SAHYADRI International Journal of Research, Vol. 3, Issue 2, 2017
ISSN: 2456-186X, Published Online December, 2017 (http://www.sijr.in/)

Research Paper

Ontology Based Search of Document Repositories - Correction of Terms and Ontology Learning - A keyword Based Approach

Jatin Talati[1] and N. Balaji[2*]
[1]IBM Research India Pvt. Ltd. Bangalore, India
[2]ISE Dept., Sahyadri College of Engineering & Management, Adyar, Mangaluru - 575007
*Email: balaji.hiriyur@gmail.com

Abstract

Information Retrieval system focuses on retrieving the set of documents relevant to a query. Over the years, the information available has increased rapidly. Also, because of the unstructured nature of the information, it is very difficult for the users to retrieve the relevant information. Ontologies are an efficient way of storing the real world entities and the relationships between them. It is a key structure that guides the semantic browsing of the information. In this paper, we have used ontology to represent the information for a domain specific information retrieval system. To improve the results of a search system LUCENE, a full text search engine, we consider the semantic information about the query keywords by augmenting the related terms from the ontology. The set of documents retrieved as results are the rendered back to the user.

The domain ontology used for augmenting the query might not be available readily. Thus, in this paper we have mentioned a semi-automated partial ontology generation process that creates an ontology from the glossary or using definitions from *WordNet*. From the glossary, a topic modelling tool (Latent Dirichlet Allocation-LDA) extracts a set of initial words which are further used for the relationship extraction process. Relationship extraction phase extracts the synonym, antonym and hypernym relations from the glossary or *WordNet*. After that it extracts object properties (called predicates) between two keyword(s) and form triples in the form (*subject* − *predicate* − *object*). These set of triples (including the synonym, antonym and hypernym relations) are then converted to a Controlled Natural Language (Attempto Controlled English-ACE).

Keywords: IR System, Spell Check, Ontology Creation, Latent Dirichlet Allocation, OWL DL, Attempto Controlled English, LUCENE.

1 Introduction

1.1 Overview

The basic goal of Information Retrieval (IR) system is retrieving the relevant documents from a set of documents given a query. The document corpus can contain the information about various heterogeneous domains and does not stick over a particular domain. Over the years, the volume of information available has increased at a rapid rate. Also, the unstructured nature of the information has made it difficult for the users to retrieve the relevant information. Thus, the IR task has become very crucial these days. Various techniques have been proposed to overcome the issue of unstructured nature and the huge volume of information. However, no technique has solved this problem completely [1-4].

This problem has a direct relevance to the operation of domain specific search queries where the user is aware of the background knowledge for searching the corpus of a particular domain. For example, within a domain of computer science, searching for a query like *Tree Data Structure*, the end-user being aware of the domain, might get some other results than what he/she actually expected. A structured representation of the data also plays an important role in this kind of domain specific searches. Thus here ontology is used as a platform for representing the data [5-7].

Ontology is a key structure that guides the semantic browsing for the document set exploration. The formalization of ontology is a means to facilitate the management of concepts as objects as well as their relationships as properties. In other words, an Ontology is a structured representation of data where objects are referred as concepts and their relationships as properties. Ontology can be domain specific or can be generic so that a domain unaware user can also take the benefit of the representation of the ontology. There has been a lot of contributions from ontologies to the IR system.

- Ontology can improve precision and recall: They are precise enough to provide some unique information about the query that is not explicitly given by the end-user.
- With the help of ontology, users can express their needs more easily.
- Ontology can include the information about various heterogeneous sources.

In this paper, we have demonstrated an IR system which uses ontologies to augment the query and improve the results of search. We have used a full-text search engine LUCENE as baseline for the basic search operation. On top of LUCENE, we have implemented our system that uses LUCENE search function to search in the document corpus and rank them. It starts with creating an ontology from the document corpus that will be used for augmenting the query keywords by their semantically similar words present in the ontology. The user will enter some keywords to be searched as query which will pass through an analyzer phase for the correction of spelling mistakes, if any. In this paper, we have described a spell check system based on *Bayesian Noisy Channel* [8] spell checker and it is described in further section.

After correction, the query will pass through the expansion phase where it will get augmented by some semantically similar words in the ontology. After the aggregation of query, these keywords will be searched into the index made by LUCENE. From the set of documents, LUCENE creates an inverted index that contains the document ID's corresponding to each word which is present in that particular document. The searcher phase of LUCENE will search the query keywords using the index and retrieve the set of resultant documents. This will be ranked according to the ranking function which includes the boost factor given to each of the terms present in the query. Finally, the top-k documents from the ranked documents are returned back to the user as result.

1.2 Spell Check System

The spell checker module checks and corrects the spelling mistakes, if any in the query keywords entered by the user. This module implements the spell check from [8] that uses *Bayesian* classification technique for identifying the correct word from corpus. It uses *n-gram* techniques for checking whether the query keyword is present in the dictionary. It is assumed that all the words in the document corpus are error free and we use these words to create the dictionary. If the word is not present in the dictionary, a list of candidate words will be extracted from the dictionary which are one edit distance from the query keyword and scores them. Then the top scored word is replaced by the erroneous word in the query.

The spell check system is a part of LUCENE's analyzer phase which parses the query keywords. The query will pass through this analysis phase for the correction of the spelling errors as well as for *stemming and lemmatization*. These steps are required to create the inverted index. For example, the words *dancing*, *danced* will be stemmed to the word *dance* in the index.

1.3 Ontology Creation

It is the process of creation of ontology by taking the domain terms and extracting relationships between them using some natural language corpus and encoding them with an ontology language. Generally, the ontology has to be authored manually by some domain experts. Constructing ontology manually is an extremely labor-intensive and time-consuming task. In this paper, we have proposed a technique to partially automate the ontology learning procedure. This subsystem takes a set of domain words as input and extract the relationships between them from the corresponding natural language corpus using *parts-of-speech tagging and phrase chunking*.

