A STUDY ON LEXICAL CHAIN IDENTIFICATION AND WORD SENSE DISAMBIGUATION

Ştefan Daniel DUMITRESCU[1], Ana GĂINARU[2], Ştefan TRĂUŞAN-MATU[3]

Lucrarea de faţă investighează problema lanţurilor lexicale şi a dezambiguizării de sens a cuvintelor precum şi conexiunea între aceste două subiecte. Este propus un sistem care extrage cuvinte dintr-un text nestructurat şi generează un set de lanţuri lexicale precum şi dezambiguizarea cuvintelor bazată pe sinseturi WordNet. Sunt testaţi trei algoritmi nesupervizaţi, fiecare dintre aceştia cu câte trei măsuri de similaritate bazate pe conceptul de Conţinut Informaţional. Pentru evaluarea sistemului comparăm rezultatele obţinute cu fişiere adnotate manual ce conţin cuvinte dezambiguizate.

The present paper investigates the issues of lexical chains and word sense disambiguation and the strong connection between them. We propose a system that extracts words from unstructured text and provides sets of lexical chains and also words and their disambiguation based on WordNet's synsets. We test three unsupervised algorithms, each with three similarity measures based on the concept of Information Content. To evaluate the system we compare the results against manually annotated files containing disambiguated words.

Keywords: word sense disambiguation, lexical chains, semantic distance, clustering algorithms

1. Introduction

The article investigates two major tasks in the Natural Language Processing domain, with strong implications in many other fields like Information Retrieval, Information Extraction and up to Artificial Intelligence. We investigate the methods and results of extracting lexical chains and of disambiguating words (WSD).

We aim to build a system that takes natural text as input and outputs sets of lexical chains and disambiguated words. As lexical chains are directly connected to the problem of sense disambiguation, we will focus our attention at both of these basic but very difficult issues.

[1] PhD student, Computer Science Department, University POLITEHNICA of Bucharest, Romania, e-mail: dumitrescu.stefan@gmail.com
[2] Eng., Computer Science Department, University POLITEHNICA of Bucharest, Romania
[3] Prof., Computer Science Department, University POLITEHNICA of Bucharest, Romania

Lexical chains are the result of connecting semantically related words in portions of text. The obtained set of chains represents cohesion threads throughout the text. This set of chains is used in natural language processing sub-tasks like automatic summarizing, information retrieval and information extraction, spell checking, topic segmentation and others.

Word sense disambiguation is the task of assigning senses to words. A word is polysemous, meaning that depending on the context, it can have multiple meanings. Disambiguating its correct sense is the first building block in many NLP applications, from the same categories as lexical chains.

There is a direct, two-way connection between lexical chain identification and word sense disambiguation. On one hand, lexical chaining could benefit from WSD because knowing for certain (or at least with a good degree of confidence) what sense a word has in a sentence, adding that word to a lexical chain is a much simpler problem, as all the words in a chain gravitate along one idea (concept). On the other hand, having correct lexical chains greatly simplifies the task of WSD, because disambiguating a set of words is much easier if it is known that they point to the same meaning. The two problems are interconnected [5], and each can be to the other, if not a solution, at least a great method to improve accuracy.

We propose a system that tries to extract both lexical chains and to disambiguate words in one step, using a set of unsupervised clustering algorithms. In the clustering process itself, the choice of assigning a word to a cluster is based on calculating a similarity measure which implies choosing a sense for that word. In this way, we extract lexical chains as clusters as well as disambiguating the senses of words in respect to WordNet synsets, as further described.

2. Related Work

The proposed system takes on the problems of word sense disambiguation and lexical chain identification by method of clustering. This chapter looks at each of these subjects and briefly discusses the supporting resources, currently existing methods and approaches.

2.1. Word Sense Disambiguation

Entity disambiguation is the task of identifying which sense (meaning) of an entity (a simple or composed word) is used in a sentence, given the fact that words are affected by polysemy / homonymy problems.

Currently, there are three major directions for word sense disambiguation - WSD:

- *Knowledge-Based Disambiguation*, where lexical resources like thesauri and dictionaries are used;

- *Supervised Disambiguation*, where machine learning approaches are used to train various classifiers; these systems encode custom features into feature-vectors, and, based on the provided labeled training data build models used to assign appropriate word senses;

- *Unsupervised Disambiguation*, where the learning system uses unlabeled corpora.

