Automatic word clustering for Swedish

Paul Stapleton and **Antonios Antoniadis**
Computer Science Department
Lund University
{paul.stapleton.769, antonios.antoniadis.227}@student.lu.se

Abstract

Applications dealing with textual information often require knowledge of the word semantic categories. Constructing clusters manually is a task whose difficulty ranges from very hard to practically impossible, given the number of possible meanings for all possible words. In this paper we present an implementation for the Swedish language of the $UNICON$ (UNsupervised Induction of CONcepts) algorithm, found in Lin and Pantel (2001), along with some improvements/optimisations. Its main advantages are that it is an unsupervised algorithm and is not specific to any language, and so the implementation can be run on any large corpus.

1 Introduction

Word clustering is the act of placing words into semantically similar groups, so that words are clustered according to their meaning or context. Such a technique is useful when performing word disambiguation as we can understand the sense of a word from the semantic group it belongs to. Clustering can also be used for automated thesaurus generation, that is linking together words with a similar meaning. It takes large resources to construct such clusters manually, mainly due to the size of the corpus required to get a good result. Thus, an algorithm to construct the clusters automatically is required. Additionally due to the effort required to semantically annotate a corpus, an unsupervised algorithm is particularly desirable. In this paper, we present an implementation of the $UNICON$ algorithm, originally found in Lin and Pantel (2001), along with some improvements/optimisations and this is applied to a Swedish corpus to form clusters of Swedish words. This algorithm takes words from the corpus and constructs clusters with specific features, putting each word into the most suitable clusters. The aim is for each cluster to represent a semantic class and for it to be possible to assign words not found in the corpus to clusters based on the context it is found in.

Section 2 discusses previous work in this area. In Section 3, we present a broad description of the algorithm and how to construct the input to $UNICON$. Section 4 is a presentation of the algorithms used along with our improvements/optimisations. Section 5 discusses our implementation and its details and in Section 6 we analyse the resulting clusters when our implementation was run on a Swedish corpus and how we evaluated the quality of the clusters. Finally Section 7 is a short discussion about future work.

2 Related Work

There is a vast amount of previous work in relation to clustering algorithms (both within language processing and other areas of computer science). Clustering algorithms usually belong to one of the following three categories: Hierarchical, partitional and hybrid.

2.1 Hierarchical

Hierarchical algorithms merge and split clusters to produce a nested partitioning of the data elements. Due to the way they work, hierarchical algorithms are quite efficient but they have the disadvantage that they will not correct bad clustering decisions. Well known hierarchical clustering algorithms include:

- *AGNES* - Starting with each of n elements defining a cluster, the most similar clusters are merged iteratively $n-1$ times (Kaufmann and Rousseuw, 1990).

- *DIANA* - Starting with a cluster containing all the elements, the largest cluster is split into two iteratively (Kaufmann and Rousseuw, 1990).

2.2 Partitional

Partitional algorithms are probably the most efficient clustering methods. They work by generating a single partitioning, often of a predefined size, by optimising some criterion. It is typical that they are run multiple times with different starting points, in an attempt to get better results. Partitional clustering algorithms include:

- K-means - Constructs K clusters using K random words as centroids. Then assigns each element to it's "closest" cluster until convergence, or for a predefined number of steps (McQueen, 1967).

- Bisecting K-means - A divisive variation of K-means (M. Steinbach and Kumar, 2000).

- K-menoids - Same as K-means, but at each iteration a representative element is replaced by a randomly chosen non-representative element if the criterion is improved (Kaufmann and Rousseuw, 1987).

2.3 Hybrid

Hybrid algorithms consist of multiple phases, combining the above two categories. Some hybrid clustering algorithms are:

- Buckshot - Same as K-means, but the K initial words which form clusters are selected by applying coverage-link to a random sample of $\sqrt{n}$ elements (D. R. Cutting et al., 1992).

