Unsupervised Morphological Word Clustering

Sergei A. Lushtak

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

University of Washington

2012

Committee:

Gina-Anne Levow

Fei Xia

Program Authorized to Offer Degree:

Computational Linguistics Master of Science

Abstract

This thesis describes a system which clusters the words of a given lexicon into conflation sets (sets of morphologically related words). The word clustering is based on clustering of suffixes, which, in turn, is based on stem-suffix co-occurrence frequencies. The suffix clustering is performed as a clique clustering of a weighted undirected graph with the suffixes as vertices; the edges weights are calculated as similarity measure between the suffix signatures of the vertices according to the proposed metric. The clustering that yields the lowest lexicon compression ratio is considered the optimum. In addition, the hypothesis that the lowest compression ratio suffix clustering yields the best word clustering is tested. The system is tested on the CELEX English, German and Dutch lexicons and its performance is evaluated against the set of conflation classes extracted from the CELEX morphological database (Baayen, et al., 1993). The system performance is compared to that of other systems: Morfessor (Creutz and Lagus, 2005), Linguistica (Goldsmith, 2000), and χ^2 - significance test based word clustering approach (Moon et al., 2009).

Table of contents

1. Introduction

Morphological analysis is used in many NLP applications, such as spell checkers, Machine Translation, Information Retrieval, Speech Recognition, Text-to-Speech systems and many others.

Various systems that perform morphological analysis have existed for a substantial amount of time. The systems developed in the twentieth century mostly used the supervised approach to morphological analysis. For instance, stemmers have existed since the 1960s (Schone and Jurafsky, 2001), like, Lovins (1969), Chapin and Norton (1969), Porter (1980). There have been other supervised systems, like the Two-Level Morphology by Koskeniemmi (1983) and supervised morphology learning by Olfazer et al. (2001). There are also 'weakly' supervised systems, which use small training sets, like Shalonova and Golénia (2010).

There is one important deficiency that all those systems share: in one way or another and to various degrees they rely on human compiled data, which means probable human errors, long time spans and high expense. Another issue is maintenance: languages change and human made annotated data or rule sets have to be updated to include new words and their morphological characteristics. In addition, supervised systems are generally language dependent.

Within the last 15 years, however, more and more work has been done in the area of unsupervised morphological induction from large corpora with the use of various statistical techniques. This became possible because of the large quantities of text in many languages that are available on the Internet. It is also preferable for many applications since it does not require

much of the expensive human labor. In addition, many of the new approaches to unsupervised morphology acquisition do not directly depend on the language they process.

According to Moon et al. (2009) the tasks of unsupervised morphological acquisition can be divided in three main categories:

1. Segmentation of words. This can include breaking the words into morphemes as well as labeling the morphemes. Much work has been done in this area, for instance, Goldsmith (2000), Bernhard (2006), Demberg (2007), and others. This is an important part of morphological analysis.

2. Clustering words into conflation classes, where conflation classes are defined as sets of words morphologically related to each other, such as "abuse", "abusive", "abused" (Schone and Jurafsky, 2000, 2001). This can be useful for building Machine Readable Dictionaries (Schone and Jurafsky, 2000, 2001), generating "interlinearized glossed texts" in endangered or under-resourced languages (Moon, 2009), and for other applications such as dictionary compression for devices that use word prediction.

There has been some work done in word clustering, e.g. Schone and Jurafsky (2000) do morphological segmentation using semantic context for English, Schone and Jurafsky (2001) improve the results of their previous work by using syntactic context and orthographic probabilities in addition to the semantic context and applying the algorithm to English, German and Dutch; Moon et al. (2009) cluster words of English and Uspanteko (an endangered Mayan

language) using χ^2- significance testing of stem-suffix co-occurrence counts and, also, document boundaries.

3. Generation of Out-Of-Vocabulary words. This is a very interesting topic, which is, however, outside of the scope of this thesis.

Not too long ago I was working on a project that involved converting sizable lexicons (50,000 - 2,000,000 word types) in different languages to a given compression format. The format was based on a compact representation of certain lexicon subsets, namely, the words sharing the same stem. Any such subset would be represented by the shared stem and a sequence of suffix categories, where every word of the subset would be a concatenation of the stem and a suffix from one of the categories. This format was taking advantage of the stem-suffix regularities and it could significantly reduce the size of the lexicon, especially, in cases of morphologically rich languages. The key to the compression, of course, was finding a good set of stems and suffix categories. My task was to design an algorithm that would find such sets. The algorithm had to work on any lexicon and, thus, be language independent and fully unsupervised.

After having tried a number of different approaches I, finally, came up with the needed algorithm. While working on the problem I came across the Moon et al. (2009) paper. Their approach was very interesting to me because they also were splitting words in two segments, stem and suffix, and they were also looking for suffix categories. Their purpose, however, was different: using the obtained suffix categories they were clustering words into conflation classes.

