Indian Journal of Research in Applied Sciences Engineering (IJRASE)
Vol.2.No.1 2014pp 52-56.
available at: www.goniv.com
Paper Received :05-03-2014
Paper Published:28-03-2014
Paper Reviewed by: 1. John Arhter 2. Hendry Goyal
Editor : Prof. P.Muthukumar

IMPLEMENTING A HIGHER QUALITY INPUT PHRASE TO DRIVEN REVERSE DICTIONARY

E.Kamalanathan[1] and C.Sunitha Ram[2]
Department of Computer Science and Engineering,
SCSVMV University Enathur
kamalanathane@gmail.com[1] csunitharam@kanchiuniv.ac.in[2]

ABSTRACT

Implementing a higher quality input phrase to driven reverse wordbook. In contrast to a conventional forward wordbook, that map from word to their definitions, a reverse wordbook takes a user input phrase describing the specified construct, and returns a group of candidate words that satisfy the input phrase. This work has important application not just for the final public, notably those that work closely with words, however conjointly within the general field of abstract search. The current a group of algorithms and therefore the results of a group of experiments showing the retrieval accuracy and therefore the runtime latency performance is implementation. The experimental results show that, approach will offer important enhancements in performance scale while not sacrificing the standard of the result. Experiments scrutiny the standard of approach to it of presently on the market reverse dictionaries show that the approach will offer considerably higher quality over either of the opposite presently on the market implementations.

Keywords: Dictionaries, thesauruses, search process, web-based services.

1. INTRODUCTION

A Report work on creating a reverse dictionary, As against a regular (forward) wordbook that maps words to their definitions, a WD performs the converse mapping, i.e., given a phrase describing the required conception, it provides words whose definitions match the entered definition phrase.
It's relevant to language understanding. The approach has a number of the characteristics expected from a strong language understanding system. Firstly, learning solely depends on unannoted text information, which is abundant and contain the individual bias of an observer. Secondly, the approach is predicated on all-purpose resources (Brill's PoS Tagger, WordNet [7]), and also the performance is studied below negative (hence additional realistic) assumptions, e.g., that the tagger is trained on a regular dataset with doubtless totally different properties from the documents to be clustered. Similarly, the approach studies the potential advantages of victimization all potential senses (and hypernyms) from WordNet, in an endeavor to defer (or avoid altogether) the necessity for Word Sense Disambiguation (WSD), and also the connected pitfalls of a WSD tool which can be biased towards a particular domain or language vogue

The user input is unlikely to precisely match (indeed, would possibly disagree wide from) the definition of a word within the forward lexicon. for instance, a user could enter the phrase "to waste resources on unimportant things" once searching for an inspiration like "fritter," whose definition may be "spend frivolously and unwisely"— that is conceptually similar, however doesn't contain any of an equivalent words because the user input.

A implemented only the forward mechanism to search for the Keyword in the dictionary. The forward dictionary that maps words given by the user to their definitions. The two most common methods to achieve latent semantic indexing (LSI) and principal component analysis (PCA), both analyze the keywords of documents in a corpus to identify the dominant concepts in the document. In most implementations of CSP (concept similarity problem) solutions, vectorization is done a priori, and at runtime, only vector distances are computed. Concepts are represented as vectors in a feature (or keyword) space.

1.1 Background:

Natural Language Processing: Natural Language Processing (NLP) [6] is a large field which encompasses a lot of categories that are related to this thesis. Specifically NLP is the process of computationally extracting meaningful information of natural languages. In other words: the ability for a computer to interpret the expressive power of natural language. Subcategories of NLP which are relevant for this thesis are presented below.

WordNet: WordNet [7], [2] is a large lexical database containing the words of the English language. It resembles the traits of a thesaurus in that it structures words that have similar meaning together. WordNet is something more, since it also specifies different connections for each of the senses of a given word. These connections place words that are semantically related close to one another in a network. WordNet also displays some excellence of a dictionary, since it describes the definition of words and their corresponding part-of-speech.