In the present paper we investigate a Knowledge-Based approach to WSD. We want to semantically differentiate between each word's senses, and thus we need to use an existing lexical resource that we can refer to when assigning a certain sense to a word. The senses (meanings) usually come from thesaurus or dictionaries, where, depending on the resource used, polysemy, hyper/hyponymy relations, homonymy, etc. relations are represented. For the purposes of this paper, we will use the WordNet lexical resource for the WSD task.

2.2. WordNet Lexical Resource

Princeton University's WordNet[4] is a free electronic lexical resource containing dictionaries of nouns, verbs, adjectives and adverbs. It provides not only dictionaries but also organizes related concept from the individual categories into synsets (synonym sets). Currently the latest version of WordNet is 3.0, containing around 150000 words organized in around 115000 synsets.

The basic WordNet concepts are: synsets, glosses and lemmas. The gloss is an explanation or definition of a word in a text, basically a sense-disambiguated corpus. In addition to the definition itself, the gloss also contains additional explanations and examples. Lemmas are the words that belong to a synset. They represent the string text of the word from WordNet database.

The synset is the equivalent of a concept. A synset is, in essence, an ordered list of synonyms. The synonyms themselves are words that are in the same lexical category and are commonly used to express the same meaning. Synsets as well and the synset hierarchy (created by relations like is-A, part-of, etc.) represent the most used information in WordNet, bringing also semantic value over the standard lexical value a dictionary provides.

WordNet is currently the most commonly used lexical resource for word sense disambiguation. It encodes many senses for every word, and while this seems at first a solid base to use for the diverse tasks, it has been argued that there may be too many senses even for humans [7]. This issue might prevent WSD systems from performing at their best. Solutions have been proposed, like clustering methods that might be used to group similar senses together and reduce the total number to only a few, more manageable and distinct senses [8]. For English, accuracy is over 90% if we take coarse-grained senses (every word has

[4] http://wordnet.princeton.edu/

few, clearly defined and separate senses), and about 59.1% - 69.0% for fine-grained senses (reported by Senseval-2[5]) (every word has a many senses covering many possible meanings). We must note that for fine-grained senses, the baseline algorithm is that of always choosing the first sense of every word, with accuracy ranging from 51.5% to 57%. This fine-grained baseline accuracy is a problem for most algorithms to even reach, let alone out-perform.

2.3. Semantic similarity measures

For semantic similarity we will use Information Content based measures. Information Content (IC) is a specificity measure for concepts [2]. For example, concepts that are more specific have a higher IC associated value than more general concepts (ex: locomotive has a higher IC than device). The IC value is calculated depending on the frequency of concepts from the text. The process is as follows: the text (corpus) from which IC values are to be derived from is parsed, and for every concept found, its frequency as well as the frequency count of its ancestors is increased by one in WordNet. The ancestor hierarchy is a concept hierarchy where links are WordNet relations (e.g.: for nouns we have *is-A* or *part-of* relations).

After the frequency count is completed, for each concept in WordNet the IC value is computed as the negative log of the probability (frequency count) of the concept.

$$IC(s) = -\log(P(s)) \tag{1}$$

For the purposes of this paper we have used IC information that was extracted from the Brown and Semcor corpuses.

We have implemented three different Information Content measures [1]: Resnik's measure *res*, Lin's measure *lin*, Jiang and Conrath's measure *jcn*. All these measures take two synsets as inputs, and produce a floating point value that represents the similarity between the two synsets. They are all based on the idea of finding the least common ancestor (LCA), meaning finding the concept that subsumes both of the synsets in WordNet's synset Is-A (hypernym) hierarchy. If there is more than one LCA, the least general LCA is taken (the lowest in the hierarchy).