Also, from document corpus, the system will extract some important set of words, using topic modelling. These set of words are the initial set of words for extraction of relationship from the document corpus. For topic modelling, we have used *Latent Dirichlet Allocation (LDA)* that uses *Word2Vec* for representation of documents and then extract top-k words based on some threshold.

1.4 Major Contribution

- The system has a domain specific spell checker that checks and corrects the spelling error in the query.
- A new semi-automatic ontology learning procedure has been introduced.
- System uses ontology to aggregate the query which improves the search results rather than direct search on LUCENE.

2 Background and Literature Survey

2.1 Background

Information Retrieval stands for retrieving relevant information from a corpus of information resources. The search can be either full-text search or some content based semantic search. The basic IR system must have a representation of data, a system to represent user queries as well as a searcher that searches the query on the representation. In our system, we are using a full-text search engine LUCENE which retrieve a document if the query keyword is present in it.

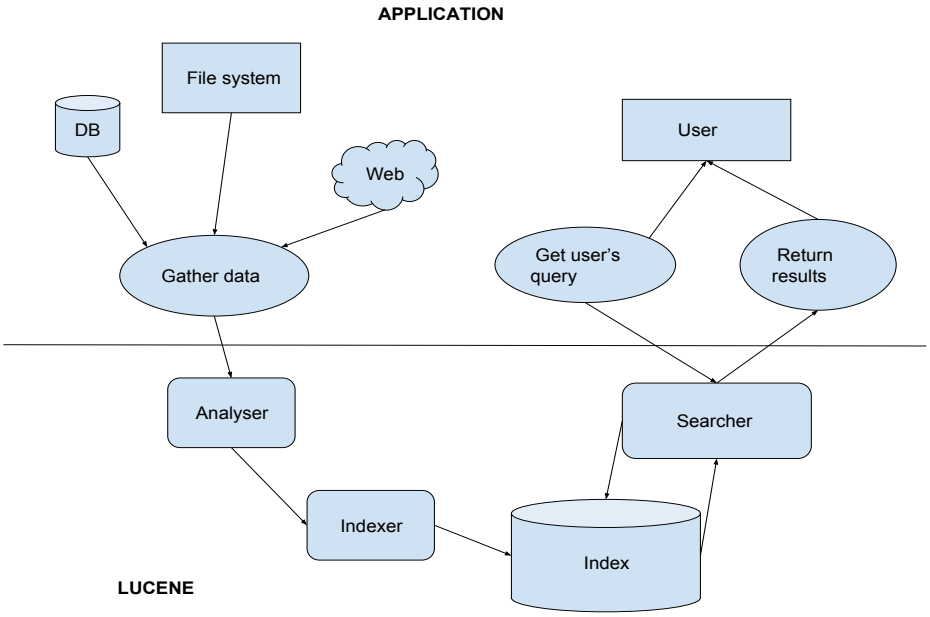

Figure 1: LUCENE System Architecture.

2.1.1 LUCENE

LUCENE, being a full-text search engine, matches exactly the query keywords to the documents. It creates an inverted index from the set of documents and then carries out the search in the inverted index. The basic architecture of LUCENE is explained in the Figure 1.

LUCENE's architecture consists of the following tasks to perform search:

1. **Analyze the document:** For creating the index document has to be analyzed. This is done by LUCENE's analyzer phase. It consists of the following tasks to analyze the document as well as the query:
   - **Stemming and Lemmatization:** This step bring all the words to their root word to create an index.
   - **Parts of Speech Tagging:** It is done to differentiate between noun and verb senses of the same word.
2. **Create an index:** In this phase an index needs to be created, that will be used to perform the search. *Indexer* phase of LUCENE will perform this task and it creates an inverted index that matches a word to the document it is present.
3. **Interface for user to enter the query:** Once the index is ready, there must be an interface where the user enters the query.
4. **Build Query:** The application must create a query object that can be used to inquire the index. The *Query Parser* phase of LUCENE does this part building a Boolean query consisting of Boolean clauses.
5. **Search Query:** This phase must perform the search using index and retrieve relevant documents. LUCENE's *searcher* phase implements this task and performs search using the inverted index created in the previous phases. This will retrieve the documents which are matching to user query and rank them in order of the number of query terms matched.
6. **Render Results:** The documents that are retrieved then returned back to the user in the order of rank.

We are using complete LUCENE system as our baseline and on top of it building our own semantic search system that uses ontology to improve the search results.

2.2 Literature Survey

A lot of research work has been done before in the area of semantic search using machine learning approaches whereas many techniques used lexical representation of the data. In the problem of semantic search, the representation of the data plays a crucial role. Thus, a good representation along with a equally good algorithm gives better results for an IR system [9].

In [10] the authors describe the functioning of a system of effective and efficient entity search in Resource Description Framework (RDF) data. RDF is a data representation technique where the data is stored in the form of triples as *(subject-predicate-object)* where the subject and object are the entities related with a relationship given by the predicate. The authors also

have implemented an index structure that can be used for entity search in RDF data. This index structure stores all the triples in three categories viz *topic*, *predicate* and *object* and matches the position of each topic to each predicate position. However, this index structure is very inefficient in case of increasing amount of data as we cannot store all the triples as index as well as the information about their position.

Fetahu et. al. [3] have proposed a machine learning approach for improving entity search. Assuming the RDF data entity centric, they are clustering the entities based on their similarity, using a combination of lexical and structural features. The intuition behind doing this is they assume that semantically related data would belong to the same cluster. This is not necessary for textual data as the text actually from different topics may also fall into the same cluster. Another disadvantage of this approach is that it is supervised which may not be the case with textual data always.

Blanco et. al. [10] provide users the solution for exploring a domain using a domain specific ontology as an information representation. The relations and axioms in ontology gives us the platform to look for concepts which have not been explicitly written in the query. The hypothesis behind using this domain, ontology are more relevant for document set exploration as they are constructed from the text analysis of the documents. Also, ontology provide a rich background from various heterogeneous sources [11].

