A Semantic Search Engine Based On Lexical Frequency Pattern Clustering For Measuring Similarities Between Data Files

Snehika Datla
Department of computer science and engineering,
MVGR college of engineering,
Vizianagaram,
India.

Abstract

Validly measuring linguistic similarities between two entities or words is a difficult task. On a web in various tasks the linguistic affinity between words is a vital component. Relation expulsion, compatriot mining, document clustering, and analogue metadata expulsion are various tasks which performs some affinity between words. An empirical method to estimate linguistic affinity is proposed by page counts and text snippets. By using page counts we can able to get various co-occurrences of word measures and integrate those with lexical patterns extracted from text snippets. To spot the various linguistic relations and its frequency between two given words, we tend to propose a lexical pattern extraction and lexical frequency pattern clustering algorithm. We can determine the page counts co-occurrence measures and lexical frequency pattern clustering using support vector machine. The proposed technique states varied baselines and previously planned web-based linguistic affinity measures on three benchmark knowledge sets showing a high correlation with human ratings and it improves community mining accuracy. The proposed technique shows lexical pattern frequency techniques to measure similarity between words in a data files.

***Key words** – Web mining, frequency pattern clustering, lexical pattern,*

1. Introduction

In any web mining, information retrieval, and natural language processing are the two vital concepts to measure the linguistic affinity between words in a data files. The web mining applications such as, compatriot expulsion, relation detection, and entity clarification require the ability to accurately measure the semantic similarity between concepts. In information retrieval, the disadvantage is to extract a collection of documents that is linguistically interconnected to a user query. In language process the approximation of linguistics affinity between words is extremely tough. A user is sorting out a word apple that is of times found within the internet that isn't conferred in most well liked thesauri or dictionaries. New words square measure often being created and new senses also are being allotted to the present one. To take care of these new words and senses is extremely overpriced. By mistreatment internet program associate analogue methodology will be projected to estimate the linguistics identity between words or entities. It will increase the time whereas analyzing every document on an individual basis as a result of the high accretion rate of the online. In internet program there square measure two helpful information sources. They are:

a. Page Counts, and
b. Snippets.

We can calculate the quantity of pages that contain the query words through the page count. The queried word seems many times on one page, therefore there is no necessary to the page count to be equal to the word frequency. Page count of a query will be considered as a global measure of co-occurrence of words. The page count of the query “apple” AND “computer” in Google is 288,000,000, and for “banana” AND “computer” is only 3,590,000. This shows that apple has more additional frequency with computer when compared to banana. Rather the simplicity of page counts, using this alone as a measure of co-occurrence of two words may cause several drawbacks. Firstly, the position of a word in a very page is analyzed by a page count. Secondly page count of a word with multiple senses may contain a combination of all senses. We can say that page count of apple may contain as a fruit and as a company. Some words may co-occur in the web without any relation [1]. Page counts are inaccurate when measuring linguistic similarities. A search engine extracting a text in a document query term is known as snippets. It provides the helpful information regarding the context of the query term. The linguistic affinity measures outlined over snippets are employed in question enlargement [2], personal Disambiguation [3], and community mining [4]. Snippets process is been useful for time consuming and downloading WebPages. One of the common drawbacks in snippets is that, because of wide range of searching results in web and huge documents it shows only the top-ranking results. Therefore whatever information we need to measure between a given pair of words has no guaranty that it is in top most snippets. We tend to propose a technique that considers both page counts and lexical syntactic patterns extracted from snippets and lexical frequency pattern clustering method that we show experimentally to overcome the above mentioned problems.

2. Existing Systems:

For a given taxonomy of words, there is a blunt method to calculate the affinity between the two words i.e. to find the length of the shortest path connecting two words. When multiple paths exist between the two words there is only one way for calculating affinity i.e., finding the shortest path between two senses. But here the drawback is that the frequency acknowledged with this approach represents the same distance for multiple paths proposed a new measure of linguistic affinity using information content. He defined the maximum of the information content. The concept C includes both the concepts C1 and C2, in taxonomy here the affinity between the two words can be measured as the maximum affinity between two concepts that words belong to. Information content is calculated by using Brown corpus and uses Word Net as taxonomy to combined structural linguistics information from a lexical taxonomy and knowledge content from a corpus during a nonlinear model. They projected a similarity live that uses shortest path length, depth, and native density in taxonomy. Their experiments according a high Pearson coefficient of correlation of 0.8914 on the Miller and Charles [7] benchmark information set. They failed to judge their methodology in terms of similarities among named entities. Sculptor [8] outlined the affinity between two ideas because the info that is in common to each idea and also the information contained in every individual thought. Cilibrasi and Vitanyi [9] projected a distance metric between words using page counts retrieved from an internet programmer. The projected metric is known as Normalized Google Distance (NGD) and is given by