- Birch - Constructs a structure called a *CF-tree* containing all the data and then applies any clustering algorithm on the leaves of the *CF-tree* (T. Zhang and Livny, 1996).

- Cure - Clusters are represented by a set of points that are initially well-scattered, and are shrunk towards the center of gravity of the cluster (S. Guha and Shim, 1998).

- Rock - An algorithm for clustering binary and categorical data (S. Guha and Kyuseok, 1999).

- Chameleon - Combines the advantages of CURE and ROCK while employing dynamic modelling of clusters to improve clustering quality (G. Karypis and Kumar, 1999).

- CBC (Clustering By Committees) - Similar to the $UNICON$ algorithm (discussed in detail below), but using committees in place of centroids (Pantel, 2003).

The $UNICON$ algorithm is a hybrid clustering algorithm. It was chosen for this analysis of the Swedish language due to its success with regards to English language compared to its complexity to implement. Another benefit was that it is unsupervised, so it could be run on a semantically unannotated corpus.

Whilst most analyses of clustering algorithms have been applied to English, several algorithms have been applied to non-English languages, for example Russian (Mitrofanova et al., 2007).

3 Dependency relationships

The structure of a sentence can be described using dependency relationships between the words in that sentence (Tesnière, 1959). The relationship between a pair of words can be represented as a directed edge, with the head pointing to the modifier. Furthermore each edge can be given a function that describes the type of relationship between the two words. Using this representation a sentence can be drawn as a tree with words being the nodes and edges being the dependency relationships. The root of the tree is known as the root of the sentence. An example sentence with its corresponding dependency tree is shown in Figure 1. There are a number of algorithms to compute these dependency relationships efficiently and to a high accuracy for a given corpus, one example is found in Nivre et al. (2004).

Each dependency relationship can be used to form a triple (w, r, w') where w is the head, w' is the modifier and r is the function. We can then use a hashtable to store the set of all dependency triples, in such a way that efficient look-ups of all triples related to a given word can be made. This set is the input for the algorithm described in Section 4. It is these dependency triples which allow the sense of a word to be determined automatically. The basic idea of the algorithm is to cluster together words which share similar dependency triples.

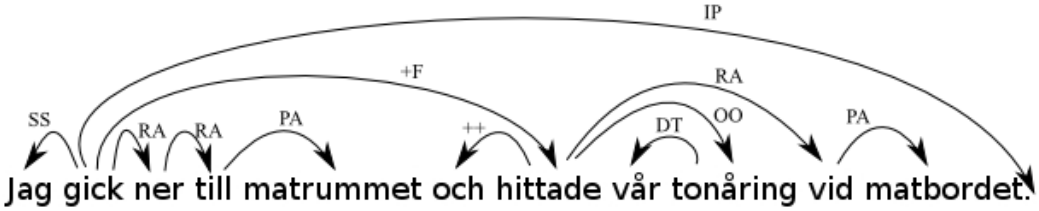

Figure 1: Dependency relationships for a Swedish sentence. The letters over the edges show the function of the dependency relationship.

4 The algorithms and their implementations

The main algorithm that we implemented was $UNICON$ (Lin and Pantel, 2001) and this in turn used two other algorithms as sub-routines: computing maximal cliques (Lin and Pantel, 2001) and computing similarity matrices (Lin, 1998). All three algorithms broadly followed their original designs. However, we made improvements to allow some parts of the computations to be done using parallel processes. The two smaller algorithms are described first, and the main algorithm afterwards.

4.1 Similarity matrix algorithm

The similarity matrix is a measure of how similar each pair of words are and the algorithm to construct it is described in Algorithm 1. The algorithm is largely based on Lin (1998), but some improvements were made. The input to the algorithm was the set of dependency triples described above and the output was for each word a list of similar words with their similarity score.