From the papers discussed till now it is clear that domain ontology are most prominent technique for the representation of data in any IR system. They also help users in expressing their needs in a more useful manner so that they can be precise and accurate while querying. We searched for many domain specific ontologies but there are very few available for our work. So, in this paper, We have demonstrated a partially automated ontology generation module that generates domain ontology using document corpus by extracting similarities and relations from the corpus.

Thanopoulos et. al. [12] address the problem of extracting similarities via statistical processing of the data. They used the sequential context, *Word-based* or *Pattern-based* of the words to extract word clusters which are semantically similar, as well as the similarity relations between them. This process is domain independent and is therefore based on minimal resources [13].

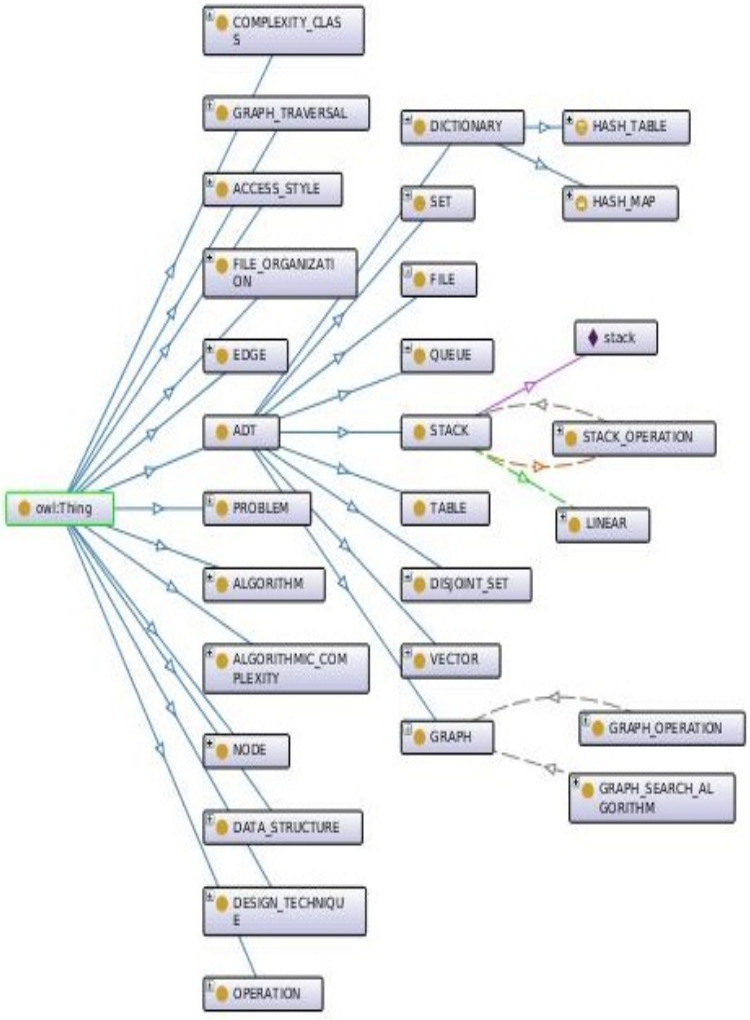

Figure 2: Data Structure and Algorithms Ontology.

In [14], the authors are trying to extract the taxonomy relationships from the document corpus and web sites. Here, authors are trying to construct a hypernym graph from the automated extraction of terms, definitions and hypernyms which are learned using a tool named *OntoLearn*. However, the complete process is Unsupervised, resulting in poor quality definitional patterns.

Some machine learning approaches were also used for Ontology Learning as in [9], the authors use *Deep Learning* technology as it has been proved very beneficial in natural language processing tasks. Here, Ontology learning problem have been tailored as a transductive reasoning task that converts the knowledge from natural language to logic-based specification. They trained a *Recurrent Neural Network (RNN)* model to extract OWL formulae from text. The model is not using any NLP tool for training. Also, *ACE Parsing Engine (APE)* is used for constructing the OWL statements.

2.3 Ontology

Ontology is a data representation technique that represents the real world entities as concepts and their relationships as predicates. In other words, it is the complete specification of conceptualization. We are using ontology for representing the domain knowledge for LUCENE search application where the data is represented in the form of triples *(subject-predicate-object)*. In this triple, subject and object are the real world entities or concepts in the domain and the predicates are the relationships extracted from the domain corpus.

Many ontologies are available related to a particular domain. Here is an example of Data Structure and Algorithms [15] ontology consisting of the terms related to data structures and algorithm domain. This ontology contains 1441 total axioms out of which 424 are logical axioms. There are 107 classes in this ontology related to each other by 26 object properties and having 13 data properties. The ontology have total 154 instances that specifies the classes.

The representation of *DSA* ontology in *Protege* editor is depicted in Figure 2. Protege is a tool for representing the ontology as concepts and properties in a structured way so that by storing minimum information we can get the maximum output.

From Figure 2 it is clearly visible that the basic entities in the domain, such as *ADT*, *Algorithm*, *Complexity class*, etc. are the main concepts in the ontology. All the other concepts, such as *Graph*, *Stack*, *Table*, etc. are the sub-classes of these main concepts. They are also related to each other by some relationships using arrows. The subclass relationship is shown as a blue arrow towards the subclass. The dotted lines denotes that the concepts are related to each other by a predicate.

2.3.1 OWL

Web Ontology Language (OWL) is a knowledge representation language for authoring of ontology. Also it is a computational logic based language, so that it can be exploited by the computer programs for the creation and verification of the knowledge. It uses URI's to uniquely identify the concepts and properties in the ontology. Ontology also allows us to put some restrictions on classes and relationships using a set of axioms so that they can be made more specific.