The Resnik measure (*res*) provides the basic metric that is used both for *lin* and *jcn* measures. The similarity value is the IC value of the synset's LCA.

$$sim_{res}(s1, s2) = IC(lcs(s1, s2)) \tag{2}$$

[5] http://www.senseval.org/

The *res* measure may provide the same value for different synsets that share the same LCA, and thus is not a very informative measure. Lin's measure attempts to improve the accuracy by incorporating information about the IC of each of the synsets.

$$sim_{lin}(s1, s2) = \frac{2 \times sim_{res}(s1, s2)}{IC(s1) + IC(s2)} \tag{3}$$

Jiang and Conrath provide an alternate distance metric instead, using the same elements as Lin:

$$dist_{jcn}(s1, s2) = IC(s1) + IC(s2) - 2 \times sim_{res}(s1, s2) \tag{4}$$

However, to transform *jcn* into a similarity measure, we can simply invert the distance formula:

$$sim_{jcn}(s1, s2) = \frac{1}{dist_{jcn}(s1, s2)} \tag{5}$$

While this formula provides a similarity measure instead of a distance measure between synsets, it does alter the value differences between sets of synsets due to the division.

These three measures types represent standard measures used for a long time in NLP applications and other fields like WSD [6], with consistent results. We have implemented Resnik's measure because it provides a baseline for the other measures.

2.4. Clustering methods

Cluster methods can be categorized into two major categories: supervised learning techniques and unsupervised data mining methods. All supervised algorithms need a training phase that is quite expensive since it requires all word to be annotated manually. Unsupervised techniques are easier to use since they require only a few input parameters and then everything is computed by the algorithm automatically.

We chose only unsupervised algorithms for our study. The most used methods in data mining are *apriori* which extract most frequent sets of words from unstructured text, Latent Semantic Indexing that establishes associations between words from similar patterns, K-means which cluster data points by using k-centroids and reorganizing words around it until all clusters are stable, and hierarchical clustering that iteratively divides clusters until they are good enough.

Our system aims to group words that have similar meaning into one group, so that it can construct a context that describes the sequence of points. In order to have an accurate context, the system needs to be able to disambiguate word senses. The models that can be used for extracting lexical chains from unstructured text are K-means and hierarchical clustering. A first computational model of lexical chains was introduced by [9]. This algorithm suffers from inaccurate WSD, since their greedy strategy immediately disambiguates a word when it is first encountered. More recently, [10] proposed a system for lexical chain extraction which models Barzilay's approach that has good results, but has inaccuracies in WSD. An even more recent approach is [4] based on the fact that separating WSD from the actual chaining of words can increase the quality of chains. Their results are better than the ones before but still under 65% accuracy.

3. System Implementation

3.1. System architecture

The system is built in Java, using a various set of wrappers and tools like WordNet 3.0, JWNL - a java WordNet wrapper, Stanford's POS tagger library. The core functions, including the similarity metrics as well as the clustering algorithms, have been manually implemented in Java. The system takes as input text files containing free unstructured text to be analyzed.

Before starting file processing, initialization is performed. Synsets are extracted into memory from WordNet, and then the Information Content data is loaded and attached to the synsets.

The first processing step is reading the text file, part-of-speech tagging it, and extracting the nouns. After the file is processed, the list of nouns is passed to the clustering engine. The clustering engine uses 3 algorithms in turn to process the nouns and extract clusters of words with chosen synsets. All running clustering algorithms use each of the three similarity metrics (*res*, *lin* and *jcn*) and each of the methods of choosing words' synsets (baseline first sense, random and maximum similarity). The similarity metric functions are provided by the Semantic Similarity module that in turn is directly linked to WordNet data files.

When searching for the right cluster for one data point, in the first phase, we investigate different senses to try to fit the word into an existing lexical chain or group. However, after the word belongs to a lexical chain or a group, the algorithm identifies the most probable sense using the context created by the rest of the words in the cluster. We use this last method to give the output for word sense disambiguation. In this last step the algorithm uses the context created for each group in order to extract a series of summary words or tags from each lexical chain.

The resulting clusters then go into the evaluation module. The evaluation module first evaluates the chosen senses for each word versus a manually annotated file with correct sense ids.

3.2. The similarity metric module

The module contains the algorithms that provide the semantic similarity metrics. Before being able to use such a metric, at system start, WordNet's synsets are all extracted as an in-memory array list. JWNL[6], a java wrapper for WordNet provides the necessary functions to parse synset by synset. Each synset is then added new properties, the most important one being the Information Content value. For each synset the IC is filled by using WordNet::Similarity's[7] IC files.