Synonym relation is the main connection between words, which means that words which are conceptually equivalent, and thus interchangeable in most contexts, are grouped together. These groupings are called synsets and consist of a definition and relations to other synsets. A word can be part of more than one synset, since it can bear more than one meaning. WordNet has a total of 117 000 synsets, which are linked together. Not all synsets have a distinct path to another synset. This is the case, since the data structure in WordNet is split into four different groups; nouns, verbs, adjectives and adverbs (since they follow different rules of grammar). Thus it is not feasible to compare words in different groups, unless all groups are linked together with a common entity. There are some exceptions which links synsets cross part-of-speech in WordNet, but these are rare. It is not always possible to find a relative between two words within a group, since each group are made of different base types. The relations that connect the synsets within the different groups vary based on the type of the synsets. The most used relation connecting synsets is the hypernym/hyponym relation, which specifies "*IS-A*" relations. The easiest way to capture the nature of these relations is to think of them as taxonomies. It then becomes evident that hyponym relations are transitive i.e. all dogs are canines and all golden retrievers are canines. In terms hypernyms are more standard than their hyponyms, which are more specific.

The coordinate term is easy to understand from the above example, since both wolf and dog shares the hypernym canine. The holonym/meronym relation connecting noun synsets specifies the part-whole relation. These relations are also called "*HAS-A*" relations and inherits from their superordinate. Properties are inherited downward and show that the meronym is part of the holonym. The reverse is not necessarily true i.e. a building is not part of a window. The troponym relative is the manner in which something is being done. These relate to one another in the way they are performed i.e. to yell is to communicate in some manner. Specificity is inherited downward, thus the more general terms are super ordinate. Entailment describes dependencies. By doing something you must also be doing something else i.e. by driving you must also be moving. Adjectives are stored together in antonyms, i.e. opposites. These are then linked to semantically alike words. In some sense these semantically related words are antonyms of their counterparts, which they are stored together with. Most adverbs are easily derived from adjectives. WordNet relates these adverbs to adjectives.

WordNet Categories WordNet, the lexical database developed by Miller et al., is used to include background information on each word. Depending on the research setup, words are replaced with their synset IDs, which constitute their different possible senses, and also different levels of hypernyms, more general terms for the a word, are added.

Application Programming Interface

Several Application Programming Interfaces (API) exists for WordNet. These allow easy access to the platform and often additional functionality. As an example of this the Java WordNet Library [8] (JWNL) can be mentioned. This allows for access to the WordNet Library files.

PoS Tagging PoS tags[8] are assigned to the corpus using Brill's PoS tagger. As PoS tagging require the words to be in their original order this is done before any other modifications on the corpora.

Part-of-speech (POS) tagging is the field which is concerned with analysing a text and assigning different grammatical roles to each entity. These roles are based on the definition of the particular word and the context in which it is written. Words that are in close proximity of each other often affect and assign meaning to each other. The POS taggers job is to assign grammatical roles such as nouns, verbs, adjectives, adverbs, etc. based upon these relations. The tagging of POS is important in information retrieval in general text processing. This is the case since natural languages contain a lot of ambiguity, which can make distinguishing words/terms difficult. There are two main schools when tagging POS. These are rule-based and stochastic. Examples of the two are Brill's tagger and Stanford POS tagger, respectively. Rule-based taggers work by applying the most used POS for a given word. Predefined/lexical rules are then applied to the structure for error analysis. Errors are corrected until a satisfying threshold is reached. Stochastic taggers use a trained corpus to determine the POS of a given word. These trained corpuses contain pre-tagged text, which define the correct POS of a given word in a given context. The corpuses are vast enough to cover a large area, which defines different uses of terms. The stochastic tagger retrieves the context of the word in question and relates it to the trained data. A correlation is found by geometric analysis upon the use of the word in the trained data. This means that the content of the trained corpus very much influences the outcome. Trained corpuses should thus be picked, such that reflection of the field they are trying to tag is maximal. Current taggers have a success rate above the ninety-seven percent mark. This is to the extent where even some linguists argue, which is the correct result. It can thus be concluded that these taggers exhibit near human results.

Stopword Removal Stopwords, i.e. words thought not to convey any meaning, are removed from the text. The approach taken in this work does not compile a static list of stopwords, as usually done. Instead PoS information is browbeaten and all tokens that are not nouns, verbs or adjectives are removed.