$$NGD(P,Q) = \frac{\max\{\log \mathrm{H(P)}, \log \mathrm{H(Q)}\} - \log \mathrm{H(P,Q)}}{\log N - \min\{H(P), \log H(Q)\}}$$

P and Q: the two words;
NGD (P, Q): the gap between P and Q;
H (P), H (Q): the page count for the word P and Q;
H (P, Q): the page count for the question “P and Q”.

3. Proposed System:

Let us assume that P and Q are two words by which we can perform the linguistic affinity measures by building a function sim(P, Q) that has a value in range [0,1]. If the range is 1 then it says that A and B are similar to each other, If the range is 0 then it states that P and Q are dissimilar. By using page counts and snippets taken from web we can define affinity between P and Q. By using this we can classify the synonymous and disaffinity by training a support vector machine for the word pairs. The function sim(P, Q) is approximation of the trained SVM.

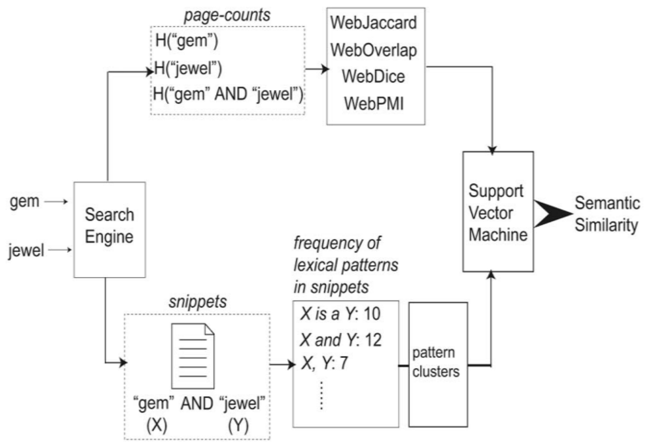


Figure 1: Shows an example of using the semantic similarity between two words, gem and jewel.

This shows associate example to work out the similarity engine and retrieve page counts for the two words is "gem," "jewel," and for his or her conjunctions is "gem AND jewel". Page counts-based similarity scores consider the global co-occurrences of two words on the web. The local context is not considered in which two words are co-occur. Snippets taken by a search engine represent the native context within which two words co-occur on the web. For the conjunctive query of the two words the snippets give the frequency of various lexical patterns. It is that by using more than one lexical pattern a linguistic relation may be expressed. Clustering different lexical pattern that shows the relation enables us to represent a linguistic relation between two words accurately. For this, we propose a sequential pattern clustering algorithm that each page counts-based co-occurrence measures and lexical pattern extraction are used to define various features that represent the relation between two words. In this paper we tend to propose a lexical frequency pattern clustering algorithm through which we are able to calculate the frequency of co-occurrence of a word in a data file.

3.1 Page Count-Based Co-occurrence Measures:

Page counts for the question P AND Q could also be thought of as associate approximation of co-occurrence of two words P and Q on the web. However, page counts for the query P AND Q alone do not accurately express linguistics similarity. As a result of the page count for "car" AND "automobile," is 11,300,000 and where as for "car" AND "apple is 49,000,000." Although, automobile is more similar to car than apple, page counts for the query "car" AND "apple" are more than four times greater than those for the query “car” AND “automobile.” One must consider the page counts not just for the query P AND Q, but in addition for the individual words P and Q to assess linguistics affinity between P and Q. we tend to calculate four modern co-occurrence measures; Jaccard, Overlap Simpson, Dice, and point wise mutual information (PMI), to calculate linguistics similarity using page counts. For the remainder of this paper, we use the notation H (P) to denote the page counts for the query P in a search engine. The WebJaccard coefficient between words P and Q, WebJaccard (P, Q), is defined as

$$WebJaccard(P,Q) = \begin{cases} 0, & if\, H(P \cap Q) \le c, \\ \dfrac{H(P \cap Q)}{H(P) + H(Q) - H(P \cap Q)}, & otherwise. \end{cases}$$