2.4 Attempto Controlled English (ACE)

ACE is a subset of English language with some restricted syntax, semantics and a small set of construction and interpretation rules. It appears to be very natural to human beings and thus is very easily readable and understandable by any English speaker. The benefit of ACE is, it has some standard construction rules which can be useful for the interpretation of the language and extract OWL statements from it. Some of the examples of ACE sentences are as follows:

1. Every man is a human.
2. A man is not a woman.
3. Every aunt is the sister of a mother or a father.

3 System Architecture

An Information Retrieval process starts after a query is entered by user. The query keywords are further sent to the system that parses the query and carries out the search in the index created using the document corpus and retrieve the relevant documents. In this system we are using *LUCENE* as the baseline system and we are constructing a search system on top of it. Any Information Retrieval system is supposed to have the following basic things:

- The representation of the content;
- A system for representation of user's information need;
- A system for searching of the user query in the document corpus.

Here, we are using *LUCENE* to perform the search operation and for the indexing process. Ontology is used for representing domain specific knowledge. The architecture of the system and the working of all the modules is being described in further sections.

3.1 Design

System architecture is a conceptual model that describes the behaviour as well as the structure of the system.

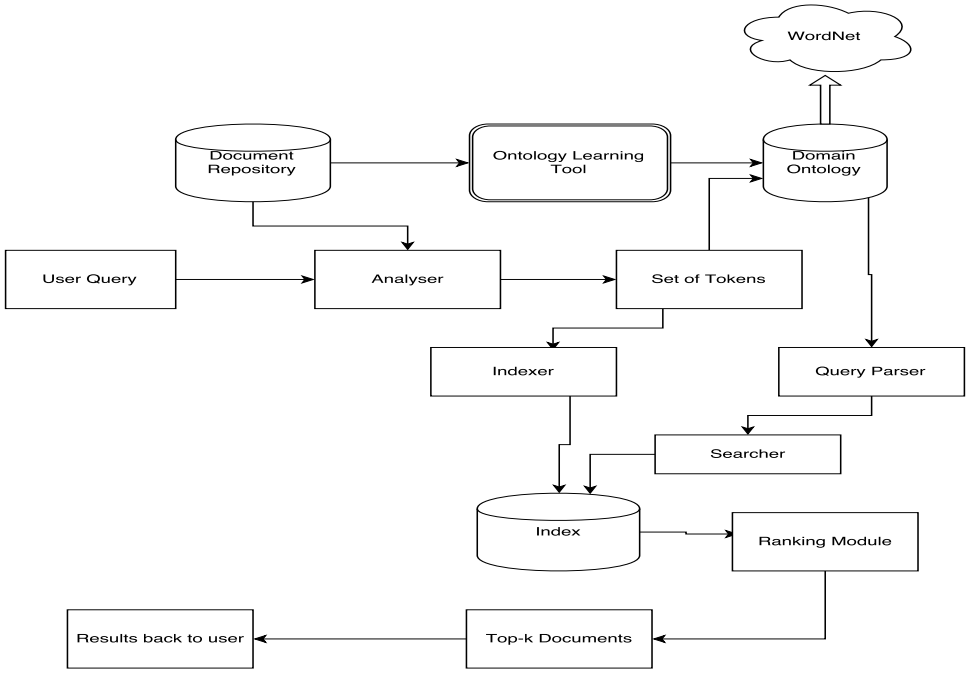

Figure 3: Architecture of Information Retrieval system consisting of all the modules.

It also includes the data as well as control flow of the system. The basic architecture of our system is described in Figure 3 followed by the description of each modules.

3.2 Components of Architecture

Each of the components describes data as well as control flow over the whole system. The various components involved in the IR architecture are as follows:

Document Repository: This contains all the documents contained in the enterprise. Each document has a unique ID which will be helpful in identifying the document. This document corpus would be useful for making the index. For this, the document corpus will pass through an Analysis phase for further processing. In the first part, we assumed that the domain ontology is given as an input by the user. However, to reduce user's overload in the second part, we implement an ontology learning module which partially automates the generation of ontology and decreases the labour of creating ontology manually.

Ontology Learning Tool: This tool generates an ontology from the document corpus. It creates parts of the domain ontology which can further be extended by a domain expert. We have mainly three phases of Ontology Learning:

- **Topic Modelling:** The document corpus will be passed through a topic modelling tool which will extract important terms which describes the corpus. Here, LDA is used for extracting the most relevant topics from the set of documents. This set of words extracted will be the initial set of keywords to be used for relation extraction.
- **Ontology Generation:** This phase takes the triples generated in the previous phase as input and generate an *Attempto Controlled English (ACE)* sentence. Then parse it using APE to make an *OWL* statement.

This phase will generate a partial ontology that can further be extended by a human expert. After being through the domain expert, this ontology will be used for query augmentation for this search system.

User Query: User will give his/her information need in the form of a natural language query which is passed on to the analysis phase for further processing. The query could be any natural language sentence, a keyword or a question according to the user.

Analysis: Various analysis processes with the query will be done in this phase. This phase consists of the following things to be done before the query goes for expansion:

- **Spell Check:** Search to be done in the corpus, the query needs to be checked for spelling errors. This module will correct the spelling errors and then passes the query for further analysis. After the correction of errors, query keywords will go to the next analyzer phase for analysis.
- **Parts-Of-Speech Tagging:** The documents coming to the analysis phase will be given parts-of-speech tags which are needed to make the index.
- **Stemming and Lemmatization:** It is a process of retrieving head-words. It is required for making an index as the words present in document collection may contain words in different tenses.

Set of Tokens: It contains the set of tokens given by analysis phase after processing of the documents or query.

From here the documents will be sent to indexer while the query will be sent to ontology for *aggregation*.

Indexer: The set of tokens will be helpful to make the index for searching. The index will be created using the traditional method of accessing all the terms in the documents and then going through all the documents and finding a set of documents it is present.

Index: This is an index made by the indexer for the set of documents that are present in the document corpus. The index is known as inverted index as it contains the mapping of the terms to the set of document ID's corresponding to the document the term is present.