I have tried their approach to word segmentation and suffix clustering, but it did not work too well for my particular problem. It did, however, give me an idea to try clustering words using my suffix clustering algorithm.

This thesis describes a system based on that algorithm: the lexicon words are segmented into stem-suffix pairs, the suffixes are clustered in a few different ways, and the clustering that provides the best lexicon compression is considered the optimum. The suffix clusters are used to generate the conflation classes the same way as Moon et al. (2009) do it.

The segmentation part of the system is rather simplistic but intuitive and quite fast compared to e.g. Morfessor (Creutz and Lagus, 2005) or Linguistica (Goldsmith, 2000). The suffix clustering is more elaborate and time consuming: the suffixes are considered vertices of a weighted graph, where the weight of an edge connecting two suffixes is set to be the similarity measure between their signatures (a suffix signature is defined as the set of stems with which the suffix co-occurs in the words of the lexicon). The similarity measure between two suffix signatures is calculated in the following way:

$$\mu(e_i, e_j) = \frac{|Sig\ (e_i) \cap Sig\ (e_j)|}{min\ (|Sig\ (e_i)|, |Sig\ (e_j)|)} \quad (1.1)$$

Two suffixes are considered members of the same cluster if and only if the weight of the edge connecting them ($\mu(e_i, e_j)$) is greater than a certain threshold value. The clustering is done for 21 different threshold values. The compression ratio is calculated for each clustering. The word conflation classes are generated using the clustering that yields the best compression ratio.

Testing of the hypothesis that the clustering that yields the best compression ratio produces the best conflation classes set is a part of this work.

The system was developed using a Russian lexicon containing a little over 400,000 words and an Italian lexicon containing over 2,000,000 words.

The testing is done on a lexicon extracted from the CELEX morphological database (Baayen, et al., 1993) for English, German, and Dutch. The obtained conflation classes were evaluated against the gold standard: a set of conflation classes also extracted from the CELEX database.

The conflation class extraction is implemented based on the description of the concept of a conflation class provided in Schone and Jurafsky (2000) and in Schone and Jurafsky (2001).

2. Literature review

The literature reviewed here is on various approaches to unsupervised morphological processing.

The first two articles (Bernhard, 2006 and Demberg, 2007) are on word segmentation and segment labeling. Another two (Schone and Jurafsky 2000 and 2001) are on word segmentation with the use of semantic and syntactic contexts and orthographic probabilities. The conflation classes based on the segmentation are used as means of evaluating it - they are compared to those extracted from the CELEX morphological database. Schone and Jurafsky (2000) also come up with the evaluation metric on which I base the evaluation metric of my own.

Next, Moon et al. (2009) article is reviewed. This work was particularly interesting to me because the approach Moon et al. (2009) use is similar to mine: they break each word into two segments and use the suffix clustering in order to cluster words. Moon et al. (2009) also represent suffixes with the sets of stems the suffixes co-occur with; the similarity measure they use for the clustering is the χ^2 test, which seemed more statistically sound than the measure I came up with.

I have implemented their approach, ran the implementation on the same lexicon and evaluated the obtained classes against the same gold standard class set. Additionally, Moon et al. (2009) propose the evaluation metric based on that of Schone and Jurafsky (2000), extended to handle overlapping conflation classes. I tried to use this metric until I found problems with it.

In addition to Moon et al. (2009), I use three other systems for results comparison: Linguistica

(Goldsmith, 2000), Morfessor Baseline (Creutz and Lagus, 2005) and Morfessor Categories-MAP (Creutz and Lagus, 2005). Linguistica and Morfessor Baseline do word segmentation using different approaches, while Morfessor Categories-MAP does morpheme labeling in addition. The articles that describe these systems are also reviewed in this section.

2.1. Word segmentation based on low predictability of letters at morpheme boundaries (Bernhard, 2006)

In this paper a method of unsupervised word segmentation is presented. The method is based on low predictability of letters and strings of letters at morpheme boundaries. Given a flat word list the algorithm based on this method segments the words and labels each segment as one of the four following categories: prefix, suffix, stem, and linking element (an example of a linking element is given: in the word 'hormon*o*therapy' the 'o' between 'hormon-' and '-therapy' is a linking element).

2.1.1. Stage 1: Prefix and suffix extraction

The word list *L* is sorted in the order of descending word length, so that the affix extraction starts from the longest words in the list. The words boundaries are marked with "#".

Let n be the length of the word including boundary markers and $s\ (i, j)$ a substring of the word

starting from thee *i-th* character and ending with the *j-th*, $0 \le i < j < n$.

