, this approach results in a highly precise set of domain term candidates. We generated approximately 2900 and 3400 domain terms in the financial domain and smartphone domain from 9k and 16k titles, respectively.

samsung blackberry device software novatel software-version application htc-evo wi-fi memory-card bluetooth motorola kyocera browser voicemail microsoft-exchange lg-optimus elite samsung-m400 samsung-galaxy-victory software-updates samsung-array text-messaging wallpaper synchronization face-book iphone htc aircard touchscreen gps blackberry-bold ipad motorola-xprt htc-evo-3d sanyo-vero

Table 1: Snapshot of multi-word domain terms discovered using noun phrase chunking.

3.2 Parser Domain Term Lexicon

The ESG parser maintains a lexicon of multi-word entries which are used by the parser in the subsequent phases as a single unit resulting in a better parser output. For example, if the ESG parser has the prior knowledge that 'touch screen' is a multi-word from the domain term lexicon, then it will parse "touch screen of the mobile" with 'touch screen' as a *noun*. Without this domain knowledge, 'touch' will be treated as a *verb* and 'screen' as a *noun*, resulting in a different parse; and in many cases a noisy or incomplete parse. The domain term discovery process adds domain terms to the parser lexicon. Subsequently, the parser uses the multi-words as a single token (e.g. 'sprint navigation' will be considered as a noun concept, and not processed separately as a verb and a noun) and the unigrams are favored as nouns (e.g. 'files' will be processed as a noun during ambiguity). The DTD process results in the parser generating better quality output as the number of incomplete or noisy parses is reduced. Since the lexicon is core to the functionality of the parser, we would like to keep it as clean as possible (high precision). Hence we use the noun phrase chunking approach only on the document titles for the following three reasons: 1) The extraction of titles is invariably easy and precise. 2) The titles do not suffer from arbitrary formatting, references to pictures etc. present in the body of the text, and therefore the parsing of titles is almost always correct. 3) In our end-application of a QA system, the titles represent the information need that the body of the text is providing.

However, this leads to low recall for domain term discovery and is only used to *bootstrap* the parser. In order to further enrich the lexicon with more fine-grained domain terms, we use another algorithm called HITS on the refined parser output. This is explained in details in Section 4.4. The newly discovered domain terms by HITS are again incorporated in the parser lexicon, further enriching it, and previous steps are iterated till convergence.

Once the domain terms are identified and refined parser relations generated, the relationship between the terms need to be discovered to construct the ontology. Since we want to discover domain relations across all possible pairs of domain concepts which would be computationally costly and involve a lot of redundant computation, we describe an efficient framework that indexes all parser relations and creates a random projection for every domain concept in a relatively lower dimensional space. This facilitates fast computation of similarity between candidate domain terms as well as different relation discovery.

4. INDEX CREATION: FEATURE SELECTION, DIMENSIONALITY REDUCTION

In this section, we present the index creation module of our framework that helps in dimensionality reduction as well as identifying dominant domain terms and relations that are used as features in the remaining part of the system.

After the discovery of domain terms, as explained in Section 3, they are provided as an input lexicon to the slot grammar parser. Multi-word tokens are now identified as single tokens by the parser resulting in better parser output. In the example in Figure 3, 'Wi-Fi radio' will be parsed as a single token.

Shallow semantic relationship (SSR) [10] annotation is done over the ESG parser output which consists of rules to generate projections for all the frames in the corpus and generate normalized parser relations. SSR detects alternative syntactic contexts expressing the same semantic relationship between two or more entities. For example, the sentences "Samsung has a battery" and "Samsung's battery died" will both generate the same relation 'nnMod:samsung_ battery', where "nnMod" represents the type of the SSR relation (noun modifier), and "samsung" and "battery" represent the associated concepts.

As we process the domain corpus, SSR annotations over ESG parser output are discovered across various domain terms and other terms. We index these SSR to use them later for finding distributional similarity, and querying for other statistics such as "how many documents contain a relationship between the term 'install' and other terms".

4.1 Primary Index

For each document D in the domain corpus, we index the SSR $R_d = \{r_1, r_2, ..., r_N\}$ in the primary index using Lucene. Each SSR r_i has a type (e.g. verb-object, noun-adjective etc.) and associated concepts. The Lucene analyzer was changed to produce a tokenstream that consists of regular tokens, their Part-of-Speech tags and the SSR shared between the tokens as opposed to the default white space tokenizer that produces a tokenstream consisting of words or regular tokens. For example, a document with text content as *"use this cable to connect iPhone to your computer and sync changes"*, the following tokenstream is produced by the analyzer: "*use cpt:verb:use rel:dm_obj:use_cable this cable cpt:noun: cable to connect cpt:verb:connect rel:dm_comp:connect_computer rel:dm _obj:connect_iphone iphone cpt:noun:iphone to your computer cpt: noun:computer to sync cpt:verb:sync and charge cpt:verb:charge...*"

In the example above, the SSR relations are marked with "rel:" prefix and the concepts associated with these relations are separated by an underscore. Accordingly, the token "rel:dm_obj:use_cable" means that this is a relation token with "dm_obj" type (verb object pair) and "use" and "cable" are the associated concepts. The tokenstream maintains the order of the concepts and SSR in which they appear in the documents. Such tokenization and indexing allows us to retrieve information such as tokens (or SSR) in the neighborhood of a querying token (or SSR), all the corpus documents containing a specific SSR, document frequencies of SSR etc.

4.2 Secondary Index

After the primary index has been created, we traverse over its indexed tokens to create the secondary index. Note that in this index, we store only the unique SSR (i.e tokens starting with 'rel:' prefix). This index allows us to retrieve unique SSR based on their types or based on one of the constituents. For example, the query "get all the 'dm_comp' relations, where 'connect' is one of the constituents" can be effectively answered by this index.