Therein, P∩Q denotes the conjunction query P AND Q. Given the size and noise in internet information, it is possible that two words may appear on some pages even though they are not connected. So as to scale back the adverse effects because of such co-occurrences, we tend to set the WebJaccard coefficient to zero if the page count for the query P∩Q is a less than a threshold c^2 . Similarly, we tend to outline Web Overlap, WebOverlap (P,Q), as

$$WebOverlap = \begin{cases} 0, & if\, H(P \cap Q) \le c, \\ \dfrac{H(P \cap Q)}{\min\{H(P), H(Q)\}}, & otherwise. \end{cases}$$

WebOverlap could be a natural modification to the Overlap (Simpson) coefficient. We tend to outline the WebDice coefficient as a variant of the Dice coefficient. WebDice(P,Q) is outlined as

$$WebDice(P,Q) = \begin{cases} 0, & if\, H(P \cap Q) \le c, \\ \dfrac{2H(P \cap Q)}{H(P)\_H(Q)}, & otherwise. \end{cases}$$

Point wise mutual information [10] is a measure that is motivated by information theory; it is meant to mirror the dependence between two probabilistic events. We tend to outline WebPMI as variant form of point wise mutual information using page counts as

$$WebPMI(P,Q) = \begin{cases} 0, & if\, H(P \cap Q) \le c, \\ \log_2\left(\dfrac{\frac{H(P \cap Q)}{N}}{\frac{H(P)}{N}\frac{H(Q)}{N}}\right), & otherwise. \end{cases}$$

Here, N is that the number of documents indexed by the search engine. To calculate PMI accurately, we tend to understand N, the amount of documents indexed by the programmer. Though estimating the amount of documents indexed by a search engine [11] is a remarkable task itself, it is beyond the scope of this work. In the present work, we tend to set N ¼ 1010 in step with the amount of indexed pages reported by Google. As previously mentioned, page counts are mere approximations to actual word co-occurrences within the web. However, it has been shown by trial and error that there exists a high correlation between word counts obtained from a web programmer which from a corpus [12]. Moreover, the approximated page counts have been successfully used to improve a variety of language modelling tasks.

3.2 Lexical Pattern Extraction:

Page counts-based co-occurrence measures do not take into account the local context during which those words co-occur. This could be problematic if one or each words area unit ambiguous, or once page counts area unit unreliable. On the opposite hand, the snippets came by a search engine for the conjunctive query of two words give helpful clues associated with the linguistics relations that exist between two words. A snipping contains a window of text hand-picked from a document that features the queried words. Snippets area unit helpful for search as a result of, most of the time, a user will browse the snipping and judge whether or not a selected search result is relevant, while not even gap the computer address. Using snippets, contexts are additionally computationally economical as a result of it obviates the necessity to transfer the supply documents from the online, which might be time overwhelming if a document is giant. Here, the phrase could indicate a linguistics relationship between cricket and sport. Several such phrases indicate linguistics relationships. As an example, additionally called, is a, part of, is associate degree example of all indicating linguistics relations of various varieties. Within the example given on top of, words indicating the linguistic relation between cricket and sport seem between the query words. Exchange the query words by variables X and Y, we will type the pattern X could be a Y from the instance given on top of. Despite the potency of using snippets, they cause two main challenges: 1st, a snipping may be a fragmented sentence; second, a search engine would possibly turn out a snipping by choosing multiple text fragments from completely different parts in a very document. As a result of most syntactical or dependency parsers assume complete sentences because the input, deep parsing of snippets produces incorrect results. Consequently, we tend to propose a shallow lexical pattern extraction rule using web snippets, to acknowledge the linguistics relations that exist between two words. Lexical syntactical patterns are employed in varied linguistic communication process tasks like extracting hypernyms [13], [14], or meronyms[15], query answering[16], and paraphrase extraction[17]. Though a search engine would possibly turn out a snipping by choosing multiple text fragments from completely different parts in a very document, a predefined delimiter is employed to separate the various fragments. As an example, in Google, the delimiter "..." is employed to separate completely different fragments in a very snipping. We tend to use such delimiters to separate a snipping before we tend to run the projected lexical pattern extraction rule on every fragment. Given two words P and Q, we tend to query an online programmer using the wildcard query "P***** Q" and transfer snippets. The "_" operator matches one word or none in a very webpage. Therefore, our wildcard query retrieves snippets during which P and Q seems at intervals a window of seven words. as a result of a search engine snipping contains ca. twenty words on the average, and includes two fragments of texts hand-picked from a document, we tend to assume that the seven word window is decent to hide most relations between two words in snippets. In fact, over 95 percent of the lexical patterns extracted by the projected technique contain but five words. We tend to plan to approximate the native context of two words using wildcard queries. As an example, Fig. four shows a snipping retrieved for the query "ostrich***** bird." For a snipping nine, retrieved for a word combine H (P, Q), first, we tend to replace the two words P and Q, severally, with two variables X and Y. We tend to replace all numeric values by D, a marker for digits. A subsequence should contain only prevalence of every X and Y. The most length of a subsequence is L words. A subsequence is allowed to skip one or additional words. However, we tend to don't skip over g variety of words consecutively. Moreover, the whole variety of words skipped in a very subsequence shouldn't exceed G. We tend to expand all negation contractions in a very context. As an example, didn't is dilated to failed to. We tend to don't skip the word not once generating sub sequences. As an example, this condition ensures that from the snipping X is not a Y, we tend to do not turn out the subsequence X could be a Y. Finally, we tend to count the frequency of all generated sub sequences and solely use sub sequences that occur over T times as lexical patterns.