Stop words are words which occur often in text and speech. They do not tell much about the content they are wrapped in, but helps humans understand and interpret the residue of the content. These terms are so generic that they do not mean anything by themselves. In the context of text processing they are basically just empty words, which only takes up space, increases computational time and affects the similarity measure in a way which is not relevant. This can result in false positives.

Stop words is a broad term and there is no precise requirement of which words are stop words. To specify if a given word is a stop word, it has to be put in context. In some situations a word might carry relevance for the content and in others it may not. This is defined by the area in which the content resides. A stop word list should thus be chosen such that it reflects the field which is being analysed. The words in such a list should be filtered away from the content in question.

This class includes only one method; which runs through a list of words and removes all occurrences of words specified in a file. A text file, which specifies the stop words, is loaded into the program. This file is called "stop-words.txt" and is located at the home directory of the program. The text file can be edited such that it only contains the desired stop words. A representation of the stop words used in the text file can be found in table - 1. After the list of stop words has been loaded, it is compared to the words in the given list. If a match is found the given word in the list is removed. A list, exposed from stop words, is then returned.

a	who	with	these	among	onto
be	the	this	where	although	anybody
but	in	from	those	whether	whenever
person	of	whic	how	else	whereas
some	and	h	during	anyone	
someone	to	whe	much	beside	
too	that	n	upon	whose	
very	for	what	toward	whom	
		than			
		into			

Table: 1 List of Stop words

Stemming Words with the same meaning appear in various morphological forms. To capture their similarity they are normalised into a common root-form, the stem. The morphology function provided with WordNet is used for stemming, because it only yields stems that are contained in the WordNet dictionary.

This class contains five methods; one for converting a list of words into a string, two for stemming a list of words and two for handling the access to WordNet through the JWNL API[8]. The first method *listToString()* takes an *ArrayList* of strings and concatenate these into a string representation. The second method *stringStemmer()* takes an *ArrayList* of strings and iterates through each word, stemming these by calling the private method *wordStemmer()*. This method checks if the JWNL API has been loaded and starts stemming by looking up the lemma of a word in WordNet. Before this is done, each word starting with an uppercase letter is checked to see if it can be used as a noun. If the word can be used as a noun, it does not qualify for stemming and is returned in its original form. The lemma lookup is done by using a morphological processor, which is provided by WordNet. This morphs the word into its lemma, after which the word is checked for a match in the database of WordNet. This is done by running through all the specified POS databases defined in WordNet. If a match is found, the lemma of the word is returned, otherwise the original word is simply returned. Lastly, the methods allowing access to WordNet initializes the JWNL API and shuts it down, respectively. The *initializer()* method gets an instance of the dictionary files and loads the morphological processor. If this method is not called, the program is not able to access the WordNet files. The method *close()* closes the dictionary files and shuts down the JWNL API. This method is not used in the program, since it would not make sense to uninstall the dictionary once it has been installed. It would only increase the total execution time. It has been implemented for good measure, should it be needed.

Stemming[5] is the process of reducing an inflected or derived word to its base form. In other words all morphological deviations of a word are reduced to the same form, which makes comparison easier. The stemmed word is not necessarily returned to its morphological root, but a mutual stem. The morphological deviations of a word have different suffixes, but in essence describe the same. These different variants can therefore be merged into a distinct representative form. Thus a comparison of stemmed words turns up a higher relation for equivalent words. In addition storing becomes more effective. Words like observes, observed, observation, observationally should all be reduced to a mutual stem such as observe.

There are a lot of proposed methods for finding stems. Some examples are; lookup tables, suffix-stripping, affix stemming and lemmatisation. Stemmers can both be language dependant and independent. This is based on how the relevant stemmer is implemented. A lot of work has been put into the area of stemming; some of the more popular stemmers are Porter and Snowball.

2. PROPOSED SYSTEM

Reverse dictionaries approach can provide significantly higher quality. The proposed a set of methods for building and querying a reverse dictionary. Reverse dictionary system is based on the notion that a phrase that conceptually describes a word should resemble the word's actual definition, if not matching the exact words, then at least conceptually similar. Consider, for example, the following concept phrase: "talks a lot, but without much substance." Based on such a phrase, a reverse dictionary should return words such as "gabby," "chatty," and "garrulous."