Each SSR in the secondary index is of the form $rel : word_1\_word_2 \_word_3$, where $word_1$ and $word_3$ are domain terms (*nouns*) and $word_2$ belongs to *verbs* or *prepositions* and $rel \in \{svo, dm\_*, nnMod, npo\}$. The SSR in the secondary index are of the following form:
1. *svo* depicts a subject-verb-object tuple. For example: *rel:svo:phone _offer_feature, rel:svo:phone_show_message etc.*
2. *nnMod* depicts noun-noun modifications. For example: *rel:nnMod:iPhone_battery, rel:nnMod:screen_icon etc.*
3. *dm* depicts actions on entities. For example: *rel:dm_obj:use_ phone, rel:dm_comp:plug_iPhone etc.*
4. *npo* depicts terms connected by prepositions. For example: *subscription_to_service, battery_on_phone etc.*
We traverse the indexed SSR in this secondary index for discovering domain terms and building the distributional similarity matrix as explained in the next section.

4.3 Random Index: Dimensionality Reduction

In this section, we explain our random indexing framework for computing *distributional similarity* [31] between two terms, which suggests that terms sharing similar contexts are likely to be similar. Random Indexing (RI) [45] is a word co-occurrence based approach to statistical semantics. RI uses statistical approximations of the full word co-occurrence data to achieve dimensionality reduction, resulting in much quicker running time and fewer required dimensions.

In most co-occurrence models, a word-by-word matrix is constructed, where the values denote how many times the column's word occurred in the context of the row's word. RI instead represents co-occurrence by assigning each word a high dimensional index vector and keeping a running sum of all the index vectors for words that co-occur. The index vectors are very sparse reducing chances of a sparse match.

Random Indexing can also be seen as an alternative to Latent Semantic Analysis [9]. Random Indexing is more scalable and allows for the incremental learning of context information. Although LSA is efficient, it suffers from scalability issues. It starts by generating a $termXdocument$ matrix which grows with the corpus. For finding the final LSA model, Singular Value Decomposition (SVD) is commonly used for factorization of the term-document matrix, which is computationally costly. Also, the LSA model cannot be implemented easily and efficiently in an incremental or out-of memory fashion.

To find the most likely candidate terms for a given term requires computing similarity between all possible pairs of words. The quicker running time and reduced dimensionality features of the random indexing approach allow us to do this efficiently.

However, as opposed to most of the previous works that consider raw neighborhood of a term as context (say, preceding and following N terms) for random indexing [14, 39], we use only those neighboring terms to define the context for a target term that share a syntactic dependency (given by the slot grammar parser) with the target term. These syntactic dependencies are represented in Figure 3 by connecting edges between nodes. We traverse over the indexed SSR in the secondary index and for each SSR, we add the index vectors of a constituent term to all the other constituent terms.

In the distributional similarity computation, we consider all SSR to be important. However, some SSR (like "nnMod:samsung_charger, dm_obj:charge_samsung") are more important to certain domain terms (like 'samsung') than others (like "nnMod:samsung_color, dm_obj:use_samsung"). In order to discover the dominant domain SSR as well as additional domain terms we use another algorithm, as explained next.

4.4 HITS Index: Dominant Domain Term and SSR Discovery from Secondary Index

In order to discover the dominant domain SSR and terms from corpus, we use a graph based algorithm on the secondary index. The earlier DTD process (refer to Section 3) used noun phrase chunking only on document titles, in order to keep the lexicon as clean as possible to bootstrap the parser.

Any SSR can be visualized as a *hub* generating features to create the domain terms. Any domain term can be visualized as an *authority* influenced by incoming features from the hubs as depicted in Figure 4. A good hub or domain SSR is the one generating a lot of important domain terms. A good authority or domain term is the one being influenced by a lot of domain SSR. The link strength between the hub and authority is taken as the number of SSR in the Primary Index in which both of them participate. We apply HITS algorithm [23] on the hub-authority graph to discover dominant SSR and domain terms, and add them to the HITS index.

In this algorithm the authority and hub scores are defined in terms of each other recursively. An authority score is computed as the sum of the hub scores of each node that points to it. A hub value is the sum of the authority scores of each node that point to it. This is done iteratively for all nodes until convergence. In our case, the scores are weighed by the link strength connecting 2 nodes.

Table 2 shows a snapshot of the domain words discovered by HITS, that are not detected earlier by the domain term discovery approach using noun phrase chunking on titles. The dominant SSR are used in the latter stages for relation discovery.

The newly discovered domain terms by HITS are incorporated in the parser lexicon, further enriching it, which makes the parser generate a refined output. The parser is run again and previous steps are iterated till no additional domain terms are discovered with high authority score.

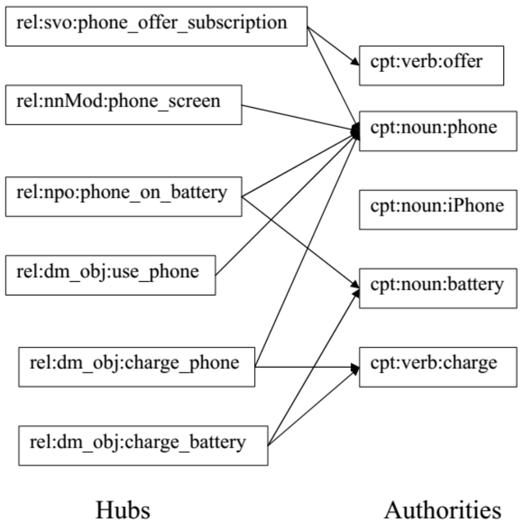

Figure 4: Hubs and authorities in the domain.

wi-fi optimus-g set-up novatel-wireless e-mail sierra-wireless apple-id google-maps play-music mobile-network 10-digit internet-explorer slacker-radio caller-id google-search address-book my-computer software-update blackberry-id as-well-as windows-update terms-of-service drop-down pro-700 add-on scp-2700 mac-os device-manager voice-mail non-camera tru-install back-up task-manager mobile-email thinkfree-office exchange-server preferred-roaming-list minus-sign new-profile windows-live-messenger non-subscribers non-subscriber

Table 2: Snapshot of multi-word domain terms discovered by HITS (not found by noun phrase chunking on titles).