3.3 Lexical Frequency Pattern Clustering:

The basic plan of Lexical Frequency Pattern agglomeration (LFPC) consists finds all the consecutive patterns in a very organization, which is made from the document information (DDB). The info structure stores all the various pairs of contiguous words that seem within the documents, while not losing their consecutive order. Given a threshold β nominative by the user, LFPC reviews if its combine is β-frequent. During this case, LFPC grows the sequence so as to see all the attainable peak consecutive patterns containing such combine as a prefix. An attainable peak consecutive pattern (PMSP) is a peak consecutive pattern (MSP) if it is not a subsequence of any previous MSP. This suggests that each one the keep MSP that area unit subsequence of the new PMSP should be deleted. The projected rule consists of three steps represented as follows:

In the initiative, for every completely different word (item) within the DDB, LFPC assigns associate degree number as symbol. Also, for every symbol, the frequency is keep, i.e., the amount of documents wherever it seems. These identifies area unit employed in the rule rather than the words. Table one shows associate degree example for a DDB containing four documents.

In the second step (Fig. 1), DIMASP-C builds an information structure from the DDB storing all the pairs of contiguous words $<w_i, wi_{+1}>$, which seem in a very document and a few extra data to preserve the consecutive order. the info structure is associate degree array that contains in every cell a combine of words $C=<w_i,w_{i+1}>$,the frequency of the combine (Cf), a mathematician mark and an inventory Δ of nodes δ wherever a node δ stores a document symbol (δ.Id), associate degree index (δ.Index) of the cell wherever the combine seems within the array, a link (δ.NextDoc) to take care of the list Δ and a link (δ.NextNode) to preserve the consecutive order of the pairs with relation to the document, wherever they seem. Therefore, the amount of various documents given within the list Δ is Cf. this step works as follows: for every combine of words, within the document DJ, if, doesn't seem within the array, it is accessorial, and its index is gotten. Within the position index of the array, add a node δ at the start of the list Δ. The accessorial node δ has J as δ.Id, index as δ.index, δ.NextDoc is connected to the primary node of the list Δ and δ.NextNode is connected to future node δ reminiscent of the document DJ. If the document symbol (δ.Id) is new within the list Δ, then the frequency of the cell (Cf) is exaggerated. In Fig. a pair of the data structure designed for the document information of table one is shown.

Table 1: Example of a document database and its identifier representation.