The mutual information content $I(w, r, w')$ of two words w and w' for a function r is defined as,

$$I(w, r, w') = \log \frac{||w, r, w'|| \times ||*, r, *||}{||w, r, *|| \times ||*, r, w'||}$$

where $||w, r, w'||$ is the number of occurrences of the dependency triple (w, r, w') in the corpus, and a $*$ represents a wildcard, e.g. $(*, r, *)$ means all triples with function r. $T(w)$ is the set of (r, w') for word w for which the mutual information is positive. Finally the similarity between two words w_1 and w_2 is

$$\text{sim}(w_1, w_2) = \frac{\sum_{(r,w) \in T(w_1) \cap T(w_2)} (I(w_1, r, w) + I(w_2, r, w))}{\sum_{(r,w) \in T(w_1)} I(w_1, r, w) + \sum_{(r,w) \in T(w_2)} I(w_2, r, w)}$$

This algorithm was particularly suited for multiple parallel computation and we did this in three main areas. The frequency of each dependency triple was counted in parallel. This was done by each parallel process counting a part of all the triples and the counts being merged afterwards. The information content of each dependency triple $I(w, r, w')$ could be computed independently from every other triple. Thus, $I(w, r, w')$ for all triples could be computed using parallel computations, with each parallel process computing a part of all the mutual information values and the results being merged at the end. This was also similarly true for $T(w)$. Once I and T had been computed for all words, the similarity score between all pairs of words (with the same part of speech (POS) tags) could be computed. The similarity score for each pair of words could be calculated independently from all other pairs of words and so parallel computations could again be used. Each parallel process would compute a portion of all similarity scores and they would be merged at the end.

4.2 Maximal cliques algorithm

The $CLIMAX$ algorithm is almost the same as described in Lin and Pantel (2001) and is shown in Algorithm 2. However, instead of using a heuristic to compute the maximal cliques we used an exact algorithm, as described in Bron and Kerbosch (1973). Computing maximal cliques is NP-Complete, however by using small graphs (setting

Algorithm 1 SIMMATRIX

Input: Set of dependency triples.
Count the frequency of each dependency triple (w, r, w') using parallel computations.
Compute information content of each triple, $I(w, r, w')$ and $T(w)$ using parallel computations.
for all pairs of words, w_1, w_2 with the same POS tag. **do**
  Compute sim(w_1, w_2) using parallel computation if the number of occurrences of w_1 and w_2 is greater than some threshold.
end for
for all words, w. **do**
  Compute a list of the N most similar words to w, storing both the similar word and the similarity score.
end for
Output: For each word a list of the N most similar words and the similarity scores.

n small) this could be computed in a reasonable amount of time, and generally produced better results. At the end of the algorithm cliques which overlapped were removed. We decided that the degree of overlapping could be set by the user to ensure flexibility.

Algorithm 2 CLIMAX

Input: A similarity matrix M for a list of elements E and a number n.
for all $e \in E$ **do**
  $S_e \leftarrow e$ and top n most similar elements to e
  $C_e \leftarrow$ collection of all maximal cliques in S_e using (Bron and Kerbosch, 1973)
  $C \leftarrow \bigcup_{e \in E} C_e$
end for
Sort the collection of cliques in C in descending order of clique size and average similarity among clique members.
Remove cliques in C that have some overlap with higher ranked cliques.
Output: A list of clusters.

4.3 $UNICON$ algorithm

The $UNICON$ algorithm is the same as described in Lin and Pantel (2001). The pseudocode has been modified slightly to fit in with our implementation and is shown in Algorithm 3. The input is the set of dependency triples described in Section 3 and a number which decides how big the graphs in $CLIMAX$ are.