5. RELATION DISCOVERY

The previous sections discuss the approach for domain term discovery from corpus and creation of the indices namely, *Primary, Secondary, Random* and *HITS*. In this section, we outline the method for identifying relations between the discovered domain concepts using the indices.

Random index is used to get a set of similar candidates for a word based on similar SSR distribution in the corpus. The HITS index consists of dominant domain terms and SSR discovered from the secondary index. We categorize the SSR discovered by HITS into the following 2 types:

1. 1-*hop neighbor SSR of a word* denotes the SSR that the target word participates in. For example : "phone - {*rel:svo:instruction_phone_press, rel:dm_arg:keep_phone, rel:dm_subj:screen_phone rel:npo:port_on_phone*}"

2. *Mutual SSR of a word pair* denotes the SSR in which the word pair participates together. For example : "iphone & mobile - {*rel:nnMod:iphone_mobile-data, rel:nnMod:iphone_mobile-hotspot, rel:nnMod:iphone_mobile-charger*}"

5.1 Synonym Discovery

We define two words to be similar if they appear in a similar context. Here, we follow the notion of *relational distributional similarity* [31]. Only those word pairs whose part-of-speech tags are the same are retained ('noun-noun' or 'verb-verb'). Word pairs, whose members participate in any SSR together, are filtered out. As we will see later, such mutual SSR shared by a word pair are indicative of *Action-On*, *Type-Of* and *Feature-Of* relations.

Let w_i and w_j be a candidate word pair extracted from the random index. Feature vector for w_i is defined as: $F_{w_i} = \{rel_{l_i} : w_{k_i}\}$, where rel_{l_i} is any 1-hop neighbor SSR of w_i in the HITS index, connecting w_i and w_{k_i}. The *relational distributional similarity* between 2 words is given by:

$$Sim(w_i, w_j) = \frac{\sum_p \mathbf{I}_{l_i=l_j, k_i=k_j}(f_{w_{k_i},p}, f_{w_{k_j},p})}{\sum_p \sum_r \mathbf{I}_{l_i=l_r, k_i=k_r}(f_{w_{k_i},p}, f_{w_{k_r},p})}$$

where, $f_{w_i,p}$ is the p^{th} element of F_{w_i} and $\mathbf{I}$ is an indicator function.

The numerator of the above equation counts the number of times any word w_{k_i} appears in both the feature vectors F_{w_i} and F_{w_j} with SSR rel_{l_i}. The denominator counts the number of times the word w_{k_i} appears in the feature vector of any other word with the SSR rel_{l_i}. For example, consider the feature vector of the words 'device' and 'handset' and the feature 'nnMod:charger' that appears in both the feature vectors. Now, if 'charger' appears only in the context of 'device' and 'handset' with the SSR 'nnMod' it will contribute 1 to the similarity score. But if it appears a large number of times with other words, the contribution will be less indicating that it is a frequently occurring word in the corpus (e.g. *prepositions* or frequently occurring *verbs*). Word pairs with similarity scores greater than a threshold are added to the Synonym list.

This process results in high recall but low precision, as words like 'folder' and 'tab' have a high relational distributional similarity due to overlapping context words like "*open, close, minimize, shortcut, multiple*" *etc.* that can act on both of them. To alleviate this, we use another parser feature to increase the precision. The ESG parser tags a word with a set of attributes like *noun, animated, physical object, device etc.*. We constrain a word pair to have overlapping ESG parser attributes in order to be deemed similar, which increases the precision of the classifier. For example:
"*Folder: noun cn sg physobj artf capped creation*
Photo: noun cn sg physobj artf capped creation
Tab: noun cn sg physobj abst artf inst capped doc comm"

In the above example, the relational distributional similarity is high for all the word pairs. However, the parser feature distinguishes between 'folder' and 'tab', but cannot distinguish between 'folder' and 'photo'.

5.2 Action-On Discovery

Action-On represents any activity or 'method' on the domain term. For example, 'charge' can be an activity on 'battery' and 'iPhone', 'display' can be an activity on 'menu' and 'icon' etc. By definition, an Action-On relation pair consists of a 'verb' that acts on a 'noun'. The SSR *dm and svo* help in Action-On identification.

The *dm* SSR can be classified as *dm_subj, dm_obj, dm_prep, dm_comp and dm_arg* which depict scenarios where the noun is the subject or object of the verb, the verb is connected to the argument with a preposition etc.

The *svo* SSR represent subject-verb-object tuples.

E.g. "*rel:svo:tap_**add_account**, rel:svo:phone_**access_internet**, rel:svo:mobile_**sync_phone**, rel:svo:account_**use_phone** etc.*" The *verb-object (vo)* SSR is extracted from the 'svo' SSR which represents action-on activities.

HITS index is traversed to extract all the *dm* and *vo* SSR. Each word pair in the index is ranked according to the number of such mutual SSR (dm and svo) directly shared by them. Those with a count greater than a threshold are added to the Action-On list.

5.3 Type-Of Discovery

Type-Of relations depict *Is-A* hierarchy *i.e.* a parent-child relation. For example: "Samsung is a Type-Of mobile, Internet Explorer is a Type-Of browser, Angry Birds is a Type-Of application etc.". In order to discover the Type-Of clues, the *svo* SSR in the HITS index are grouped by the verbs, and *npo* SSR are grouped by the prepositions.

The *svo* SSR having the verb *include* and npo relations having prepositions *like, such-as* and *as* are found to be most informative, and are used to discover Type-Of relations from the HITS index. E.g. "*rel:svo:**devices**_include_**HTC**, rel:npo:**applications**_ such-as_**WhatsApp**, rel:npo:**device**_like_ **computer**, rel:npo:**features**_like_**call** rel:npo:**contact**_such-as_**address**, rel:svo:**location**_include_**address** etc.*".