Domain Ontology: This phase consists of domain Ontology constructed by Ontology Learning phase. It will be useful for augmenting the query by the terms similar to the query terms. It will also take the help of *Word Net* for extracting synonyms of the query terms. Because, it may happen that the original term is not present in the query but the synonym is present.

Query Parser: This phase parses the expanded query coming from the domain ontology. Parsing of a query consists of giving weights to the query terms according to their relevance to the original query. The process of giving weights includes the modification of ranking function of LUCENE which includes addition of a boosting factor for each term. Then this weighted term query will be converted to a boolean query that makes use of connectives as *AND* and *OR* only then sent to searcher.

Searcher: It will accept the Boolean query and then match the query terms with index structure. All terms connected with *AND* must be present in the document, then only the document will be considered for scoring. Among all the terms connected with *OR*, if any one of the terms is present then the document will be retrieved for ranking process. All the document ID's retrieved in this phase are then sent to the Ranking Module for the ranking of the document.

Ranking Module: It will take the input, set of document IDs, and then will perform ranking of documents according to the modified ranking function of LUCENE.

4 Spell Check and Ranking Function

This section talks about spell checker module which corrects the spelling as well as context errors in the query keywords entered by the user. Then the corrected query will pass through query aggregation phase where it will be aggregated by the terms in ontology which are semantically similar to it. This augmented query will pass through the query parser phase that outputs a Boolean query that will be searched in the index by *Searcher*. The documents that are retrieved as a result of search will be ranked by the ranking function using the boost factor that has been given to the terms. The ranking function and the boost factor is implemented in further sections.

This phase contains two types of spell check viz word spell check, which checks for spelling errors in the single word and phrase spell check that checks and corrects the context errors.

4.1 Spell Check Module

This module will accept the keywords from the user and correct the misspelled words using the dictionary of correct words taken from the document corpus. Before creating the dictionary, it is assumed that all the words in document corpus are error free and can be used for correction.

To create the dictionary, document corpus is first passed through some pre-processing steps where all the files required to train the spell checker model are created. The algorithm for pre-processing of data is explained in the algorithm-1.

Algorithm 1 pre-processing(document_set)

1: Merge all the data files into one text file (say *data.txt*)
2: Extract all the correct words into a single file excluding the stop-words
3: Extract uni-grams with their frequencies (*unigram.txt*)
4: Extract bi-grams with their frequencies (*bigrams.txt*)
5: Extract tri-grams with their frequencies (*trigram.txt*)
6: Extract for each word, the nearby 3 words on left and right each. These will be the context words for each word.
7: Extract the similar sounding words from the document corpus using Phonetics Algorithm.
8: Create 4 matrices for addition, subtraction, deletion and substitution and write these into files.

The spell checker module is trained using the files created in the pre-processing step. The dictionary will be used for the correction of misspelled terms in the query. Also, the dictionary for similar sounding words will be created that will be used for correction of context errors. The four matrices for each of the edit operations are standard matrices that denotes the number of times the operation has taken place for each character. These matrices have taken from *1988 News wire Corpus* mentioned in [8]. The matrices consists of the average of the number of times the errors made by user while querying the corpus.

Following are the two phases of spell checker module in the system:

1. Word Spell Check: This phase checks for the spelling error of a single word in the query keywords.
2. Phrase/Sentence Spell check: This phase checks for the whole sentence for the context errors after each word is spelt correctly and it is also known as *context sensitive spell check*.

4.2 Word Spell Check

The query keywords entered by the user are parsed and each word is checked for the error. A word w is taken and first matched with the terms present in the dictionary using the *n-gram* techniques. *n-gram* technique checks for the *n-grams* of w in the set of words containing the corresponding *n-gram*. The most frequently used *n-grams* are *bi-grams* or *tri-grams*. If the word w is not present in the set, then it is declared as incorrect word or a typo and then sent for correction. This process is done for all the words in the keyword set entered by the user.

For correction, the typo is first checked against the words which differ from it by a single operation (insertion, deletion, substitution, transposition).

- *uni-gram dictionary:* It contains the uni-grams \$ their corresponding set of words in which that particular uni-gram is present.
- *bi-gram dictionary:* This dictionary contains the bi-grams and their corresponding set of words in which the particular pair of characters occur together.
- *tri-gram dictionary:* It contains the tri-grams and their corresponding set of words in which the particular tri-gram is present.

After extracting the candidate words, they will be given a score according to the *Baye's Theorem*. It says that the probability of a word w being correct, given a typo t, can be computed as the probability of the typo t being generated from the correct word w, multiplied by the probability of the correct word. This method is defined more precisely in [8] and it can be expressed mathematically in equation.

$$Pr(w|t) = \big[Pr(t|w) * Pr(w)\big] \tag{1}$$

Here the prior $Pr(w)$ is computed as the frequency of the word w ($f(w)$) in the dictionary out of the total frequency of words (N) in the dictionary as in equation.

$$Pr(w) = \left[\frac{f(w)}{N}\right] \tag{2}$$

The conditional probabilities, $Pr(t|c)$ are computed using four different matrices (computed in the pre-processing step) that gives the probabilities for the four operations respectively. These four matrices are of 26×26 dimensions containing the alphabets on each row and column respectively. The elements of the matrix represent the frequency of corresponding edit on the respective row and column characters in the correct word.

1. *del[x, y]* gives the frequency of the characters in the correct word, xy being written as x in the training set.
2. *add[x, y]* gives the frequency of the character x mistakenly written as xy.
3. *sub[x, y]* gives the frequency of the character x incorrectly replaced by y.
4. *trans[x, y]* gives the frequency of the characters xy in the correct word being written as yx.

The matrices are calculated in the Pre-processing step and this data will be used to calculate the probabilities for the corresponding edit.