The IC files contain all of WordNet's synsets and their associated IC values. Each line contains a synset ID and an IC value. The values are computed by analyzing different corpuses. We have used the Brown and the Semcor corpus annotated files for IC values.

The module exports a function taking as input two words as strings, the type of similarity measure (either *res*, *lin*, of *jcn*) and a synset choice type. A search in WordNet gives us for every word a set of possible senses (synsets). The synset choice type is needed to select, given the two input words, which of the word's synsets to be used to calculate the similarity with.

We have implemented 3 types of choices. The first method is of always choosing the *first sense* for each word. This is also the baseline for the system. The second method is that of *randomly choosing a sense* (a synset). The third method is that of choosing the synsets that *maximize similarity*. This means that for any two words, their synsets are extracted and then each possible combination of pairs of synsets are measures and the pair that scores the maximum similarity is returned.

3.3. The clustering algorithms module

We implemented three clustering algorithms, all which use a semantic distance metric between words for building the word groups. We use these algorithms with each distance metric in order to compute clusters of words that are related to each other. We use the groups for name entities disambiguation. For each word in the group we compute the distance to each other word in the same cluster and we select the sense (synset) of the word that has the highest similarity. We identify the sense that occurs more frequent and assign it to the word in question.

[6] http://sourceforge.net/projects/jwordnet/
[7] http://wn-similarity.sourceforge.net/

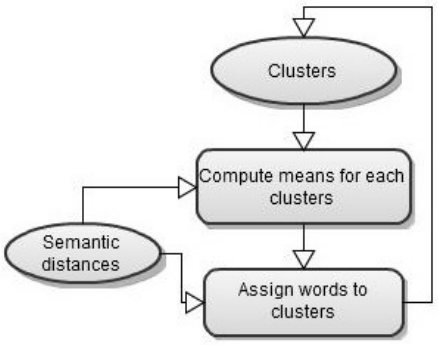

Fig. 1. The k-means clustering

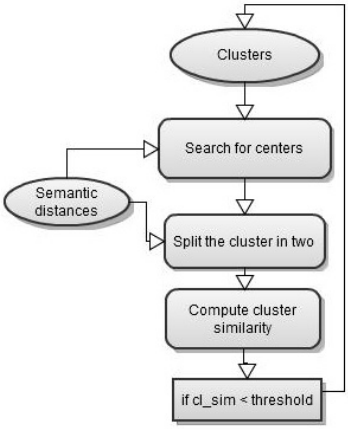

Fig. 2. The hierarchical clustering method

In the following subsections we will describe the algorithms in more detail.

K-means. The *k*-means clustering is a method of grouping similar words into a specific number of clusters in which each observation word belongs to the cluster with the nearest mean. The methodology is presented in Fig. 1.

Basically the algorithm uses two steps: in the first phase the algorithm computes the means for all created clusters and in the second one redistributes data points from one cluster to another according to the semantic distance between all means and the point. The mean of a cluster is represented by one of the words in the group that has the minimal distance towards all the other data points in the cluster.

The process is recursive and keeps reorganizing the words form one cluster to another until there are no more changes between two sequential loops.

The only input parameter for this method is the number of clusters used for the initial division of words. In our experiments we tested the method with different values for this parameter.

k-Nearest Neighbors. The k-Nearest Neighbors is a clustering technique that classifies words based on closest point classes. Initially this technique was a supervised data mining method where the closest point classes were computed from a training set. In our implementation we used an unsupervised method that has its basis in the k-nearest neighbor algorithm.

Initially all points are distributed randomly into a specified number of classes. The algorithm then iterates through all points in the dataset and inspects the k closest neighbors to that point. The algorithm searches for the most frequent class from the extracted set and assigns it to the data point in study.

The method has a recursive form; it iterates the dataset and reassigns classes until there are no more modifications. The parameters that are needed for this method are the number of initial classes and the number of neighbors to be

inspected. The k-nearest neighbor just assigns for each point the class of the word that has the most semantic similarity.