Algorithm 3 $UNICON$

Input: A set of dependency triples D and a number n.
$M \leftarrow SIMMATRIX(D)$ and set E to be the list of words in M.
$C \leftarrow CLIMAX(M, E, n)$
repeat
  for all $c \in C$ **do**
    Compute the centroid of c.
  end for
  $M' \leftarrow SIMMATRIX$(all centroids), the similarity matrix between all the centroids.
  $S \leftarrow CLIMAX(M', C, n)$, where S is a collection of subsets of C. Each subset is a cluster of clusters.
  for all elements in S **do**
    Remove the clusters from C that belong to the element and create a new cluster in C that is the union of these clusters.
  end for
until S is empty.
Compute the centroids of all the clusters in C and add them as pseudo-words in D.
Find all the similar words, above a certain threshold to the centroids, and add them to the corresponding clusters and store their similarity to the centroid.
for all words $w \in C$. **do**
  Find all clusters w belongs to and the similarities to the clusters.
  Remove w from clusters where it's similarity score is lower than 90% of the highest similarity between w and all clusters.
end for
Output: C, a list of clusters.

A centroid of a cluster is a pseudo-word which is representative of the cluster. That is to say the centroid's features are the average of all the features of all the words in the cluster. Algorithm 4 shows how a cluster's centroid can be computed. By computing the similarity score between a centroid and a particular word we can see how similar the word is to the cluster. This is important when deciding membership of words in each cluster.

The output of $UNICON$ is a list of clusters, where each cluster has a unique name and POS, and a list of the words which are included in each of the clusters. The words in each cluster share

Algorithm 4 Compute centroid

Input: A cluster of words c, and a set of dependency triples D.
Create the centroid, t, which initially has no dependency triples associated with it.
for each word w in c. **do**
  for each distinct dependency triple (w, r, w') of w **do**
    Count the number of occurrences of the triple.
    Average this number by multiplying the count by the number of occurrences of w and dividing by the total number of dependency triples for the cluster.
    Append this number of dependency triples (t, r, w') to t.
  end for
end for
Output: The centroid t.

the same POS with the cluster. It is also possible to output the clusters' centroids as pseudo-words, which would allow words not in the corpus to be placed into clusters (by computing the similarity between the word and centroid).

5 Implementation

5.1 Preprocessing the corpus

The corpus that we ran the $UNICON$ algorithm on was the first half of the Proceedings of the EU in Swedish (Koehn, 2005), which contained around 16 million words. The corpus comes completely unannotated, so in order to get it ready for input for our implementation we needed to transform it into a form where the dependency relationships had been calculated for each sentence. The corpus was first tokenised and afterwards tagged with suitable POS. We chose Granska (Domeij et al., 2000) to do both of these as it is one of the best POS taggers for Swedish. After this the POS tagged corpus needed to be dependency parsed. We used MaltParser 0.4 (Nivre et al., 2007) to achieve this as it has a high accuracy for Swedish. The machine that we used for these tasks had an 8-core 64-bit processor and 32G of RAM available. Granksa ran very fast (less than 2 hours for 16 million words), whilst Maltparser took considerably longer (18 days for 16 million words). In order to process the corpus in a reasonable amount of time the corpus was split into smaller sections and Maltparser was run in parallel on each section, recombining the corpus afterwards.

5.2 Implementation details

For the actual implementation of the algorithms described in Section 4 we used Python. The implementation for computing the similarity matrix was optimised to use multiple processors as described above. In $SIMMATRIX$ we required that each word appeared in the corpus at least 100 times before the similarity score between it and any other word was computed. In $CLIMAX$ we used the Python NetworkX package[1] to compute the maximal cliques, and this library used the algorithm described in Bron and Kerbosch (1973). The size of the graphs we used were $n = 10$, which ensured the maximal cliques could be found in a reasonable amount of time. The amount that two clusters needed to overlap before we removed the smaller cluster was 70%. In $UNICON$ we set the minimum required similarity score between a word and a centroid to place the word into the cluster at 0.5.

Using the corpus and machine described above, our implementation of $UNICON$ required slightly more than 31 hours to produce the clusters. Computing the similarity between words (in the $SIMMATRIX$ algorithm) took the largest portion of the time. Due to RAM restrictions and the implementation of Python's multiprocessing package (that the processes could not share the data between themselves and had to keep their own copy in memory), only two processor cores could be run at the same time. Given larger amounts of RAM or being able to share the data in memory between parallel processes, we could have run the program using more processors in parallel and this would have greatly reduced the total running time.