The mean of maximum transition probabilities is calculated $\forall k : 0 < k < n$,

$$f(k) = \frac{\sum_{i=0}^{k-1} \sum_{j=k+1}^{n} max\ \left(P\left(s\ (i,j) \middle|\ s\ (k,j)\right),\ P\ (s\ (k,j) \mid s\ (i,k))\right)}{k\ (n-k)}, \qquad (2.1)$$

where probability $P\left(s\ (i,k) \middle|\ s\ (k,j)\right)$ is estimated as $\frac{fr\left(s\ (i,j)\right)}{fr\left(s\ (k,j)\right)}$, while the probability $P\ (s\ (k,j) \mid s\ (i,k))$ is estimated as $\frac{fr\left(s\ (i,j)\right)}{fr\left(s\ (i,k)\right)}$, and $fr\ (s)$ is a number of times substring s occurres in L.

Those k where $f(k)$ has a pronounced local minimum are considered potential segment boundaries. 'Pronounced' here means that the minimum's "difference both with the preceding and following maximum" is "at least equal to a standard deviation of the values."

Once the word has been segmented, the longest and the least frequent segment is chosen to be the word's stem if it (the stem) appears at least twice in *L* and at least one word begins with it. The segment preceding the stem is added to the prefix list if it is more frequent then the stem. The segment following the stem is added to the suffix list if it is more frequent then the stem.

The process of affix acquisition stops when more than half of affixes acquired from N current words had already been acquired before, where N is a parameter of the system. (2.2)

2.1.2. Stage 2: Stem acquisition

At this stage all the words in *L*. are stripped of the affixes known so far. The remaining word segments are considered stem candidates. A list of stems is formed out of the candidates that satisfy the following stem constraints:

1. The minimum length of a stem is 3.
2. A stem must be followed by at least 2 different letters, or a letter and the word boundary marker. A stem cannot contain a hyphen.
3. For every stem there must be at least one word that begins with it.

2.1.3. Stage 3: Word segmentation

Words sharing the same stem are compared. Using previously acquired lists of segments the limits between shared and different ones are established.

The segments are labeled according to their positions relative to the stem either prefix, or suffix, or linking element. Those segments containing other stems are labeled 'potential stems'.

The words compared will likely have segments that were not present in the previously acquired affix lists. These segments are validated. The following is the validation procedure.

From a set of words sharing one stem those starting from the same segment are picked out. For this subset the obtained segments are counted. The author offers the following example:

housekeeping = hous + ekeeping

housing = hous + ing

household = hous + ehold

house’s = hous + e’s

house = hous + e

housed = hous + ed

This is a subset of words sharing the stem 'hous'; '-ekeeping' and '-ehold' are 'potential stems'; let '-ing', '-e' and '-ed' be suffixes acquired at stage 1, while "-e's" is a 'new suffix', a suffix discovered in the process of comparison, which needs to be validated.

Let A_1 be the set of previously acquired suffixes for our set of words : A_1 = {"-ing", "-e", "-ed"}, A_2 set of potential stems: A_2 = {"-ekeeping", "-ehold"} and A_3 the set of new suffixes: A_3 = {"-e's"}.

The segmentation is considered valid if the following two constraints are satisfied:

$$\frac{|A_1|+|A_2|}{|A_1|+|A_2|+|A_3|} \geq a,$$

$$\frac{|A_1|}{|A_1|+|A_2|} \geq b, \qquad (2.3)$$

where a and b are parameters of the system.

A word may contain more than one stem so it may be segmented more than once. All valid segmentations are stored for each word.

After all stems have been analyzed in this manner, the "potential stem segments are either replaced by other segments (as a whole or only partially) or assigned a final category (prefix, suffix, or linking element) if no replacement is possible" (Bernhard, 2006).

At the end of this stage all the segments are counted. A segment frequency is the number of words whose valid segmentations contain the segment.

2.1.4. Stage 4: The best segments

The purpose of this stage is assigning each word one segmentation per stem. "In order to choose the best possible segments," the author performs "a best-first search privileging the most frequent segment, given a choice" (Bernhard, 2006). The following constraints are applied to the final segmentation of a word:

1. A word must contain at list one stem.
2. A prefix cannot be followed by a suffix.

3. Only one linking element can be between two prefixes or two suffixes.
4. The stem must be less frequent than any other types of segments.

At this point all the words in *L* are segmented. Separate lists of stems, suffixes, prefixes and linking elements are created.

2.1.5. Stage 5: Segmenting other words using learned segments

The obtained lists can be used to segment other words in the same language. Since more than one segmentation may be possible, the best segmentation must be found. First, the constraints listed in the previous stage are applied. Out of the remaining segmentations the one that has the minimum *cost* is chosen. The author uses two different cost functions, which constitute two methods:

method 1: $$cost_1(s_i) = -\log\frac{fr(s_i)}{\sum_j fr(s_j)}$$

method 2: $$cost_2(s_i) = -\log\frac{fr(s_i)}{\max_j fr(s_j)} \quad (2.4)$$

2.1.6. Validating the segmentation

The system was run on the Morpho Challenge 2005 datasets. The languages were English, Finnish, and Turkish. Because of the memory limitations the system could only use subsets of Finnish and Turkish corpora – the 300,000 most frequent words from each corpus. The system has 4 parameters: N (2.2), a, b (2.3), and method (2.4).