Hierarchical clustering. All hierarchical clustering methods divide or merge groups due to different criteria until a certain conditions are met. In our implementation we use a top down hierarchical method by starting with the whole dataset as a big cluster and then divide it constantly until all clusters have the similarity between their components over a specific threshold. The methodology for our top down implementation is presented in Fig. 2.

In each division step, the algorithm searches for two centers for each of the remaining clusters. The centers must be as further away from one another but still close to the other existing words in the group. The two centers are then used to split the cluster in two depending on the semantic distances between them and the two centers. For each new created cluster we compute the similarity between the included words as the mean distance between each pair of points. If the value is over a specified threshold then the cluster is good enough and the algorithm will not inspect it again. The stop criterion in our case is when all clusters have the similarity over the threshold.

The algorithm receives one input parameter, the threshold for the cluster similarity. We could not chose a default value for the threshold since different distance metrics have different ways and values when measuring similarities.

3.4. Lexical chains

In our previous work [3] we implemented an algorithm for lexical chain identification that was specifically build for chat conversations. We adapted the algorithm for structured text and compare the results with the ones obtained using different clustering methods. In [3] we defined two metrics that measure semantic closeness between words, one for strong connections, synonyms or much related concepts, and one for medium connections. Medium connections are represented by word that share related meanings in only some contexts. The difference between a strong and medium connection between words is given by different values obtain by the semantic distance. We use the same metrics in our adaptation of the model.

The algorithm used has 3 basic steps. In the first phase we investigate each word from the text and look for strong connections between it and all existing chains. The first chain found will be linked to the new word since a strong connection has the highest priority. We limit the search to 20 sentences in the past since it is more computational efficient and also because as we search words that are further away, the chances of finding a relevant lexical chain decreases.

The second step takes place only if no strong connection was found. In this phase medium connections are investigated. The method searches each chain and selects the one with the highest semantic closeness. For the same reasons as in the previous step, the search is limited to 10 sentences in the past.

The last step deals with words that have no strong or medium connection with any of the existing chains. If no relations are found then a new chain is created containing only the inspected word.

4. Results

In our experiments we compare the results for different types of clustering algorithms using different types of metrics for the measurement of similarity closeness with the ones manually annotated. We used two input text files with different specifics; one is a large report from computer science that tests the way algorithms deal with technical terms having a formal language and the second file is a news report that tests the accuracy of the implemented methods in an everyday type of text.

We conducted two types of experiments. In the first experiment we tested the accuracy of all methods in identifying the correct synset for each word in the input text. For each input dataset we manually identified the nouns and then searched in the WordNet taxonomy for the correct sense. Each of the 4 methods presented in the previous paragraph are able to output the synset id and number for all the grouped words. Finally, we compute the percentage of correctly identified senses.

In the second experiment we tested the behavior of all methods in identifying the lexical chains from input files. For all clustering algorithms, the lexical chains are represented by the group of words that they output. We manually extracted the clusters from the input text files as an array of the same size as the number of nouns, where each value represents a group id that corresponds to a word from the input dataset. All algorithms output the same array type and finally compute the percentage of correctly classified words.

The two experiments are used to inspect how well each algorithm is able to identify lexical chains from different types of input text files and also to show its accuracy of finding the correct sense for each noun.

4.1. Experiment 1

In the first experiment we test the accuracy of all algorithms in identifying the correct synset for nouns in the input text. All results were compared against the manually assigned senses for the nouns. The accuracy is computed as the total number of correctly classified nouns divided by their total number.

In table 1 we show the results obtained using the k-means algorithm with an input parameter value of 3, 4 and 5. The best results obtained for all similarity measure choice and synset choice are for an initial number of classes of 4. The same pattern is also observed for the second input text file.

Table 1

k-Means results with different input parameter value

k-means	*FIRSTSENSE*	*MAXSIM*	*RANDOM*
3 classes			
RES	55	64	57
JCN	58	68	61
LIN	55	65	59
4 classes			
RES	55	66	59
JCN	**61**	**70**	**65**
LIN	58	68	61
5 classes			
RES	51	58	57
JCN	55	63	58
LIN	51	63	55

Fig. 3 shows the charts for using k-means with the input parameter value of 4 for all measuring units. *Res* was used as a base line and obtained the worst results for almost all synset choices. For Fig. 5 the *lin* similarity measurement and with RANDOM synset choice had worst results than *res*. Since the RANDOM synset choice chooses the sense of a word without a prefixed algorithm is expected its results to fluctuate. However the accuracy obtained by *jcn* has better results than all other methods for all metrics considered.