Type-Of candidates are also discovered from the dominant domain terms from the HITS index, where the words are connected by *or* and *especially.* These keywords are taken from the Hearst [19] patterns. E.g. *service_or_process*, *prev_or_next*, *messages_especially_sms_and_ mms etc.*

5.4 Feature-Of Discovery

Feature-Of relations depict components or functionalities of a domain term. For example: "screen is a Feature-Of mobile", "wi-fi is a Feature-Of network", "life is a Feature-Of battery" *etc.* In order to discover Feature-Of relations we use 2 primary SSR.
1. *nnMod* SSR that depict noun-noun modifications. For example: "*rel:nnMod:**network_life**, rel:nnMod:**account_settings**, rel:nnMod:**iPhone_battery*** etc."
2. *svo* SSR (Section 5.2) that depict subject-verb-object tuples. The *subject-object (so)* word pairs are extracted from 'svo' SSR as Feature-Of candidates, excluding the feature *include* used for Type-Of discovery. E.g. "*rel:svo:**motorola-photon-4g**_run_**device-software**, rel:svo:**router**_decrease_**signal-strength**, rel:svo:**find-a-store**_open_**locator** etc.*"

6. EXPERIMENTAL EVALUATION

We collected 5000 articles, tutorials and manuals from the smartphone domain. We consider WordNet [32] as the first baseline. We also compare our system to other existing semantic knowledge bases like BabelNet [36] and Yago [48]. Both of them harvest knowledge from manually created lexicons and encyclopedia like WordNet and Wikipedia.

6.1 Domain Term Evaluation

We used the back-of-the-book index of manuals, which contains pointers to relevant topics in the book, to create the ground truth for domain term discovery. The set of relevant topics extracted from manuals represents the set of terms one would like to discover from corpus. Thus terms discovered through the proposed technique should cover as much as possible of this set.

Table 3 compares the recall of the 2 approaches for domain term discovery (using NP chunking on titles and by using HITS) with WordNet, Yago and BabelNet on this set. It is observed that the HITS approach discovers a number of new multi-words and discards many of the multi-words discovered by noun-phrase chunking from titles, with scores less than the threshold. Recall of WordNet is low as it mostly misses out on the multi-words and detects only unigrams. BabelNet and Yago use world knowledge acquired from manually constructed WordNet and Wikipedia, and obtain a better recall in identifying domain terms.

Method	Recall
WordNet	22.62%
NP Chunking on Titles	32.45%
HITS	40.87%
Yago	43.77%
BabelNet	53.74%

Table 3: Domain term evaluation.

To align the domain term discovery evaluation process with our end goal of improving a QA system, we evaluated the performance of a QA system [13] trained in the smartphone domain, with and without such automatically discovered domain term lexicon. Table 4 presents the comparison of the two scenarios.

Recall@N	With Domain Term Lexicon	Without domain term lexicon
recall@1	0.4	0.33
recall@2	0.49	0.45

Table 4: Performance of a QA system with and without domain term lexicon.

6.2 Relation Evaluation

For relation discovery, the performance of all the systems, in terms of recall and precision, is computed with respect to the annotation statistics provided by *two* annotators. We extracted 3000 domain terms from the secondary index and 100 similar neighbors for each term from the random index generating 0.6 million word pairs. From these word pairs, we retained word pairs such that one of the members belong to the titles. As the titles contain most of the important domain terms, this constraint ascertains that most of the extracted word pairs would be relevant ones in the domain. The word pairs are partitioned into *two* sets.

- Set 1 consists of word pairs having some mutual SSR with each other
- Set 2 consists of word pairs not in Set 1 and having *relational distributional similarity* score greater than a threshold

We further divided word pairs in Set 1 as 'noun-noun' and 'verb-noun' word pairs. *Two* annotators (Ann) were asked to annotate 500 'noun-noun' word pairs in Set 1 as *Feature-Of* or *none* and 500 'verb-noun' word pairs in Set 1 as *Action-On* or *none*. They were further instructed to annotate 1000 randomly picked word pairs from Set 2 as *Synonyms, Type-Of* or *none*. Since Feature-of and Action-On relations are pretty straightforward to detect, the annotators worked on disjoint sets; whereas for Type-Of and Synonyms each of them annotated the entire Set 2.

Table 5 shows the annotator agreement for the Synonyms. The *Kappa Score* [5] is high at 0.92 due to the large number of word pairs the annotators agree are not Synonyms. Table 6 shows the annotator agreement for Type-Of. The Kappa Score is 0.70 due to the large number of word pairs the annotators agree are not Type-Of. Table 7 shows the annotation statistics for Feature-Of and Action-On, with 'True' denoting the number of word pairs that the annotators agree belong to the given relation type.

Agreement	Ann 2: Yes	Ann 2: No
Ann 1: Yes	49	66
Ann 1: No	41	844

Table 5: Annotator agreement for synonyms.

Agreement	Ann 2: Yes	Ann 2: No
Ann 1: Yes	92	18
Ann 1: No	274	616

Table 6: Annotator agreement for type-of.

Table 8 shows the precision-recall figures for Feature-Of, Action-On and Type-Of. Table 9 shows the precision-recall figures for Synonym discovery using only Random Indexing (RI) approach with different features. Table 10 shows the precision-recall comparison of the full Domain Cartridge (using RI + HITS + Similarity Eqn.) with other systems for Synonym discovery.

Relations in BabelNet come either from Wikipedia hyperlinks or WordNet. However the relations obtained from Wikipedia are unlabeled. Therefore, the performance of BabelNet is the same as WordNet for relation discovery (ignoring the unlabeled relations). Yago uses "wikipedia:redirect" links to detect synonymous concepts. "A redirect is a page which has no content itself, but sends the reader to another page, usually an article or section of an article." [50]. Possible reasons for re-directions are misspellings, alternative names, closely related names, abbreviations etc.