The best performance was shown for English (F-measure = 66.6) using method 1 (2.4) with the parameters a = 0.85 and b = 0.1 (2.3). For Finnish and Turkish method 2 worked better; it yielded F = 64.7 with a = 0.8 and b = 0.1, and F = 65.3 with a = 0.7 and b = 0.1 respectively. For N (2.2) the value 5 worked best for all three languages.

2.2. Word segmentation using modified RePortS algorithm (Demberg, 2007)

This is another paper on unsupervised morphological segmentation. It describes a modification of the RePortS algorithm (Keshava and Pitler, 2006). The RePortS algorithm itself is also described.

This algorithm is chosen because it performed best on English in Morpho Challenge 2005, a competition on unsupervised morphological segmentation (Kurimo et al., 2006). It outperformed all other systems obtaining an F-measure of 76.8% (76.2% precision and 77.4% recall). The RePortS algorithm, however, had a very low recall on morphologically complex languages like German, Finnish or Turkish. That is why the modification is proposed: a new step was developed to achieve higher recall on those languages and a method of identifying related stems that

underwent regular non-concatenative morphological processes such as umlauting or ablauting, as well as morphological alternations along morpheme boundaries was designed.

2.2.1. The RePortS Algorithm

This algorithm is based on low predictability of the next letter at the suffix boundary and previous letter on a prefix boundary. It consists of three steps.

Step 1. Segmentation.

The set of words in the word list is represented as a forward trie and a backward trie. Each of the trie nodes corresponds to a letter. Each node is annotated with the ancestor sequence frequency counted over the whole corpus.

With the use the forward trie the probability of seeing each letter given the previous letter sequence is calculated. These probabilities are used in acquisition of suffixed. For acquisition of prefixes the backward trie is used in a similar manner: the probabilities of seeing a letter given the following letter sequences are calculated.

The places where such transitional probability is low are hypothesized to be morpheme boundary candidates.

Step 2. Finding good affixes.

All possible affixes are generated by splitting the words at morpheme boundary candidates obtained in the previous step and then validated. For the validation the following criteria are used: if for at least 5% of the affix occurrences

1. the stem, which is the substring remaining after the affix has been cut off the word, is itself a word in the lexicon,
2. the transitional probability between the second-to-last and the last stem letter is close to 1, and
3. the transitional probability of the affix letter next to the stem is less than one with the tolerance of 0.02

then the affix is valid.

Next, all the compound affixes (those that consist of other affixes) are split.

The author states that the assumptions that all stems are words themselves, that affixes are only word-initial or word-final, and that affixation does not change stems might be working for English but also may cause low recall for other languages.

Step 3. Stem acquisition.

All the words are stripped of most probable affixes found in the previous steps. Transitional probabilities of the affixes must be less than 1.

At this point the proposed modification of the RePortS algorithm is described. First, it is stated that the assumption 1. should be reconsidered. For instance, in German, stems do not appear as separate words in the lexicon. Thus, an additional step is introduced: obtaining a stem candidate list. This step consists of three sub-steps:

Sub-step 1. The stems that stem that satisfy assumptions 2. and 3. but not 1. are stored together with their affixes.

Sub-step 2. Ranking stem candidates. The ranking takes into account the number of affixes that occur in the lexicon (compounding) and the average frequency of the other affixes.

Sub-step3. Pruning. The stem candidates with frequency less than 3 are removed from the list. The remaining stems are ordered according to the average frequency of their non-word affixes. This puts high quality stems on the top of the list. The list is truncated at a certain point. This point is chosen visually depending on the data: the graph of the function

$$f\colon (stem\ rank) \rightarrow (affix'\ average\ frequency) \tag{2.5}$$

changes steepness noticeably around a certain rank; that is the rank at which the list is

truncated.

Another possible drawback of the RePortS algorithm is specified: peeling off affixes without taking into account their morphological context can lead to things like chopping off an inflexional suffix after having chopped off a derivational suffix, thus, decreasing accuracy.

Another danger is that after removing potential affixes on both sides of the word the remainder may be not a valid stem, e.g. a letter sequence with no vowels.

To counter these dangers the use of bi-gram model to capture the morphotactic properties of a particular language is proposed. The language model is created and all possible segmentations of the word are ranked based on this model. This step increases F-measure significantly.

Finally, the modification addresses the problem of letter variation within the morpheme, which leads to data sparseness impeding the morphological segmentation and affix acquisition.