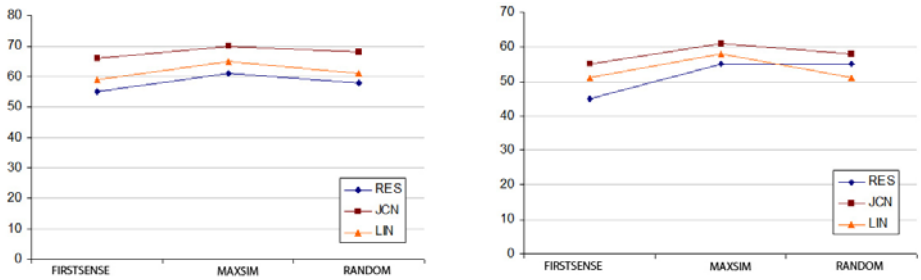

Fig. 3. Accuracy for k-means algorithm for the first and second input files

The results show that MAXSIM has better results than FIRSTSENSE and RANDOM in almost all the cases. For the second input file, the *res* similarity has the same value for RANDOM and MAXSIM. Since the RANDOM synset choice function could return any sense for a word and since this is an isolated case, this information is not very relevant for our analysis.FIRSTSENSE always returns the first meaning for all nouns, and still using this method we obtain good results for

all metrics in both input files. If we use *jcn* for the first input file the algorithm gave us an accuracy of 61%. This shows that more than 50% of the words used in a regular text keep their implicit first meaning.

The second text file obtained worst results than the first one for all cases under study. In the second text there are a lot of technical terms and not that many discussion topics. One explanation for the worse results could be that each cluster contains more words so the context for computing the senses is wider. Also the discussion topics used in this second input file are further apart from each other than in the first text (there is an analogy between hackers effects on the computing systems and car racks). The algorithm had some problems relating the word "person" to "hacker" or "driver" for example.

The next figure shows the results for k-nearest neighbor algorithm. We computed the output results for different values for the number of neighbors to inspect and we presented only the best result in the Fig. 4. The first chart is for the first input file and is computed using 2 neighbors and the second one is for the technical computer science text using 3 neighbors.

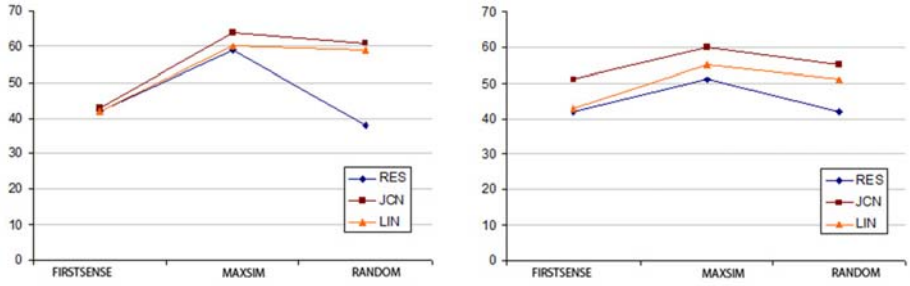

Fig. 4. Accuracy for kNN algorithm for the first and the second input file

We obtained better results for the second input file using a higher number of neighbors because of the specific characteristics of the second text, since it needs fewer and larger clusters.

This algorithm, as the previous one, has the best results for *jcn* and for MAXSIM. There is a greater distance between the results obtained using FIRSTSENSE and MASIM than in the previous algorithm. The explanation for this is that this algorithm uses just a small number of neighbors for deciding how to classify a word so it does not have the whole context. If the algorithm searches for the sense that maximizes the similarity between words (like MAXSIM does) that the context is no longer necessary. However due to this problem, in the case of FIRSTSENSE the method has problems in clustering words. Fig. 5 shows the results obtained by the hierarchical clustering method.