Here the above probabilities will be calculated according to the correct word w and the typo t, that is $Pr(t|w)$, is shown in equations 3, 4, 5 and 6:

If any character is mistakenly inserted in the typo, then:

$$Pr(t|w) = \left[\frac{add(w_{p-1}, w_p)}{chars(w_{p-1})}\right] \tag{3}$$

If any character is mistakenly deleted in the typo, then:

$$Pr(t|w) = \left[\frac{del(w_{p-1}, w_p)}{chars(w_{p-1}, w_p)}\right] \tag{4}$$

If the character is mistakenly substituted by other character in the typo, then:

$$Pr(t|w) = \left[\frac{sub(t_p, w_p)}{chars(w_p)}\right] \tag{5}$$

If the characters mistakenly exchanged their positions in the typo, then:

$$Pr(t|w) = \left[\frac{trans(w_p, w_{p+1})}{chars(w_p, w_{p+1})}\right] \tag{6}$$

where,
chars(w_p) $\leftarrow$ frequency of the p^{th} character of the correct word in the document corpus.
chars(c_{p-1}, c_p) $\leftarrow$ bi-gram frequency of $(p-1)^{th}$ character and the p^{th} character of the correct word.
These probabilities will be the score given to that particular word. The top scored word is returned back to the interface and it replaces the incorrect word in the query keywords.

The algorithm used for word spell check is described by *Algorithm-2*.

$$Score(q, D) = \sum_{t \in q} \left[\left[(tf(t \ in \ D) * idf(t)^2 * t.boost() * norm(t, D) \ big) \right] * coord(q, D) * queryNorm(q) \right] \quad (7)$$

Algorithm 2 Word Spell Check

```
query ← input query from user
bigram_dict ← import bigram dictionary
for each term t in query do
    t_bigram ← first bigram i.e. first 2 characters of term t
    if t_bigram not in bigram_dict then
        correct_word ← find_correct_word(t)
        replace the term t with correct_word in the query
    end if
end for
return query
```

Algorithm-3 will explain the working of *find_correct_word* module and this module will be called when the given word is incorrect.

Algorithm 3 find_correct_word(word)

```
list_candidates ← list of correct words with edit distance k
prob_candidates ← { }
for w in list_candidates do
    find the position and the operation between w and word where the two are different.
    prob ← Pr(word|w) which will be calculated using the conditional probabilities.
    prob_candidates.append(w, prob)
end for
return the top word from prob_candidates
```

4.3 Phrase/Sentence Spell Check

All words in the query keywords are correctly spelt but the context in which they are occurring might be incorrect. Reason can be the correct word was replaced by a similar sounding word or some *Optical Character Recognizer(OCR)* has failed to correctly identify the word. For generating the similar sounding words, *Soundex* footnotehttps://en.wikipedia.org/wiki/SoundexReferences algorithm is used. It will take a word as input and the output will be the list of words which sounds similar to the input word.

For example, *piece* has two similar sounding words as {*piece*, *peace*}. Suppose a phrase comes as *"peace of cake"*, the algorithm will detect a spelling mistake by taking each word of the phrase as pivot and computing the probability of it occurring with the context. The word giving the minimum probability will be the erroneous word.

The complete algorithm for phrase spell check is described in *Algorithm-4*:

Algorithm 4 Phrase Spell Check

```
word ← incorrect word from the phrase
context_words ← words within the range of ± 3
simlar_words ← the similar sounding words from the phoenetics algorithm
prob ← { ··· }
for sw in similar_words do
    replace the incorrect word by sw in context_words.
    prob.append(probability of the context_words)
end for
correct_context ← max(prob)
correct_word ← the word with max probability in the context
replace the incorrect word in the phrase with the correct_word in the query
return the query
```

After this the query is now error free and can be passed on for aggregation to the ontology.

4.4 Ranking Module

LUCENE uses a ranking function which matches the query term word to word by the index structure. The documents matched are considered for ranking based on the relevance to the query terms.

The boolean query represented in the form of *conjunction or disjunction*, will be the input for this function. The query terms will then be searched for in the index and all the relevant documents will be retrieved. The ranking of the retrieved documents will be according to the ranking function shown in the equation 7, where,
Score(q, D) ← score a document will be given against a query Q,
idf(t) ← inverse document frequency i.e. the number of documents the term t is present.

idf(t) is calculated as shown in equation 8.

$$idf(t) = log \left[\frac{numOfDocs\_t\_isPresent}{docFrequency \ + \ 1} \right] + 1. \quad (8)$$

t.boost() represents boost factor of the term t would be given i.e., the weight of a term according to the relevance to the query.

This is the boost factor we are adding to the query terms in order to rank the documents according to their relevance to the query. That means the original query

which is given by user will be given higher weight because the user is interested in the document containing that particular term.

After the original query, the second highest weight would be given to the *siblings* of the original terms extracted from the ontology because they are much more related to the original term and also the query keyword might not be present in the document but the *siblings* may be present.

Then the similar weights will be given to the *subclass* and *super class* of the terms in ontology as they are relevant to the query but are more specific and more generic to the original query.

norm(t, D) is a *field length normalization factor* measuring the importance of the term in the query. It is implemented as shown in equation 9,

$$norm(t, D) = \left[\frac{1}{sqrt(frequency\ of\ t\ in\ D)}\right] \quad (9)$$

coord(q, D) is the *coordination factor* gives the number of terms of the query found in the document D.

It can be implemented as shown in equation 10,

$$coord(q, D) = \left[\frac{intersection}{max\ Intersection}\right] \quad (10)$$

Here, *intersection* is the number of query terms matching to the document D and *max Intersection* is the maximum number of query terms matching in a particular document.

queryNorm(q) is the *normalization factor* so that all queries can be compared and the same can be computed as in equation 11,

$$queryNorm(q) = \left[\frac{1}{sqrt(sumOfSquareWeights)}\right] \quad (11)$$

After the ranking of the document set retrieved, the top-k documents will be rendered back to the user. These documents may contain relevant as well as non-relevant documents.