WordNet contains a number of relations defined over the taxonomy from which we pick the following relations:
1. *Hyponymy:* X is a hyponym of Y if X is a (kind of) Y.
2. *Hypernymy:* Y is a hypernym of X if X is a (kind of) Y.
3. *Meronymy:* X is a meronym of Y if X is a part of Y.
4. *Holonymy:* Y is a holonym of X if X is a part of Y.

Hyponyms and Hypernyms are same as Type-Of in our work, whereas Meronyms and Holonyms form a subset of Feature-Of relations in our work (which could detect only 1 Feature-Of relation in our experiment). BabelNet, WordNet and Yago do not have any category similar to Action-On in our work. Yago has Type-Of categories derived from WordNet hyponymy and hypernymy relations, as well as from the Wikipedia categories intended to group together pages on similar subjects in Wikipedia. Table 11 shows the recall comparison of all the systems for the discovery of Type-Of, Feature-Of and Action-On relations.

A number of similarity measures have been defined over the WordNet taxonomy that exploit distributional similarity to find the relatedness of 2 concepts. We consider the following similarity measures from [38] as our baseline for Synonym discovery by the distributional similarity approach:
1. *HSO* - Two lexicalized concepts are semantically close if their WordNet synsets are connected by a path that is not too long and that "does not change direction too often"
2. *LCH* - This measure relies on the length of the shortest path between two synsets for their measure of similarity. They limit their attention to 'IS-A' links and scale the path length by the overall depth 'D' of the taxonomy
3. *LESK* - The relatedness of two words is proportional to to the extent of overlaps of their dictionary definitions

Agreement	#Word Pairs	True	False
Feature-Of	500	328	172
Action-On	500	297	203

Table 7: Annotation statistics for feature-of and action-on.

Relation	Precision	Recall
Feature-Of	74.9%	85.7%
Action-On	63.88%	68%
Type-Of	57%	77%

Table 8: Precision-Recall of Domain Cartridge for 3 relations.

System	Precision	Recall
Domain Cartridge (DC)	14%	79%
WordNet	83%	31%
DC + ESG Parser Features	17%	58%
DC + WordNet	25%	48%

Table 9: Precision-Recall of Domain Cartridge for synonyms using only random-indexing.

4. *WUP* - The Wu & Palmer measure calculates relatedness by considering the depths of the two synsets in the WordNet taxonomies, along with the depth of the LCS
5. *RES* - Resnik defined the similarity between two synsets to be the information content of their lowest super-ordinate (most specific common subsumer)
6. *JCN* - It uses the notion of information content, but in the form of the conditional probability of encountering an instance of a child-synset given an instance of a parent synset: $1/jcn\_distance$, where $jcn\_distance = IC(synset_1) + IC(synset_2) - 2 * IC(lcs)$
7. *LIN* - The math equation is modified a little bit from JCN: $2 * IC(lcs)/(IC(synset_1) + IC(synset_2))$, where $IC(x)$ is the information content of 'x'. One can observe, then, that the relatedness value will be greater-than or equal-to zero and less-than or equal-to one

Table 12 shows the F-score comparison of different WordNet similarity measures with Domain Cartridge. Figure 5 shows a snapshot of constructed smartphone domain ontology with our system.

7. DISCUSSIONS

Domain term discovery, of which multi-words form an important component, forms the primary module of the Domain Cartridge framework. Domain terms extracted using noun phrase chunking on the document titles are used to enrich the parser lexicon to bootstrap the parser which attains 32% recall on ground-truth constructed from the back-of-the-book index. Furthermore, domain terms are extracted using HITS on a bipartite graph representation of the SSR and concepts on the entire corpus, which further improved the recall by 8%. The domain term discovery process achieves 18% improvement in recall over WordNet, which lacks most of the multi-word tokens. In fact, the ESG Parser + HITS discovers 1586 multi-word domain terms in the corpus, in contrast to 46 multi-word tokens identified by WordNet. However, BabelNet and Yago that incorporate both WordNet and Wikipedia information obtain a better recall at the cost of human annotation.

The newly discovered domain terms, mostly multi-words, result in 7% and 4% improvement at recall@1 and recall@2 respectively, in an in-house Question-Answering system on the same corpus.

System	Precision	Recall	F-Score
Yago	37.67%	31.60%	34.37%
BabelNet, WordNet	83%	31%	45.14%
Domain Cartridge (DC)	58%	41%	47.6%
DC + WordNet	62%	40%	49%
DC + ESG Parser Features	65%	39%	49.14%

Table 10: Precision-Recall comparison of Domain Cartridge (random-indexing, HITS and sim. eqn.) with other systems.

System	Type-Of	Feature-Of	Action-On
BabelNet, WordNet	19.27%	-	-
Yago	25.12%	-	-
Domain Cartridge	77%	85.7%	68%

Table 11: Recall comparison of systems for 3 relations.

WordNet	F-Score
LCH	0.22
RES	0.31
JCN	0.42
PATH	0.42
LIN	0.43
WUP	0.43
LESK	0.45
Domain Cartridge	0.49

Table 12: F-Score comparison of WordNet similarity measures with Domain Cartridge.

This results from a better parser performance, as the multi-words are used to enrich the parser lexicon, which is evident from less number of incomplete parses and reduction in the parse cost. The number of incomplete parses went down by 73% after incorporating the multi-word domain terms in the parser lexicon. For example, the sentence "Use Sprint Zone" is parsed as "*Use-Noun Sprint-Verb Zone-Noun*" in absence of multi-word information resulting in an incomplete parse. Once the parser has the knowledge that "Sprint Zone" is a multi-word domain term, the parse is complete ("*Use-Verb & Sprint Zone-Noun*").