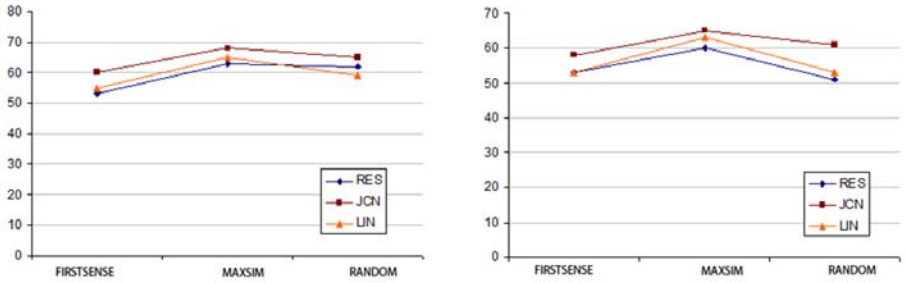

Fig. 5. Accuracy of hierarchical clustering for the first and second input file

The results are much closer to each other than with the rest of algorithms. However the overall curves of the graphs are the same. *Jcn* with MAXSIM is still the leader and in general *lin* is better than *res*. The distance between the values returned by MAXSIM and FIRSTSENSE are once again close to each other since this method, as the first one, compares each pair of words is the context is complete when splitting a cluster.

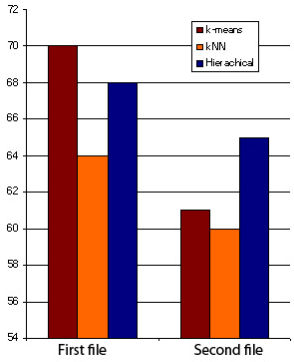

Fig. 6. Comparison between results obtained with different clustering methods

Fig. 6 compares the best results for each of the three methods: k-means, kNN and hierarchical. The k-nearest neighbor algorithm has the worst results. However between k-means and the hierarchical clustering method it is not clear. While for the first input text k-means has a better accuracy, for the second input file the hierarchical clustering algorithm is much more precise.

Since all results are obtained using *jcn* and MAXSIM, the difference is given by the methodology used by each algorithm. K-means depends on the initial number of classes. The hierarchical method divides clusters until all of them have messages with the similarity over a threshold. The threshold depends only on the measuring unit used and not on the input text.

Another difference is given by the way the hierarchical method chooses the two centers in the dividing process. If the text has many words that could belong in one group if we inspect one meaning and in another if we look at another sense, then the hierarchical method could choose the centers wrong. Once the splitting is finished there is no way back so the error will propagate until the final clusters. K-means reorganizes all clusters in each iteration so it won't have that problem. On the other hand if the text has no difficulties of that kind the hierarchical method should behave better than k-means.

4.2. Experiment 2

In the second experiment scenario we tested the behavior of all methods in identifying the lexical chains from input files. For better comparison, we manually extracted the clusters from the input text files. The output is represented by an array list where each value represents a group id that corresponds to a word from the input text file.

We implemented all algorithms to output the same type of array and then we computed the percentage of correctly classified words. Fig. 7 gives the results obtained for each method using *jcn* and MAXSIM. We use only *jcn* and MAXSIM since those are the metrics that give the best results for all clustering methods.

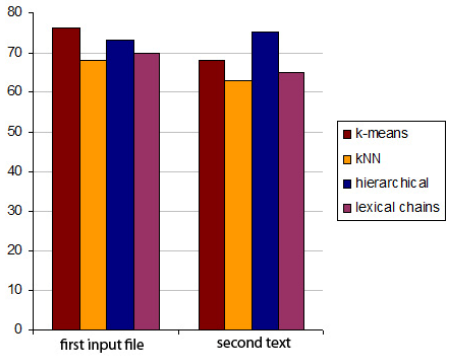

Fig. 7. Percentage of correctly classified words for different clustering methods

As we expected, since lexical chains and word sense disambiguation are closely related, the results follow the same curves. This was to be expected since if a word is in a wrong group it is likely that its meaning will be misclassified; similarly, this is valid the other way around. However all methods obtained a better value for the accuracy when inspecting the classification process and not WSD.