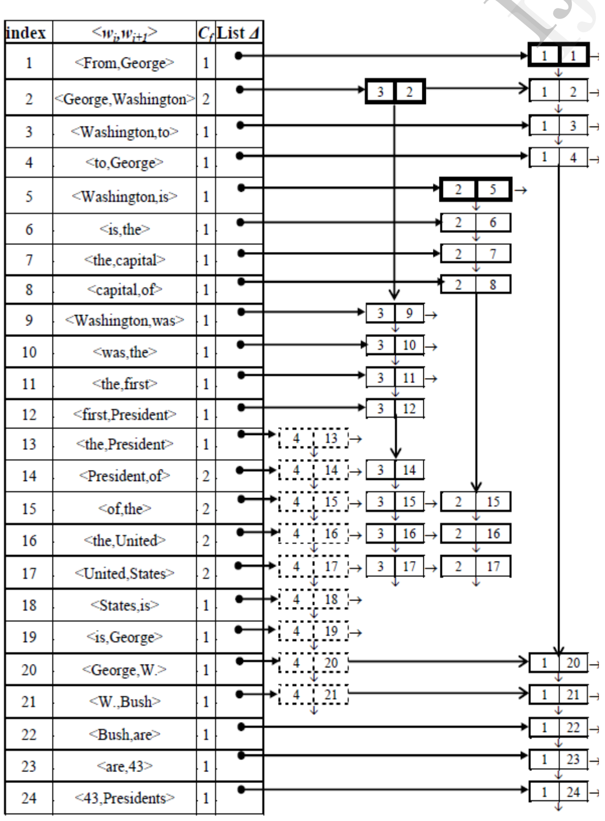
Dj	Document Database
1	From George Washington to George W. Bush are 43 presidents
2	Washington is the capital of the United States
3	George Washington is the first president of the United States
4	The president of the United States is George W. Bush
	Document database (words by integer identifiers)
1	<1,2,3,4,2,5,6,7,8,9>
2	<3,10,11,12,13,11,14,15>
3	<2,3,16,11,17,18,13,11,14,15>
4	<11,18,13,11,14,15,10,2,5,6>

```
Step 1: Algorithm to construct the data structure for the DBD
Input: A document data base DBD
Output: The Array
        For all the documents $Dj \in DBD$ do
Array ← Add a document (Dj) to the array
end-for
Step 1.1: Algorithm to add a document
Input: A document Dj
Output: The Array
For all pairs $<w_i, wi_{+1}>\in$ Dj do
δi ← create a new pair δ
δi.Id ← J //Assign the document identifier to the node δ
index ←Array[$<w_i, wi_{+1}>$] //Get the index of the cell
δi.index ← index  //Assign the index to the node δ
α ← get the first node of the list δ
If δi.Id ≠ α.Id then the document identifier is newer to the list δ
Increment Cf   //increment the frequency
δi.NextDoc ← α  //link the node α at the beginning of list δ
List δ←Add δi as the first node  //link it at the begining
δi-1.NextNode ←δi   //do not lose the sequential order
```

```
        end-if
    end-for
```

Given a threshold β, LFPC uses the constructed structure for mining all the maximal sequential patterns in the collection. LFPC verifies if this pair is β-frequent for each pair of words stored in the structure, in such case LFPC, based on the structure, grows the pattern while its frequency remains greater or equal than β. When a pattern cannot grow, it is a possible maximal sequential pattern (PMSP), and it is used to update the final maximal sequential pattern set. since LFPC starts finding 3-MSP or longer, then at the end, all the β-frequent pairs that were not used for any PMSP and all the β-frequent words that were not used for any β-frequent pair are added as maximal sequential patterns.

In order to be efficient, it is needed to reduce the number of comparisons when a PMSP is added to the MSP set. K-MSP is stored according to its length k in order to reduce the comparisons. There is a k-MSP set for each k. In this way, the k-PMSP must not be in the k-MSP set before adding a k-PMSP as a k- MSP. Therefore it must not be subsequence of any longer k-MSP. All its subsequences are eliminated when a PMSP is added.



index	$<w_i,w_{i+1}>$	C_f	List Δ
1	<From,George>	1	
2	<George,Washington>	2	
3	<Washington,to>	1	
4	<to,George>	1	
5	<Washington,is>	1	
6	<is,the>	1	
7	<the,capital>	1	
8	<capital,of>	1	
9	<Washington,was>	1	
10	<was,the>	1	
11	<the,first>	1	
12	<first,President>	1	
13	<the,President>	1	
14	<President,of>	2	
15	<of,the>	2	
16	<the,United>	2	
17	<United,States>	2	
18	<States,is>	1	
19	<is,George>	1	
20	<George,W.>	1	
21	<W.,Bush>	1	
22	<Bush,are>	1	
23	<are,43>	1	
24	<43,Presidents>	1	

Figure 2: Data structure built by LFPC for the database of the table 1.