We primarily focus on 4 relations in the constructed ontology using Domain Cartridge framework namely, *Synonyms ('similarity'), Type-Of ('hierarchical'), Action-On ('functional')* and *Feature-Of ('attributes')*. The Synonym discovery approach uses Random Indexing (RI) for dimensionality reduction and computes the top K neighbors of a given word based on *relational distributional similarity* of the word with the candidate terms using ESG parser features. This method achieves a high recall (79%) but low precision (14%) due to sparse matches, as many neighboring words sharing similar or direct SSR are treated as similar. For example, given the target word 'browser', RI retrieves "opera-mini, ebook, apps, messaging, load, application" *etc.* as candidate similar terms due to overlapping SSR in the context. In the second stage for Synonym discovery, we use a weighted relational distributional similarity measure on these candidate terms using dominant domain SSR extracted using HITS. This significantly increases precision, as we consider the discriminative power of the matched SSR in the similarity equation, and consider only dominant SSR as given by HITS, achieving an F-Score of 47.6%.

It is observed from the annotated data in the smartphone domain that 84% instances are not Synonyms, which indicates the challenge in Synonym discovery. The performance is further improved to 49.14% using parser features in the form of attributes. A number of similarity measures (Lin, Lesk, LCH, Res, Path, WUP etc.) have been built over WordNet, using the path between concepts, taxonomy depth, information content of nodes *etc.* All the methods achieve a high recall but low precision; due to less coverage but more accurate due to manual construction of the taxonomy, with the best F-Score attained at 45% (prec/recall - 83%/31%). Yago achieves an F-Score of 34.37% in discovering Synonyms; whereas Domain Cartridge achieves an F-Score of 49% for Synonym discovery, without any supervision at all.

BabelNet uses only the semantic relations in WordNet, and therefore has the same performance as WordNet in relation discovery.

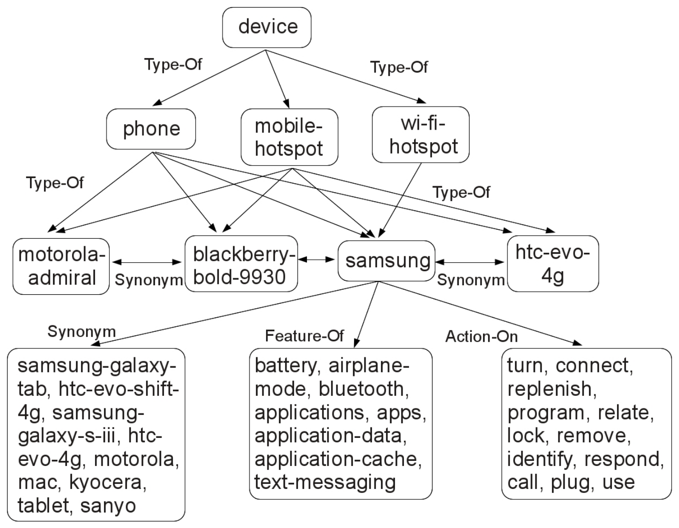
device
Type-Of
Type-Of
phone
mobile-hotspot
wi-fi-hotspot
Type-Of
Type-Of
motorola-admiral
Synonym
blackberry-bold-9930
samsung
Synonym
htc-evo-4g
Synonym
Feature-Of
Action-On
samsung-galaxy-tab, htc-evo-shift-4g, samsung-galaxy-s-iii, htc-evo-4g, motorola, mac, kyocera, tablet, sanyo
battery, airplane-mode, bluetooth, applications, apps, application-data, application-cache, text-messaging
turn, connect, replenish, program, relate, lock, remove, identify, respond, call, plug, use

Figure 5: Snapshot of constructed smartphone domain ontology using Domain Cartridge.

Yago and WordNet do not contain any relation similar to Action-On in Domain Cartridge. The only relations corresponding to Feature-Of in WordNet i.e. Meronymy and Holonymy, that form a subset of Feature-Of, hardly detect any positive examples in the annotated data. Yago does not have any Feature-Of relation category. The WordNet relations, Hyponymy and Hypernymy, that correspond to Type-Of in our framework have a recall of 20%. Yago has a recall of 25% for Type-Of corresponding to the "wikipedia:redirect" links and the WordNet relations, in contrast to the Domain Cartridge recall of 77%. Overall Domain Cartridge achieves an F-Score of 80%, 66% and 65.5% for Feature-Of, Action-On and Type-Of respectively. The SSR features prove to be very effective in capturing direct relation between domain terms. The Type-Of relations are typically detected using a modification of the Hearst [19] patterns on the ESG parser output.

8. RELATED WORK

8.1 Automatic Ontology Creation

An ontology can be seen as a data structure that specifies terms, properties and relations among them for a richer knowledge representation. Many of the existing approaches to create ontology rely on manual or supervised techniques. Due to supervision they are time-consuming, resource-demanding and difficult to scale.

YAGO [48] is a light-weight, extensible ontology, with concepts discovered from Wikipedia and unified with WordNet, using a carefully designed combination of rule-based and heuristic methods.

BabelNet [36] is a large multilingual semantic network that incorporates lexicographic and encyclopedic knowledge from WordNet and Wikipedia. It connects concepts and named entities in a very large network of semantic relations. It uses machine translation to enrich resources from all languages. Relations in BabelNet come either from Wikipedia hyperlinks or WordNet. However, the relations obtained from Wikipedia are unlabeled.

A deep NLP-based system is described in [35] that automatically extracts and populates domain-specific ontologies from morphological structures in free text. It starts with a small seed of domain concepts, performs a graph-based pattern discovery from text and finds taxonomic relations between the concepts in the current ontology. The authors manually provided 48 patterns for identifying type-of and part-of relations, together with WordNet and Wikipedia for synonym discovery. Jaguar [2] automatically builds domain-specific ontologies from text. It uses well-formed procedures to impose a hierarchical structure on the discovered concepts using the semantic relations discovered by Polaris [33] and WordNet [32].