A misclassified word in a cluster influences the context for all other data points from the group. The higher the number of words that were not supposed to be in one cluster, the more it will negatively influence the context of other words. The conclusion is that the chance of finding the correct sense for all other words that are correctly put in a group will decrease with the increase of the misclassified data points from the same cluster.

We computed the percentage of correctly classified word using the lexical chain algorithm that we imported from the chat analyzer. The results are not better than the k-means or the hierarchical method, but very close. Basically this happens because the method was implemented for chat structure and not text. The other two methods consider the whole context created for a word and not just inspecting the previous sentences.

5. Conclusions

In this paper we presented a system that identifies groups of related words or lexical chains from unstructured text. The lexical chains are further used to assign the most appropriate sense for all nouns and to extract a series of summary words that are used to represent the group. We implemented three unsupervised clustering algorithms that have distinct methodologies and that use different semantic metrics. All methods use two basic steps. The first phase extracts the groups of words that are related to each other and the second one creates the context for each cluster by disambiguating the nouns senses. The context of each group of related words allows the system to extract tag words form the lexical chains.

In the experiment phase we compared the results obtained with all methods and for all semantic closeness measures against manually annotated groups. We used two distinct types of files, one scientific and one common-style text. We have two types of experiments: for the first one we test the disambiguation process by computing the percentage of correctly identified senses for all nouns; for the second type we test the lexical chain identification techniques. The purpose of this article is to show the comparative performance of different unsupervised methods in combination with different similarity measures to perform WSD and construct lexical chains. The obtained results show that out of all semantic similarity measures *jcn* performs consistently better, and between the clustering algorithms implemented and tested, k-means and hierarchical clustering obtain the highest results. Because both clustering algorithms have relatively close results, the choice whether to use one or the other depends mostly on the type of data to be clustered. Hierarchical clustering performs better on data that can be split in a tree-like structure, with the property that when one data point is added to a set it can no longer move to another set. K-means assumes no data

structure (which might often be the case), it is simply creating k clusters, without guarantee that another run will provide the same results (with random initialization). As there is no substantially "better" algorithm (although kNN performs somewhat worse), we have presented both k-means and hierarchical clustering as viable choices to perform word sense grouping.

REFERENCES

[1] *A. Budanitsky*, Semantic distance in Wordnet: An experimental, application-oriented evaluation of five measures, 2001, Available: http://citeseerx.ist.psu.edu/viewdoc /summary? doi=10.1.1.29.2985

[2] *T. Pedersen*, Information Content Measures of Semantic Similarity Perform Better Without Sense-Tagged Text, in The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), June 1-6, Los Angeles, 2010

[3] *A. Gainaru, S. Dumitrescu, S.M. Trausan,* Toolkit for automatic analysis of chat conversations, in The 8th International Conference on COMMUNICATIONS, Bucharest, 2010

[4] *M. Galley, K. McKeown,* Improving Word Sense Disambiguation in Lexical Chaining, in The 18th international joint conference on Artificial intelligence, Acapulco, 2003, pp. 1486-1488

[5] *N. Rani, M. Stuart,* Lexical Chaining and Word-Sense-Disambiguation, Technical report, School of Engineering and Applied Sciences, TR-06-07, 2007

[6] *S. Patwardhan, S. Banerjee,T. Pedersen*, Using Measures of Semantic Relatedness for Word Sense Disambiguation, in The 4th International Conference on Intelligent Text Processing and Computational Linguistics, Mexico City, 2003, pp. 241-257

[7] *R. Navigli,* Meaningful Clustering of Senses Helps Boost Word Sense Disambiguation Performance, in The 44th Annual Meeting of the Association for Computational Linguistics joint with the 21st International Conference on Computational Linguistics, Sydney, 2006, pp. 105-112

[8] *R. Snow, S. Prakash, D. Jurafsky*, Learning to Merge Word Senses, in Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp. 1005-1014, Prague, 2007

[9] *G. Hirst, D. St-Onge*, Lexical chains as representations of context for the detection and correction of malapropisms, in WordNet: An electronic lexical database. In: MIT Press, 1998

[10] *G. Silber, K. McCoy*, Efficiently computed lexical chains as an intermediate representation for automatic text summarization, in Computational Linguistics, 2003, pp. 29(1).