For avoiding repeating all the work for discovering all the MSP when new documents are added to the database, LFPC only preprocesses the part corresponding to these new documents. First the identifiers of these new documents are defined in step 1, and then LFPC would only use the step 1.1 to add them to the data structure. Finally, the step 2.1 is applied on the new documents using the old MSP set, to discover the new MSP set, with dotted lines the new part of the data structure when D4 of table 1 is added as a new document. This strategy works only if the same â is used, however for a different â only the discovery step (step 2) must be applied, without rebuilding the data structure.

```
Step 2: Algorithm to find all MSP
Input: Structure from step 1 and β threshold
Output: MSP set
For all the documents Dj…(β-1) ϵ DDB do
    MSP set ← Find all MSP w.r.t. the document (Dj)
Step 2.1: Algorithn to find all MSP with respect to the document Dj
Input: A Dj from the data structure and a β threshold
Output: The MSP ser w.r.t. to Dj
For all the nodes δi-1….n ϵ Dj i.e. <w_i, wi_{+1}> ϵ Dj do
  If Array [δi.index].frequency ≥β then
     PMSP←Array[δi.index]. <w_i, wi_{+1}>   //the initial pair
     δ´← Copy the rest of the list of δ being from δi.NextDoc
     δ´f← Number of different documents in δ´
     δ´i←δi
     While δ´ f≥β do the growth tht PMSP
     δ´´← Array[δ´i+1.index].list
     δ´←δ´&δ´´   i.e. {αϵδ´|(α.index=δ´i+1)^(δ´i.NextNodee=α)}
     δ´f← Number of different documents in δ´
      If δ´f ≥ β then to grow the PMSP
         Array [δ´i+1.index].mark ← "used"
         PMSP ← PMSP + Array[δ´i+1.index]. < wi_{+1}>
            δ´i←δ´i+1 i.e. δ´i.NextNode
      end-while
```

```
        If |PMSP| ≥ 3 then add the PMSP to the MSP set
            MSP set ← add a k-PMSP to the MSP set  //step 2.1.1
End-for
For all the cells C ϵ Array do the addition of the 2-MSP
  If Cf ≥ β and C.mark =”not used” then add it as 2-MSP
        2-MSP set ← add C. <$w_i$, $wi_{+1}$>
```

Step 2.1.1: Algorithm to add a PMSP to the MSP set

```
Input: A k-PMSP, MSP set
Output: MSP set
If (k-PMSP ϵ k-MSP set) or (k-PMSP is subsequence of some longer k-MSP) then  //do not add anything
        return MSP set
else  //add as a MSP
        k-MSP set ← add k-PMSP
         {delete S ϵ MSP set | S ϵ k-PMSP}
         return MSP set
```

4.Results:

To understand how to find the linguistic similarity between words in a data file, we present a simple example here. As soon as we give the input word in the search engine, text snippets will be accepting. for example if we take the word “apple” then immediately snippets will be loading and it will open the snippet file if already exists, if that file doesn’t exists in the database we have to retrieve it from web. After inputting the word apple we have to input the word “computer” then immediately snippets will be loading about computer from the data base. Now we have to measure the page count measures to the words apple and computer.

Table 2: Page Count Measures:

Page Count Measures:

H(Apple):29.0

Enter:C

H (Apple and computer):32.0

Re - Compute

Extract Lexical Pattern

The above figure shows that page count co-occurrence between the words apple and computer. The page count co-occurrence of apple and computer are measured by WebJaccard, WebOverlap, WebDice and WebPMI

Table3: Page Count Measures:-

Page Count Measures:
H (Apple): 29.0 Enter: C
H(Computer): 31.0
H(Appleandcomputer):32.0

WebOverlap Measure: 1.103448275862069
WebPMI Measure: 0.035558001114496557
WebDice Measure: 1.0666666666666667
WebJaccardMeasure: 1.1428571428571428

Re - Compute

Extract Lexical Pattern

The above figure shows that, by entering some value we can extract frequency for the patterns of that value.

Table 4: Patterns:

Patterns:

PATTERN	FREQUENCY
Is maximum a	14
Design and creates ip…	2
-1. Is a personal	1

Next

Here it shows the patterns and its frequency by using the lexical frequency pattern clustering algorithm.

Table 5: Shows the Similarity Measure between apple and computer:

After Clustering……
Sim Measure between Apple and Computer is:
0.011246683037477784

OK

This shows the similarity measure between apple and computer after clustering.