6 Results

An informal discussion of what was observed is presented first, and afterwards there is a more quantitative analysis.

6.1 Observations

The results when running the algorithm on the first half of the proceedings of the EU in Swedish consisted of 6827 clusters. Some of them were very good, some were generally good but could contain one or two words that fitted poorly, and some were

[1] Available from http://networkx.lanl.gov/

bad. Here are some examples of clusters which vary in quality:

- Good - berört, redogjort, undvikit, belyst, behandlat, skildrar, undersökt, löst, formulerat, debattera, angett, studera, diskuterat, analyserat (touched, described, avoided, highlighted (a topic), considered/discussed, portray, investigated, solved, formulated, debate, reported, study, discussed, analysed)
- Good - avsky, misstro, misstänksamhet, motvilja (disgust, distrust, suspicion, dislike)
- Partially good - värdesätter, uppskattade, prägla (appreciate, appreciated, characterise)
- Partially good - anslagit, tilldelade, kontrakterat, avsatte (granted, alloted, contracted, allocate)
- Bad - välla, slumrade, missbrukas, växlas, checka, sammanträder, loggade (well, slumbered, abuse, change, check, meet/assemble, logged)
- Bad - inbjudas, vidhålla, renoverades (invite, hold to, renovated)

Generally, clusters seemed to be small, consisting of a few words. There were 808 clusters which consisted of no words at all. It was reasoned that a cluster with no members must have a poor centroid as all words had some better cluster(s) to belong to. There were also 2222 clusters with only one word as member, and these were also regarded as poor clusters. Either too general that almost no words are similar, or too specific to a single word. Additionally, some clusters were of an extremely large size and were so general that the words inside were rarely similar.

6.2 Evaluating the clusters

In order to assess the quality of the algorithm we required a quantitative analysis method for the entire output. To achieve this, we randomly picked 100 words that belonged to at least one cluster with size greater than 1. For each of these words a list was made containing all the clusters that the word was a member of, which gave 191 clusters in total. This list was handed out to two native Swedish speakers and they were asked to assign to each cluster one of the following scores:

1 The word does not fit well in the cluster.

2 It is hard to decide whether it fits or not.

3 It fits well in the cluster, but also fits some other cluster that has been marked with a 4 already.

4 It fits well in the cluster, and the cluster could not be merged with some other.

Table 1: The number of clusters assigned each score by the average of the two judges.

score	appearances on average
1	116
2	5
3	9
4	61
total	191

Using this scoring, it was possible for us to find out if the clusters are generally good, bad or ambiguous. This would then reflect on the quality of the algorithm and implementation, and its ability to correctly cluster words into their correct meanings.

6.3 Results of the evaluation

The scoring from the two judges were averaged and Table 1 shows the results of the evaluation.

As more than a third of the clusters were judged as good (score of 3 or 4) we have shown that this algorithm can be applied to a Swedish corpus and obtain a reasonable proportion of sensible clusters. However, around two thirds of the clusters were marked as bad and this shows that there is still room for improvement. One simple way would be the use of a larger corpus which would improve the quality of clusters. An impressive result was that the merging of similar clusters seemed to work very well, as there were a low number of clusters with a score of 3.

The evaluation was unfortunately not as robust as it could have been. Due to the nature of the corpus it contains a lot of words that are political or legal or used in a political/legal context. This meant that meanings of the words that the evaluators were not aware of could have been used, making a cluster be marked as poor when it fact it was good. This was suggested by fact that there were several clusters which were marked as 1 by one judge and 4 by the other. If judges whom were familiar with Swedish legal/political terms had con-

ducted the same evaluation the results would probably have been better. We can also note that the output for this algorithm would be suitable for a semi-automatic analysis, which could further improve the quality of analysing the clusters.