The stem variations are captured from examining the suffix clusters that formed during the stem acquisition step. The explanation is based on the example of the *Trainingssprung* - *Trainingssprüunge* pair.

As a result of sub-step 1 (stems are stored together with their suffixes) of step 2 (finding good affixes) '-sprung'and '-sprünge' end up in the affix list of *Training* . The affix pairs with small

edit distance are picked out from the affix list and ranked by frequencies of their occurrence. The most frequent pairs are used to make transformational rules, like u → ü..e in the above example. These transformational rules can be used for a) discovering more pairs like that in the word list and b) adjusting the frequencies accordingly during the step 1 (segmentation). For example, with the rule u → ü..e the letters u and ü are considered the same when counting frequencies.

The modified algorithm was run on training sets from Morpho Challenge. CELEX was used for evaluation. It performed the best on German, second best on English (after original RePortS) and third on Turkish (losing to Bernhard's algorithm and Morfessor Categories-MAP (Creutz and Lagus, 2005). On Finish it still yielded very low recall.

2.3. Word segmentation and clustering using semantic analysis (Schone and Jurafsky, 2000)

This system produces a set of morphological rules by using affix acquisition by co-occurrence on words from a corpus and then filters the rule set with the use of semantic analysis of the words context.

The proposed algorithm can be divided in the four following stages.

2.3.1. Finding the affix candidates

In order to find suffix candidates all the words are put into a trie (reverse trie can be used for prefix candidates) and the branches of the trie are observed. The 200 most frequent branches are collected thus forming the set of the affix candidates.

2.3.2. Building the "rulesets"

Any two branches sharing a common ancestral node are said to form an "affix rule" (Schone and Jurafsky, 2000). Two words "sharing the same root and the same affix rule" (Schone and Jurafsky, 2000) are then dubbed a "pair of potential morphological variants" or PPMV (Schone and Jurafsky, 2000).

All such pairs sharing a common affix rule form a "ruleset" (Schone and Jurafsky, 2000). All the rulesets are collected for every affix rule extracted from the data.
For example the ruleset for the rule ("s", NULL) would contain the pairs like "cars/car" and "cares/care".

At this point, the task of the system is to establish which of these PPMVs are morphologically related since not all the words sharing their beginning or ending are, e.g. "care/car". The premise on which this task is to be accomplished is that morphologically related words are usually also related semantically.

2.3.3. Building the words semantic vectors

The semantic relatedness of two words can be established by looking at with what other words these two words co-occur in the corpus. The window of (-50, +50) tokens is used to define co-occurrence. The co-occurring words are counted for every word pair and this information is presented in form of the "semantic vectors".

The authors are using the approach of Deerwester, et al. (1990) (Schone and Jurafsky, 2000) according to whom the significance of the semantic relationship between words in a corpus can be established by applying SVD (singular value decomposition) to a term-document matrix M where $M\ (i,j)$ is how many times the word w_i occurred in the document d_j.

Instead of term-document matrix Schone and Jurafsky (2000) compose $N \times 2N$ term-term matrix of $N-1$ most frequent words and use Z-scores rather than counts. $M\ (i,j)$ is, in this case, a Z-score of a how many times the *j*-th frequent word co-occurs with the *i*-th if the *j*-th word precedes the *i*-th, and $M\ (i, N+j)$ if the *j*-th follows the *i*-th (co-occurrence here is defined as appearing within the distance of 50 words or fewer). The *N*-th position is used of all the other words.

SVD is applied to M :

$$M = UDV^T \qquad (2.6)$$

where $UU^T = VV^T = I$ and D is a diagonal matrix.

$\Omega_w = U_w D_k$ is a semantic vector, where D_k is D with all $D_{ii} = 0,$ for $i > k$, and U_w is the row of U corresponding to *w*.

2.3.4. Assessing semantic relationships by correlating semantic vectors

By comparing the degree of correlation of semantic vectors within a PPMV and comparing it to those within other PPMVs it is possible to establish the probability that the words of the PPMV are semantically related.

The authors use what they call "the normalized cosine score" or NCS as a correlation. To compute the NCS they take cosines between each semantic vector Ω_w and the semantic vectors of the other 200 randomly chosen words ($\cos(\Omega_w, \Omega_u) = \frac{\Omega_w \cdot \Omega_u}{||\Omega_w||\,||\Omega_u||}$), thus getting *w*'s correlation mean (μ_w) and standard deviation (σ_w). For two words *w* and *v* the NCS between Ω_w and Ω_v is defined as

$$min_{y \in \{w,v\}} \frac{\cos(\Omega_w, \Omega_u) - \mu_y}{\sigma_y} \qquad (2.7)$$

By considering the NCSs between the semantic vectors of words within all PPMVs in a ruleset it is possible to determine whether the rule is more than random semantically.