Terminae method [4] is used for building ontological models from text using linguistic analysis. An expert is required to select the most important concepts for the targeted ontology from the list of candidate terms discovered by the tool. Lexical knowledge resources are used to generate domain ontologies from text documents in [29]. User intervention is required at the end of the process to select the relevant concepts and relations. The authors in [21] generate an ontology based on an analysis of a set of texts followed by the use of WordNet. The analysis of the corpus retrieves words as concepts. These words are then searched in WordNet to find concepts associated with these words. A modified version of SOTA algorithm is used to extract terms from documents grouped hierarchically in [22]. Concepts are assigned to the tree nodes based on WordNet hyponymy relation. The authors in [24] use WordNet as a general ontology to discover a subset of concepts to build a domain ontology; whereas [34, 1] use WordNet and a supplementary module to look for missing or associated concepts in the Web. User intervention is required to remove noise. A method for ontology merging based on concepts using WordNet is described in [8].

The USP system [41] can extract formulas from corpus but is limited to extractions for which there is substantial evidence in the corpus. The knowledge extracted is simply a large set of formulas without ontological structure. They build further on the USP semantic parser by adding the capability to form hierarchical clusterings of logical expressions, linked by IS-A relations [42].

As we can see, most of the automatic methods for ontology creation heavily depend on some manually created lexical resources like WordNet, or a seed set of manually provided terms and relations which are expanded to form the full domain ontology, or highly rule-based (as we will also see in the next section). Most of the approaches do not deal with *multi-word* domain terms. In this work, we propose a framework that does not rely on any form of supervision. In the following subsection, we talk of the different approaches to discover specific type of relations for ontology.

8.2 Relation Discovery

The most common method of discovering similar words from text uses the principle of *distributional hypothesis* [17] which says that words which occur in similar contexts tend to have similar meanings. There have been many proposals for computing distributional similarity of words [20, 40, 28]. Our approach of identifying the context is similar to [15, 20, 44, 28] in the usage of a parser, which identifies the context of a word to consist of words connected to the target word by important parser relations. However, our similarity computation differs in the following way: 1) Random Indexing is used for dimensionality reduction which is scalable (in contrast to Latent Semantic Indexing [27]) 2) The similarity computation of the context vectors uses a specialized scoring mechanism on dominant domain relations extracted from a bipartite graph representation of the parser relations and domain terms.

Inference rules from text are automatically discovered using similar paths in dependency trees in [28]. The authors in [26] used relations extracted from [6] to discover concepts from corpus. Simple patterns are used to discover taxonomic relations from Web in [25].

The authors in [19] use manually discovered lexical patterns to detect Hypernyms. These patterns (*Hearst*) suffered from limited recall, and local nature of the patterns introduced errors. The recall is improved in [43] by refining the rules to increase coverage and using supervised classification on the Hearst patterns. The authors in [46] discovered a large number of weak patterns to detect Hypernyms, using WordNet, from a corpus of NewsWire sentences. They achieved a high precision but low recall with an F-score of 0.348. More recently, the authors in [37] applied the techniques described in [46] to detect Hypernyms of Named Entities (*i.e.* Proper Nouns) to improve performance of Question Answering systems. They obtained 53% MAP and the automatically discovered Hyponyms resulted in a 9% performance boost on a TREC Question Answering data set. A method of discovering Hyponymy relation is described in [51] by combining Wikipedia and other Web documents using distributional similarity and hierarchical distances in the Wikipedia database.

The authors in [3] use a method similar to [19] for obtaining Meronym and Holonym relationships, whereas the work in [11, 12] use a set of manually devised patterns to discover part-whole relationships from text.

Most of the works, discussed so far, either use a manually constructed resource to bootstrap the classifier, or manually provided lexical patterns which suffer from the sparsity of patterns in real life texts, and noisy output due to sparse matches. Although, we do use some rules on the parser output for relation discovery (in the form of shallow semantic relations from the parser), they are limited in nature and work at the conceptual level rather than at the lexical level.

9. CONCLUSIONS

In this work, we propose an unsupervised framework for constructing a shallow domain ontology from corpus. Unlike many other existing approaches like Yago and BabelNet, we do not make use of external knowledge resources like WordNet, Wikipedia etc. or manually provided data in the form of seed words or relations. The first part of this work deals with discovering domain terms from corpus using noun phrase chunking and a bipartite graph of shallow semantic relations and domain terms. Multi-words form an important component of the domain term discovery process. We show that the incorporation of these multi-words in the parser lexicon improves the parser performance, with 73% reduction in the number of incomplete parses and better parser output, which improves the performance of an in-house Question-Answering system by upto 7%.

The second part of this work focuses on creating a shallow ontology using domain terms, and relations like *Synonyms ('similar-to'), Type-Of ('is-a'), Feature-Of ('attributes')* and *Action-On ('methods')*. We show that the Synonym discovery approach, using a modified *relational distributional similarity* measure on dominant domain SSR and weighted evidence measure, performs better than most of the existing approaches to Synonym discovery using Wordnet, BabelNet, and Yago. All the discovered domain terms and relations (using SSR) are evaluated on manually annotated data, where we achieve better performance over the compared resources, without using any mode of supervision.

An important future work is to measure the reduction in customization time required to adapt the QA system from one domain to another by integrating Domain Cartridge in its pipeline. The advantage of the framework is that we need to change only the Primary Index for migrating to some other domain, keeping the remaining part of the framework intact.

10. REFERENCES

[1] E. Agirre, O. Ansa, E. H. Hovy, and D. Martinez. Enriching very large ontologies using the www. In *ECAI Workshop on Ontology Learning*, 2000.

[2] M. Balakrishna, D. I. Moldovan, M. Tatu, and M. Olteanu. Semi-automatic domain ontology creation from text resources. In *LREC*, 2010.

[3] M. Berland and E. Charniak. Finding parts in very large corpora. ACL '99.