7 Future work

The most important improvement to the implementation would be to optimise the code, both in terms of memory usage and doing less computations, This would have allowed larger corpora to be evaluated in a reasonable amount of time (which would improve cluster quality). Not too much could have been done to reduce the number of computations, but major improvements in memory usage could have been made if some lower level language like C could have been used. However, this would have significantly increase the coding effort required. Alternatively the implementation could have been redeveloped to use a database with efficient look-ups to store the data (e.g. SQL database), instead of storing them into less memory efficient data structures.

Other improvements could be made by implementing a better (more complex) algorithm like Clustering by Committees (Pantel, 2003), which supersedes the $UNICON$ algorithm. This would give a higher quality of clustering using the same corpus.

The Proceedings of the EU is available in multiple languages and it would be interesting to see whether running this implementation on the other languages would produce exactly the same clusters, similar clusters or completely different clusters to the ones from the Swedish version. If the clusters were in fact similar for the different languages then possibly the clusters could be used to translate texts. Assign the word to be translated to a cluster in the original language, find the corresponding cluster in the other language and choose a suitable word within that cluster.

8 Acknowledgments

We would like to thank our supervising professor Mr. Pierre Nugues, for his help, support, and for providing us with access to a computer capable to perform the required task. Also we would like to thank Hanna Karlsson, and Cenny Wenner, for conducting the evaluations.

References

C. Bron and J. Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16:575–577.

D. Karger D. R. Cutting, J. Pedersen, and J. W. Tukey. 1992. Scatter/gather: A cluster-based approach to browsing large document collections. *Proceedings of SIGIR-92*, pages 318–329.

Richard Domeij, Ola Knutsson, Johan Carl-berger, and Viggo Kann. 2000. Granska an efficient hybrid system for swedish grammar checking. *In Proceedings of Nodalida 99*, pages 49–56.

E. H. Han G. Karypis and V. Kumar. 1999. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer: Special Issue on Data Analysis and Mining*, 32(8):68–75.

L. Kaufmann and P. J. Rousseuw. 1987. Clustering by means of menoids. *Dodge, Y. Statistical Data Analysis based on the L1 Norm*, pages 405–416.

L. Kaufmann and P. J. Rousseuw. 1990. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. *MT Summit 2005*.

D. Lin and P. Pantel. 2001. Induction of semantic classes from natural language text. *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001*, pages 317–322.

D. Lin. 1998. Automatic retrieval and clustering of similar words. *COLING-ACL98, Montreal*.

G. Karypis M. Steinbach and V. Kumar. 2000. A comparison of document clustering techniques. Technical Report 00-034, Department of Computer Science and Engineering, University of Minnesota.

J. McQueen. 1967. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematics, Statistics and Probability*, 1:281–298.

O. Mitrofanova, A. Mukhin, P. Panicheva, and V. Savitsky. 2007. Automatic word clustering in russian texts. *TSD 2007 proceedings*, pages 85–91.

J. Nivre, J. Hall, and J. Nilsson. 2004. Memory-based dependency parsing. in ng, h. t. and riloff, e. (eds.). *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL), 2004*, pages 49–56.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kbler, S. Marinov, and E. Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):5–135.

Patrick Pantel. 2003. Clustering by committees. *Ph.D. Dissertation. Department of Computing Science, University of Alberta*.

R. Rastogi S. Guha and S. Kyuseok. 1999. Rock: A robust clustering algorithm for categorical attributes. *Proceedings of ICDE'99*, pages 512–521.

R. Rastogi S. Guha and K. Shim. 1998. Cure: An efficient clustering algorithm for large databases. *Proceedings of SIGMOD-98*, pages 73–84.

R. Ramakrishnan T. Zhang and M. Livny. 1996. Birch: An efficient data clustering method for very large databases. *Proceedings of SIGMOD-96*, pages 103–113.

Lucien Tesnière. 1959. *Éléments de syntaxe structurale*. Klincksieck, Paris.