4.5 Ontology Learning

This section consists of constructing parts of the ontology using document corpus, a domain glossary as well as *WordNet*. The construction of ontology manually is a labour-intensive as well as time consuming task. So in this section we have tried to automate the ontology authoring process using the keywords from domain and extracting the relations between them using their definitions. This process will create a basic ontology that can be further converted to a full-fledged ontology by a human expert. This basic ontology can be inconsistent in the structure as well as the relationships that are extracted.

There are many tools which create the ontology using the taxonomic relations as discussed in the earlier section. In addition to that our system is extracting non-taxonomic relations also which can be used as object properties in the ontology. Figure 4 represents the flow of data across these modules.

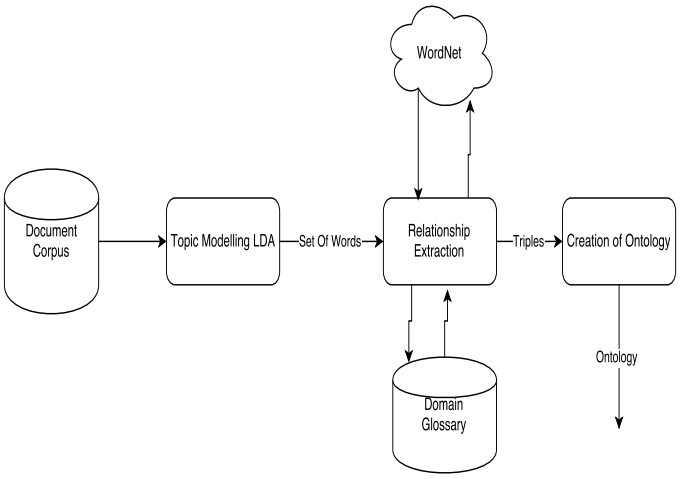

Figure 4: Architecture of Ontology Learning Module.

Ontology learning process is divided into the following three main modules which are essential for creating the ontology:

- Topic Modelling
- Relationship Extraction
- Creation of Ontology

4.5.1 Topic Modelling

In this step, input will be the document corpus. We will perform *topic modelling* on these set of documents. For performing topic modelling, we have used *Latent Dirichlet Allocation(LDA)*, that uses *Word2Vec* technique to represent the documents and extracts the most representative words among the document set. To perform LDA, we are using *gensim* library which is open source and can be used to perform basic *LDA* operations.

LDA assumes that a document is a mixture of topics and by representing the documents as vectors, the most representative set of topics can be extracted. The output here will be a set of words which will be further used to extract the relationships in the next phase.

Word2Vec represents the documents as a vector space typically of hundreds of dimensions with each unique word assigned to a vector in the space. These words are placed in the vector space such that the related words in the documents are located in the close proximity to each other. Also, it is assumed that high frequency words, such

as stop words do not give much information about the domain. Thus the words with frequency above a certain threshold can be ignored to speed up the process.

In our system, to avoid the inclusion of high frequency words, we are eliminating the stop words from the document corpus before applying *LDA* on it. Also, we are performing *stemming* so that different senses of word result to the same stemmed word.

4.5.2 Relationship Extraction

Using the set of words extracted in the previous step, the current phase extracts the relationships between these set of words using the document corpus *WordNet* or some domain specific *glossaries*. The output of this phase will be a set of triples *(subject, predicate, object)* where subject and object will be the words from the set or new words that are extracted along with the relationships and the predicate will be the relationship extracted between these words.

4.5.3 Ontology Generation

In this phase the input will be triple generated in the previous phase. It will be parsed and converted to *Attempto Controlled English(ACE)* format which is useful to generate OWL statements. The *ACE* sentences formed will be further converted to *OWL* statements using *APE*.

For each triple (subject-predicate-object), *APE* will create an IRI for subject, relation and object. IRI's are unique ID given to a concept or a property to uniquely identify it in the ontology. *APE* translates the *ACE* statements unambiguously to various formats as *discourse representation structure (DRS) or OWL*, that uses *first order logic(FOL)*. Converting the text into first order logic allows the user to verify, validate, as well as to query it.

So triples coming from the previous phase are parsed and then converted to any one of the *ACE* acceptable sentences. The following are the 4 cases which we have identified in the triple while converting it into *ACE* format.

1. **Case 1:** If two concepts (say *AandB*) are *synonyms* to each other then *ACE* sentence would be formed as: *Every A is a B* and *Every B is a A* which implies that both *A* and *B* are equivalent classes.

2. **Case 2:** If the two concepts are *antonyms* in the document corpus to each other then the *ACE* sentence would define that they are disjoint classes as: *No A is a B.*

3. **Case 3:** If the two concepts are related with a subclass *is a*, relationship then the *ACE* statement would define the subclass relationship as: *Every A is a B* which indicates that every instance of *A* is also an instance of *B* .

4. **Case 4:** Else the two concepts might be related to each other by a relationship other than synonyms, antonyms and subclass, then the *ACE* sentence representing the relatedness of the two words would be: for example, *A wife is married to a husband.*

Thus, all the triples are then mapped to one of the cases and then the corresponding *ACE* statements formed will be parsed using *APE* which in result will convert them into OWL statements. All the OWL statements are written into a file by *APE* and then the ontology is constructed.

5 Experimental Results

Many experiments were performed mainly on the three domains viz *Family*, *Data Structures and Algorithms* and *Environment*. Further sections describe the data set and the process how it is prepared followed by experiments performed and their results.

5.1 Data Sets

We have worked mainly in the three domains viz *Family*, *Data Structure and Algorithms* and *Environment*. All the documents related to these domains were taken from the Internet. As the system is domain specific, there is no standard data set available for the system. The corpus differs from user to user. So we have prepared the data set from various sources. For family domain, *WordNet* worked as the major source for definitions of the words. The corpus for family domain consists of 30 files each containing some definitions about the words in the domain.

For Data Structures and Algorithms, the documents are being prepared from the Wikipedia articles as well as some books such as [5] and [4]. For this, we have prepared around 150 documents containing the definitions of various data structures such as tree, graph, etc., traversal techniques, and many algorithms. We have created some irrelevant documents also.