Upon receipt of a user input phrase, we first find candidate words from a forward dictionary data source, where the definitions of these candidate words have some similarity to the user input. We then rank the candidate words in order of quality of match.

The find candidate words phase consists of two key sub steps:

1) Build the RMS.

2) Query the RMS.

2.1 Components:

The first preprocessing step is to PoS tag the corpus. The PoS tagger relies on the text structure and morphological differences to determine the appropriate part-of-speech. For this reason, if it is required, PoS tagging is the first step to be carried out. After this, stopword removal is performed, followed by stemming. This order is chosen to reduce the amount of words to be stemmed. The stemmed words are then looked up in WordNet and their corresponding synonyms and hypernyms are added to the bag-of-words. Once the document vectors are completed in this way, the frequency of each word across the corpus can be counted and every word occurring less often than the pre specified threshold is pruned.

Stemming, stopword removal and pruning all aim to improve clustering quality by removing noise, i.e. meaningless data. They all lead to a reduction in the number of dimensions in the term-space. Weighting is concerned with the estimation of the importance of individual terms. All of these have been used extensively and are considered the baseline for comparison in this work. However, the two techniques under investigation both add data to the representation. a PoS tagging adds syntactic information and WordNet is used to add synonyms and hypernyms.

Building Reverse Mapping Sets.

Given the large size of dictionaries, creating such mappings on the fly is infeasible. Thus, Procreate these Rs for every relevant term in the dictionary. This is a one time, offline event; once these mappings exist, we can use them for ongoing lookup. Thus, the cost of creating the corpus has no effect on runtime performance. For an input dictionary D, we create R mappings for all terms appearing in the sense phrases (definitions) in D.

RMS Query

This module responds to user input phrases. Upon receiving such an input phrase, we query the R indexes already present in the database to find candidate words whose definitions have any similarity to the input phrase. Upon receiving an input phrase U, we process U using a stepwise refinement approach. We start off by extracting the core terms from U, and searching for the candidate words (Ws) whose definitions contain these core terms exactly. (Note that we tune these terms slightly to increase the probability of generating Ws) If this first step does not generate a sufficient number of output Ws, defined by a tuneable input parameter α, which represents the minimum number of word phrases needed to halt processing and return output.

Candidate word ranking

In this module sorts a set of output Ws in order of decreasing similarity to U, based on the semantic similarity. To build such a ranking, we need to be able to assign a similarity measure for each (S,U) pair, where U is the user input phrase and S is a definition for some W in the candidate word set O.

Forward mapping (standard dictionary): Intuitively, a forward mapping designates all the senses for a particular word phrase. This is expressed in terms of a forward map set (FMS). The FMS of a (word) phrase W, designated by F(W) is the set of (sense) phrases {S1, S2, . . . S_n } such that for each Sj Є F(W_i), ($W_i \rightarrow S_j$) Є D. For example, suppose that the term "jovial" is associated with various meanings, including "showing high-spirited merriment" and "pertaining" to the god Jove, or Jupiter." Here, F (jovial) would contain both of these phrases.

Reverse mapping (reverse dictionary): Reverse mapping applies to terms and is expressed as a reverse map set (RMS). The RMS of t, denoted R(t), is a set of phrases { P1, P2, Pi,……, Pm}, such that ∀Pi ∊ R(t), t ∊ F(Pi). Intuitively, the reverse map set of a term t consists of all the (word) phrases in whose definition t appears.

Semantic Similarity

This algorithm computes a similarity score between two strings, based on how related the contents of these are. All words are put into lists and compared to each other. The ordering of the words is therefore not important. Stop words are taken out of the similarity comparison and do not influence the final result. Furthermore, all the words which are not found in WordNet are likewise taken out of the equation. Words with a capitalized first letter are also taken out if they can be used as a noun. This means that the set of words, which the comparison is being made on, is reduced. In some cases this may increase the similarity score. In others it may reduce the similarity score. The algorithm is dependent on a correct tagging of POS, since only the POS of a word is looked up and compared. With a POS tagging success rate of roughly ninety percent, tagging results are reliable. Since disambiguation is done by comparing fairly short descriptions of senses, the correct designation of the meaning of a word is not expected to be high. In most cases the most common meaning of the word will be given, which is tolerable, since it is most likely the meaning of the word. In situations where a match has been found, a more appealing meaning is given, suiting the interest of comparing words. The best matching's from each set are grouped together and a similarity score is computed by these relations. This means that the text have to be fairly similar in length in order to get a correct similarity score. There can only be a certain amount of matching's between two sets i.e. the number of elements in the smallest set. In reality the length of the texts is not important, but the number of words which are found in WordNet is.