Based on the expectation that random NCSs have a normal distribution $N(0,1)$ and the number of PPMVs in a particular ruleset (n_R), the authors estimate the distribution $N(\mu_T, {\sigma_T}^2)$ of the true correlation and the number (n_T) of terms in that distribution.

Then, the probability of a particular *NCS* being non-random is calculated:

$$P(NCS) = \frac{n_T \Phi_{\mathrm{NCS}}(\mu_T, \sigma_T)}{(n_R - n_T)\Phi_{\mathrm{NCS}}(0,1) - n_T \Phi_{\mathrm{NCS}}(\mu_T, \sigma_T)}$$, where (2.8)

$$\Phi_{\mathrm{Z}}(\mu, \sigma) = \int_Z^{\infty} e^{-(\frac{x-\mu}{\sigma})^2} dx$$ (2.9)

Thus, the PPMV $(w_{1,} w_2)$ is considered morphologically related (or pair of valid morphological variants) iff $P(NCS(w_{1,} w_2)) > T$.

2.3.5. Evaluation of the results

The results were tested against the CELEX morphological database for English as the gold standard using the metric defined in the article. I would like to review this in some detail because I am also working with the CELEX database and my evaluation metric is derived from the one defined in this paper.

At this state the words are united into the "conflation classes" according to the rulesets and these classes are compared to those extracted from the CELEX.

It is not entirely clear how the conflation classes are composed. Here is the explanation offered by the authors:

The morphological relationships can be represented by a directed graph, for example (concern, concerns, concerning) -> concern, (concerted, concerts, concerting) -> concert. Since comparing

directed graphs is “likely prone to disagreements” (Schone and Jurafsky, 2000)) authors propose to compare the sets of vertices of the graphs.

I, however, do not see how the word parings (PPMVs) can produce directed graphs, since no inequality of any kind has been indicated for the words in a pair, hence, there is no direction. If we are dealing with undirected graphs then it seems that the authors put all the words from the chains of the sort $(w_1 w_2), (w_2, w_3), \ldots (w_{n-1,} w_n)$ in one conflation class. In other words, if two words are legitimate morphological variants they belong to the same conflation class, but two words from one conflation class are not necessarily morphologically related. For instance, “art” is related to “was” and to “artful”, while “artful” and “was” are not related, but all three belong to the same conflation class.

As for the CELEX morphological database, the directedness of the relations between words is definitely present there. For example, there is a file that contains sets of inflectionally related words with the lemmata indicated, one set per lemma. These might have been used by the authors to extract the conflation classes of the gold standard from the CELEX database if it was not for the fact that some of such classes overlap. The metric used by the authors, however, does not allow for one word to belong to more than one class.

2.3.5.1. The evaluation metric

Let $X_1, \ldots, X_k$ be conflation classes produced by the algorithm and $Y_1, \ldots, Y_n$ are classes of the gold standard. Let X_w and Y_w be the classes that contain a word $w, \forall w \in L$, where L is the lexicon which contains all the words that are used both by the model and by the CELEX database.

The correct (C), deleted (D), and inserted (I) words are counted like so:

$$C = \sum_{w \in L} \frac{|X_w \cap Y_w|}{|Y_w|}$$

$$I = \sum_{w \in L} \frac{|X_w| - |X_w \cap Y_w|}{|Y_w|}$$

$$D = \sum_{w \in L} \frac{|Y_w| - |X_w \cap Y_w|}{|Y_w|}, \quad (2.10)$$

The precision (P), recall (R) and F-measure (F) are computed as follows:

$$P = \frac{C}{C+I}, \; R = \frac{C}{C+D}, \; F = \frac{2PR}{P+R} \quad (2.11)$$

Using this metric the conflation classes produced by the system with different values *T* (semantic probability threshold) are evaluated. The P, R, and F are compared to those produced by Linguistica (Goldsmith, 2000). The system presented here outperforms Linguistica and the best F is produced with $T = 0.7$.

2.4. Word segmentation and clustering using semantic analysis, local syntactic context, and branching transitive closure (Schone and Jurafsky, 2001)

This system is based on the same approach used by Schone and Jurafsky (2000), but with many improvements and it produces significantly better results, according to the authors.

First major improvement, as compared to Schone and Jurafsky (2000), is that instead of working with affixes the new system works with "circumfix pairs"(Schone and Jurafsky, 2001). Instead of affix rules like ("re", NULL) for "realign"/"align" and ("ed", NULL) for "alinged"/"alingn" it deals with one rule ("re/end", NULL).

Another difference is that while establishing the rule candidates the occurrence frequencies are used more elaborately.

And, finally, in addition to the semantics, orthographic-based probabilities, syntactic context, and "transitive branching closure" (Schone and Jurafsky, 2001) are used for rule validation.