All the data taken from above sources are then pre-processed into different formats according to their usage in different algorithms. For example, the Ontology Learning procedure requires the data to be in the text format only.

5.2 Ontology Learning Module

The OL module consists of the two main working modules *(Topic Modelling and Creation of OWL file)* which are experimented on the above three domains and the results are shown below.

5.2.1 Topic Modelling

In topic modelling, *LDA* is used for the extraction of topics as explained in the previous chapters. The

experiments are performed for different number of topics (keeping the number of words in each topic constant) and the relevant topics among them were taken for the next process. The number of words in each topic doesn't affect the relevant topics to be extracted.

The keywords that are extracted will be the input for the next phase that is *Relation Extraction* phase where each of the keywords will be checked for a definition within the corpus or outside the corpus using some domain *glossary or WordNet*. A keyword may give a relationship with another keyword present in the initial set or it may extract the relationship with a new keyword which is not present. This new keyword can be added as a concept to the initial set. The irrelevant topics which do not belong to the domain are removed using some similarity function. The report of [11] explain the process of relationship extraction in detail as shown in Table 1.

Table 1: LDA for family, DSA and Environmental domain.

Family Domain	
No. of Topics	Relevant Topics
5	4
10	8
15	10
20	13
25	13
30	13
35	13
40	14

DSA Domain	
No. of Topics	Relevant Topics
5	5
10	9
15	9
20	12
25	13
30	14
35	14
40	14

Environment Domain	
No. of Topics	Relevant Topics
5	5
10	9
15	13
20	13
25	16
30	21
35	22
40	22

5.2.2 Ontology Generation

After extraction of relationships, the output will be a set of triples *(subject-relation-object)* where subject and object are the keywords in the set and the relation is extracted in the previous phase. This triple will be the input for this phase which is Creation of an OWL statements. For this, the triple will be converted first to ACE format. For example, the triple *(mother, is a, woman)* gets converted to ACE format as *Every mother is a woman.* Similarly, all the ACE sentences will be formed according to the cases observed in the previous chapters and then these sentences will be parsed by APE which will convert these into OWL file.

This system provides better results compared to the normal *LUCENE* results as it is taking the semantics of the data into consideration. Here, we also proposed the spell check system for the LUCENE, which helps in better analysis of the search system.

The search system would be domain specific as the ontology can not be general enough for storing the whole background knowledge. The user also must have some knowledge about the domain in which the system is getting deployed. User will get the top-k documents according to his/her search query.

References

[1] R. Andrew Golding and Dan Roth, "A bayesian hybrid method for context sensitive spelling correction", Machine Learning - Special issue on natural language learning, vol. 5629, no.1, pp. 39-53, 1993.

[2] Nathalie Aussenac-Gilles and Josiane Mothe, "Ontologies as Background Knowledge to Explore Document Collections", RIAO '04 Coupling approaches, coupling media and coupling languages for information retrieval, vol. 9, no. 6, pp. 129-142, 2004.

[3] Besnik Fetahu, Ujwal Gadiraju and Stefan Dietze, "Improving entity retrieval on Structured data", Proceedings of 14^{th} International Conference on The Semantic Web, ISWC, vol 9366, no. 01, pp. 474-491, 2015.

[4] Cormen and H. Thomas, "Introduction to Algorithms", McGraw-Hill, 2013.

[5] Dasgupta, Sanjoy, Papadimitriou, H. Christos, Vazirani and Umesh, "Algorithms", McGraw-Hill, Inc, 2006.

[6] Hatcher, Erik, Gospodnetic and Otis, "Lucene in action", Manning Publications, 2004.

[7] Knublauch, Holger, Fergerson, W. Ray, Noy, F. Natalya, Musen and A. Mark, "The Protégé OWL plugin: An open development environment for semantic web applications", International Semantic Web Conference, vol. 3298, no. 1, pp. 229-243, 2004.

[8] D. Mark Kernighan, W. Kenneth Church and A. William Gale, "A spelling Correction Program Based on a Noisy Channel", COLING '90 Proceedings of the 13^{th} conference on Computational linguistics, vol. 2, no. 2, pp. 205-210, 1990.

[9] Petrucci, Giulio, Ghidini, Chiara, Rospocher and Marco, "Knowledge Engineering and Knowledge Management", 20^{th} International Conference, EKAW, Bologna, Italy, vol. 10024, no. 1, pp. 2016, Proceedings 20, November 2016.

[10] Roi Blanco, Peter Mika and Sebastiano Vigna, "Effective and Efficient entity search in RDF data", ISWC'11 Proceedings of the 10^{th} international conference on The semantic web, vol. 7031, no. 1, pp. 83-97, 2016.

[11] B. Salomi Shalini, "Ontology Based search on Domain Specific Document Repository-Query expansion and Ontology Learning", M.Tech Report, Dept. of Computer Science and Engineering, vol. 17, no. 1, pp. 23-44, IIT Madras, 2017.

[12] Thanopoulos, Aristomenis, Fakotakis, Nikos, Kokkinakis and George, "Automatic Extraction of Semantic Relations from Specialized Corpora", Proceedings of the 18^{th} Conference on Computational Linguistics, vol. 2, no. 1, pp. 2020-2026, 2000.

[13] Umar Manzoor, Bassam Zafar, Hafsa Umar and Shoaib Khan, "Semantic Image Retrieval: An ontology based approach", International Journal of Advanced Research in Artificial Intelligence(IJARAI), vol. 4, no. 4, pp. 1-8, 2015.

[14] Velardi, Paola, Faralli, Stefano, Navigli and Roberto, "Ontolearn reloaded: A graph-based algorithm for Taxonomy Induction", Computational Linguistics, vol. 39, no. 3, pp. 665-707, 2013.

[15] E. V. Vinu, "Data Structures and Algorithms ontology", Research Scholar, Dept. of Computer Science and Engineering, IIT Madras, 2017.