The Semantic Similarity [9] algorithm should produce higher similarity scores than those given by Levenshtein and Dice's Coefficient. That is because, since this algorithm takes the relation between words into account. Words can have different forms, but still express the same thing, thus giving higher likelihood of striking a match. Exceptions are gibberish words, slang, names etc. i.e. words which cannot be found in a dictionary.

2.2 Solution Architecture:

We now describe our implementation architecture, with particular attention to design for scalability. The Reverse Dictionary Application (RDA) is a software module that takes a user phrase (U) as input, and returns a set of conceptually related words as output.

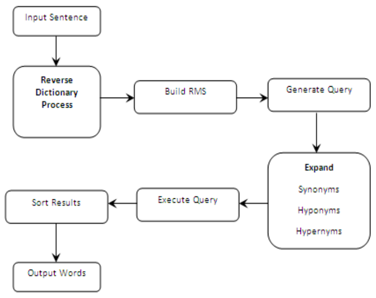

Figure 2.2.1 Architecture of reverse dictionary.

The user input phrase, split the word from the input phrase, perform the stemming. Predict every relevant term in the forward dictionary data source. In the generate query. input phrase, minimum and maximum output thresholds as input, then removal of level 1 stop words (a, be, person, some, someone, too, very, who, the, in, of, and, to) and perform stemming, generate the query. Expand the query find each word's Synonyms, Hyponyms and Hyponyms. The applying OR operation the in query. Execute the query find the set of candidate words. Finally sort the result based on the semantic similarity

2.3 Experimental Environment:

Our experimental environment consisted of two 2.2 GHz dual-core CPU, 2 GB RAM servers running Windows XP pro and above. On one server, we installed our implementation our algorithms (written in Java). The other server housed is wordnet dictionary data.

III. CONCLUSION

We describe the many challenges inherent in building a reverse lexicon, and map drawback to the well-known abstract similarity problem. We tend to propose a collection of strategies for building and querying a reverse lexicon, and describe a collection of experiments that show the standard of our results, similarly because the runtime performance underneath load. Our experimental results show that our approach will give important enhancements in performance scale while not sacrificing answer quality. Our experiments scrutiny the standard of our approach to it of dictionary.com and OneLook.com reverse dictionaries show that the Wordster approach will give considerably higher quality over either of the opposite presently obtainable implementations.

REFERENCES

[1]. D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent Dirichlet Allocation," J. Machine Learning Research, vol. 3, pp. 993-1022, Mar. 2003.
[2]. T. Dao and T. Simpson, "Measuring Similarity between Sentences," 2009. http://opensvn.csie.org/WordNetDotNet/trunk/ Projects/
[3]. T. Hofmann, "Probabilistic Latent Semantic Indexing," SIGIR '99: Proc. 22nd Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval, pp. 50-57, 1999.
[4]. D. Lin, "An Information-Theoretic Definition of Similarity," Proc .Int'l Conf. Machine Learning, 1998.
[5]. M. Porter, "The Porter Stemming Algorithm,"http://tartarus.org/martin/PorterStemmer/ , 2009.
[6]. O.S. Project "Opennlp," http://opennlp.sourceforge.net/, 2009.
[7]. G. Miller, C. Fellbaum, R. Tengi, P. Wakefield, and H. Langone, "Wordnet Lexical Database," http://wordnet.princeton.edu/wordnet/download/ , 2009.
[8]. http://extjwnl.sourceforge.net/
[9]. P. Resnik, "Semantic Similarity in a Taxonomy: An Information-Based Measure and Its Application to Problems of Ambiguity in Natural Language," J. Artificial Intelligence Research, vol. 11, pp. 95- 130, 1999.