2.4.1. Finding PPMVs

First, word beginnings with frequencies with excess of certain threshold T_1 are collected. They are dubbed "pseudo-prefixes" (Schone and Jurafsky, 2001). Pseudo-prefixes are stripped from the words and the residuals are added back to the lexicon as if they were words.

A trie is used the same way it was done in Schone and Jurafsky, (2000). "Potential suffixes" (Schone and Jurafsky, 2001) are defined as the branches that appear in the lexicon more than T_2 times.

The potential suffixes are stripped from the words of the original lexicon leaving the residuals in even if they were non-words thus creating another augmented lexicon. With the use of the reverse trie "potential prefixes" are found the same way as the potential suffixes.

Now, the concept of a "potential circumfix" is introduced as a pair A/B, where A is a potential prefix and B is a potential suffix.

The "pseudo-stem" is a residue of a word after its potential circumfix is removed.

A potential curcumfix affixed to at least T_3 potential stems is defined as a "candidate circumfix".

A pair of candidate circumfixes shared by at least T_4 potential stems form a "rule". A pair of words "sharing a rule" (Schone and Jurafsky, 2001) is PPMV (pair of potential morphological variants). Example: the rule *re/ed* =>NULL is shared by the PPMV "realigned/align". Note: the

arrow is used in the rule notation by the authors, even though it is unclear how the direction of the arrow is established.

Finally, the "ruleset" (Schone and Jurafsky, 2001) is "the set of all the PPMVs for a common rule" (Schone and Jurafsky, 2001).

2.4.2. Filtering out semantically unrelated pairs from the rulesets

This filtering is based on the premise that morphologically related words are often semantically related; it is done in the same way as in Schone and Jurafsky (2000).

2.4.3. Using the affix frequencies

This is a major improvement compared to Schone and Jurafsky (2000), according to the authors. The fact is that not all morphologically related words show high level of semantic relatedness. The semantic-based morphology filters out some legitimate word pairs. To compensate for this another criterion based on affix frequencies was added.

This is based on the assessment by Kazakov (1997) (according to Schone and Jurafsky (2001)) that the very frequent word endings are true morphological suffixes.

The concept of the "orthographic probability" of a rule ($C_1 \Longrightarrow C_2$) is introduced. It is designed to depend on frequency of the rule, the reliability of the metric in comparison to the semantics and the frequencies of other rules involving the affixes.

$$P_{orth} = \frac{2\alpha f(C_1 \Rightarrow C_2)}{max_Z f(C_1 \Rightarrow Z) + max_U f(U \Rightarrow C_2)} \quad (2.12)$$

where α is a number that reflects the importance of P_{orth} in relation to P_{sem} and is arbitrarily set to 0.5 (meaning that semantic information is twice as reliable than orthographic), $f(x \Longrightarrow y)$ is the frequency of the rule $x \Longrightarrow y$.

The overall rule probability is defined as

$$P_{s+o} = P_{sem} + P_{orth} - P_{sem}P_{orth}, \quad (2.13)$$

assuming that orthographic information is independent from the semantic.

2.4.4. The use of local syntactic context

To cover the morphologically related but not semantically related pairs even more the use of more immediate word context (in addition to the broad, 100 word semantic context window) is proposed. This is based on the premise that morphology, particularly, inflectional morphology is there to convey the syntactic information.

The syntax-based probability is added to the overall probability like so

$$P_{valid} = P_{s+o} + P_{syn} - P_{s+o}P_{syn}, \quad (2.14)$$

again, assuming independence of syntactic information from the other types.

The algorithm for calculating the P_{syn} is supplied in pseudo-code. It involves gathering the words collocated with the words in the valid PPMVs of a particular ruleset on the right and the left sides of the rule, determining the collocations that occur too many or too few times. E.g. in case of the rule ("s" $\Longrightarrow NULL$) with PPMVs like cars/car, agendas/agenda the words like "this", "a", "is" will likely be collocated with the words on the right side but not on the left side of the rule, while the words like "those", "are", "were" are a lot more probable to be found immediately close to the left side words but not to the right side ones. These sets of collocations are dubbed "ruleset signatures".

These signatures are gathered for not yet validated PPMVs as for other words in the corpus chosen randomly. The NCS are calculated in a similar way to how it is done for the semantic vectors and the syntactic validation probabilities for those PPMVs are calculated.

2.4.5. The "Branching Transitive Closure"

In order to reinstate more of the rules that got low scores due to the week corpus distribution the transitivity is used:

if (w_1, w_2),..., (w_{n-1}, w_n) are valid PPMVs, while $(\mathrm{w}_1, \mathrm{w}_\mathrm{n})$ has gotten a score too low to be valid it still may be validated depending on the score of each link and the number of links (t).

The probability of the (w_1, w_n) being valid (if it is connected with only one path of PPMVs) is defined as:

$$P = \beta^{n-1} \prod_{i=1}^{n-1} P(w_i, w_{i+1}) \,, \qquad (2.15)$$

where β is a decay factor reflecting the decay of the probability with the number of links increasing.