[4] B. Biebow and S. Szulman. Terminae: A linguistic-based tool for the building of a domain ontology. In *EKAW*, 1999.

[5] J. Carletta. Assessing agreement on classification tasks: the kappa statistic. *Comput. Linguist.*, 1996.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. H. Jr., and T. Mitchell. Toward an architecture for never-ending language learning. AAAI, 2010.

[7] E. Charniak. A maximum-entropy-inspired parser. NAACL 2000.

[8] M. Cho, H. Kim, and P. Kim. A new method for ontology merging based on concept using wordnet. In *ICACT 2006*.

[9] S. T. Dumais. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38, 2004.

[10] J. Fan, A. Kalyanpur, D. Gondek, and D. Ferrucci. Automatic knowledge extraction from documents. *IBM Journal of Research and Development*, 56(3.4), 2012.

[11] R. Girju, A. Badulescu, and D. Moldovan. Learning semantic constraints for the automatic discovery of part-whole relations. In *NAACL '03*, 2003.

[12] R. Girju, A. Badulescu, and D. Moldovan. Automatic discovery of part-whole relations. *Comput. Linguist.*, 32(1), 2006.

[13] D. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. A. DubouÃl', L. Zhang, Y. Pan, Z. Qiu, and C. Welty. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, 56(3), 2012.

[14] J. Gorman and J. R. Curran. Random indexing using statistical weight functions. EMNLP '06.

[15] G. Grefenstette. *Explorations in Automatic Thesaurus Discovery*. 1994.

[16] B. Hajian and W. Tony. A method of measuring semantic similarity using a multi-tree model proceedings. In *ITWP'11*, 2011.

[17] Z. Harris. Distributional structure. *Word*, 10(23), 1954.

[18] C. Havasi, R. Speer, and J. Alonso. Conceptnet 3: a flexible, multilingual semantic network for common sense knowledge. In *Recent Advances in Natural Language Processing*, 2007.

[19] M. A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. ACL, 1992.

[20] D. Hindle. Noun classification from predicate-argument structures. ACL '90.

[21] H. Hu and D.-Y. Liu. Learning OWL ontologies from free texts. volume 2, 2004.

[22] L. Khan and F. Luo. Ontology construction for information selection. ICTAI '02.

[23] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46, 1999.

[24] H. Kong, M. Hwang, and P. Kim. Design of the automatic ontology building system about the specific domain knowledge. In *ICACT 2006*, volume 2, 2006.

[25] Z. Kozareva and E. Hovy. A semi-supervised method to learn and construct taxonomies using the web. EMNLP '10.

[26] J. Krishnamurthy and T. Mitchell. Which noun phrases denote which concepts? In *ACL*, 2011.

[27] A. Kumaran, R. Makin, V. Pattisapu, S. E. Sharif, G. Kacmarcik, and L. V. Automatic extraction of synonymy information: An extended abstract.

[28] D. Lin and P. Pantel. Dirt – discovery of inference rules from text. In *KDD*, 2001.

[29] D. Lonsdale, Y. Ding, D. W. Embley, and A. Melby. Peppering knowledge sources with salt: Boosting conceptual content for ontology generation. In *In AAAI Workshop for Semantic Web Meets Language Resources*, 2002.

[30] M. C. McCord, J. W. Murdock, and B. K. Boguraev. Deep parsing in watson. In *IBM Journal of Research and Development*, volume 56, 2012.

[31] S. Mcdonald and M. Ramscar. Testing the distributional hypothesis: The influence of context on judgements of semantic similarity. In *23rd Annual Conference of the Cognitive Science Society*, 2001.

[32] G. A. Miller. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM*, 38, 1995.

[33] D. Moldovan and E. Blanco. Polaris: Lymba's semantic parser. In *LREC'12*, 2012.

[34] D. I. Moldovan and R. Girju. Domain-specific knowledge acquisition and classification using wordnet. In *FLAIRS Conference*, 2000.

[35] H. Mousavi, D. Kerr, M. Iseli, and C. Zaniolo. Ontoharvester: An unsupervised ontology generator from free text. 2013.

[36] R. Navigli and S. P. Ponzetto. ACL '10.

[37] P. S. Paul McNamee, Rion Snow and J. Mayfield. Learning named entity hyponyms for question answering. In *IJCNLP*, 2008.

[38] T. Pedersen and S. Patwardhan. Wordnet::similarity - measuring the relatedness of concepts. 2004.

[39] J. K. Pentti Kanerva and A. Holst. Random indexing of text samples for latent semantic analysis. *22nd Annual Conference of the Cognitive Science Society*, 2000.

[40] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *ACL*, 1993.

[41] H. Poon and P. Domingos. Unsupervised semantic parsing. In *EMNLP*, 2009.

[42] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *ACL*, 2010.

[43] A. Ritter, S. Soderland, and O. Etzioni. What is this, anyway: Automatic hypernym discovery. AAAI, 2009.

[44] G. Ruge. Experiment on linguistically-based term associations. *Inf. Process. Manage.*, 28(3), 1992.

[45] M. Sahlgren. An introduction to random indexing. *Methods and Applicatons of Semantic Indexing Workshop in ICTKE*, 2005.

[46] R. Snow, D. Jurafsky, and A. Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*. 2005.

[47] S. Staab and R. Studer. *Handbook on Ontologies*. Springer Publishing Company, Incorporated, 2nd edition, 2009.

[48] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. WWW '07.

[49] O. V. and A. P. Ontology based semantic similarity comparison of documents. In *Proc. of IEEE 14th workshop on database and expert systems applications*, 2003.

[50] Wikipedia. Wikipedia, the free encyclopedia, 2013.

[51] I. Yamada, K. Torisawa, J. Kazama, K. Kuroda, M. Murata, S. De Saeger, F. Bond, and A. Sumida. Hypernym discovery based on distributional similarity and hierarchical structures. EMNLP '09.