5.Future Enhancement:

At present this project is designed to measure linguistic affinity between two words and to calculate the frequency between the two words in a data file. We can additionally enhance this work by finding frequency between more words in a data file and can also measure more number of similar words. We can further enhance the frequency lexical pattern clustering algorithm so as to boost the efficiency of the system.

6.Conclusion:

We tend to plan a linguistic affinity measure using page counts and snippets retrieved from a web for two words. Using page counts-based co-occurrence technique four word co-occurrence measures were computed. We proposed a lexical pattern extraction algorithm to extract various linguistic relations that exist between two words. Each page counts based co-occurrence measures and lexical pattern extraction was used to define features for a word pair. Lexical frequency pattern clustering is employed to find the frequency of the two similar words in a data file. By the proposed technique we can simply calculate the frequency of the two words.

7.Refrences:

[1]Kilgarriff, “Googleology Is Bad Science,” Computational Linguistics, vol. 33, pp. 147-151, 2007.
[2]M. Sahami and T. Heilman, “A Web-Based Kernel Function for Measuring the Similarity of Short Text Snippets,” Proceedings of the 15th International World Wide Web Conference, 2006.
[3]D. Bollegala, Y. Matsuo, and M. Ishizuka, “Disambiguating Personal Names on the Web Using Automatically Extracted Key Phrases,” Proceedings of the 17th European Conference on Artificial Intelligence, pp. 553- 557, 2006.
[4]H. Chen, M. Lin, and Y. Wei, “Novel Association Measures Using Web Search with Double Checking,” Proceedings of the 21st International Conference on Computational Linguistics and 44th Ann. Meeting of the Assoc. for Computational Linguistics (COLING/ACL ’06), pp. 1009-1016, 2006.
[5]P. Resnik, “Using Information Content to Evaluate Semantic Similarity in a Taxonomy,” Proceedings of the 14th International Joint Conference on Aritificial Intelligence, 1995.
[6]D. Mclean, Y. Li, and Z.A. Bandar, “An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources,” IEEE Trans. Knowledge and Data Eng., vol. 15, no. 4, pp. 871-882, July/Aug. 2003.
[7]G. Miller and W. Charles, “Contextual Correlates of Semantic Similarity,” Language and Cognitive Processes, vol. 6, no. 1, pp. 1-28, 1998.
[8]K. Church and P. Hanks, “Word Association Norms, Mutual Information and Lexicography,” Computational Linguistics, vol. 16, pp. 22-29, 1991.
[9]D. Lin, “An Information-Theoretic Definition of Similarity,” Proceedings of the 15th International Conference on Machine Learning (ICML), pp. 296-304, 1998.
[10]R. Cilibrasi and P. Vitanyi, “The Google Similarity Distance,” IEEE Trans. Knowledge and Data Eng., vol. 19, no. 3, pp. 370-383, Mar. 2007.
[11]Z. Bar-Yossef and M. Gurevich, “Random Sampling from a Search Engine’s Index,” Proceedings of the 15th International World Wide Web Conference, 2006.
[12]F. Keller and M. Lapata, “Using the Web to Obtain Frequencies for Unseen Bigrams,” Computational Linguistics, vol. 29, no. 3, pp. 459-484, 2003.
[13]M. Hearst, “Automatic Acquisition of Hyponyms from Large Text Corpora,” Proceedings of the 14th Conference on Computational Linguistics (COLING), pp. 539-545, 1992.
[14]R. Snow, D. Jurafsky, and A. Ng, “Learning Syntactic Patterns for Automatic Hypernym Discovery,” Proceedings of the Advances in Neural Information Processing Systems (NIPS), pp. 1297-1304, 2005.
[15]M. Berland and E. Charniak, “Finding Parts in Very Large Corpora,” Proc. Ann. Meeting of the Assoc. for Computational Linguistics on Computational Linguistics (ACL ’99), pp. 57-64, 1999.
[16]D. Ravichandran and E. Hovy, “Learning Surface Text Patterns for a Question Answering System,” Proc. Ann. Meeting on Assoc. for Computational Linguistics (ACL ’02), pp. 41-47, 2001.
[17]R. Bhagat and D. Ravichandran, “Large Scale Acquisition of Paraphrases for Learning Surface Patterns,” Proc. Assoc. for Computational Linguistics: Human Language Technologies (ACL ’08: HLT), pp. 674-682, 2008.