If there are more than one independent path connecting w_1 and w_n – let k be the number of such paths – the total probability P is a combination of the path probabilities $P_i, 1 \leq i \leq k$ calculated like so:

$P = 0$

for i from 1 to k

$P \leftarrow P + P_i - P_i P$

if P is greater than some threshold T_5 the PPMV $(\mathrm{w}_1, \mathrm{w}_\mathrm{n})$ is declared valid.

2.4.6. Building conflation classes

The valid PPMVs are assembled into conflation classes. Again, it is not entirely clear how, but it seems it is done as single-link clustering: if (w_1, w_n) is a valid PPMV then w_1 and w_n belong to the same conflation class, but the reverse is not necessarily true.

If it is indeed the single-link clustering then the merit of the "Branching Transitive Closure" is entirely defeated: if there is a PPMV chain connecting two words they will always placed into the same conflation class.

The conflation classes produced by the algorithm are compared to those extracted from the CELEX morphological database for English, German, and Dutch using the same evaluation metric as in Schone and Jurafsky (2000) and it is shown that the F-measure is significantly better than the one obtained by Linguistica (Goldsmith, 2000) and by Schone and Jurafsky (2000).

2.5. Morphological word clustering based on χ^2- significance test (Moon et al., 2009)

In this paper the authors describe a system that is designed to classify words from a set of texts in an unknown language according to their morphology.

The suggested approach involves breaking each word into two segments: a stem and an affix and clustering the words into morphologically related categories (conflation sets) with the use of the segmentation. The authors claim that their system has no parameters and, thus, does not require any tuning by the user, unlike many other systems.

The system is designed to use either the prefix-stem (for prefixal languages) or the stem-suffix segmentation (of suffixal languages). For the languages that are both prefixal and suffixal the system is supposed to be run twice on the corpus, each time with a different type of segmentation. Each run would produce a set of conflation classes. The resulting conflation classes would then be the unions of those classes from the first-run-set and the second-run-set that intersect:

$$C_{12} = \{c \cup g,\ c\ \in C_1,\ g \in C_2 : c \cap g \neq \emptyset\}. \quad (2.16)$$

Since the system's algorithm works for suffixes and prefixes in similar ways I will be discussing only stem-suffix segmentation.

The algorithm consists of four distinct steps:

1. Candidate list generation
2. Candidate filtering
3. Suffix clustering

4. Word clustering

All the steps are performed on every document and on the entire corpus ignoring the document boundaries. Processing the documents separately is based on the premise that morphologically unrelated words that are orthographically related are unlikely to appear within the same document.

2.5.1. Candidate list generation

A prefix tree (trie) is built out of the words from each document and one trie out of the words collected from the entire corpus.

A trunk is defined as a path that goes from the root node of a trie to any node that has more than one child and includes that node; a branch is defined as a path connecting a child of a node that has more than one child and a leaf including the leaf. Thus, where the trunk ends, two or more branches begin.

A stem candidate is defined as a trunk which is longer than any of its branches. Any branch of a stem candidate is dubbed a suffix candidate.

A pair of suffix candidates (ec_1, ec_2) that occur with the same stem forms a ‘pair rule’ candidate.

2.5.2. Candidate filtering

For every pair rule candidate (ec_1, ec_2) the following counts are obtained:

1. The count of stem candidates ec_1 and ec_2 co-occur with.
2. The count of stem candidates only ec_1 occurs with.
3. The count of stem candidates only ec_2 occurs with.
4. The count of stem candidates neither ce_1 nor ce_2 occurs with.

Only word types are used for counting; token counts are ignored.

Using the obtained counts the χ^2 measure is calculated with each cell in the contingency table greater than 5.

The suffix pairs forming the pair rules not meeting the test criteria of $p < 0.05$ are filtered out. The suffix candidates that are not in any or the remaining rules are filtered out as well. The stem candidates whose all suffixes were filtered out are also filtered out. The candidate filtering is done for every document and for the entire corpus.

2.5.3. Suffix clustering

The suffixes are clustered by a very simple criterion: any two suffixes belong to one cluster if and only if they form a pair rule. The clusters defined in this way may overlap: one suffix may belong to more than one cluster.

2.5.4. Word clustering

Two words belong to the same conflation class if and only if their suffixes belong to the same cluster.

The result of this four-step algorithm is a set of sets of conflation classes: a conflation class set corresponding to each document and one conflation class set corresponding to the whole corpus.
The algorithm was run on corpora of two languages: English and Uspanteko, an endangered Mayan language. For English the two subsets of NY Times portion of Gigaword corpus were used: one containing 1,000 articles, 88,000 word types; the other one much smaller: 190 documents, 15,000 types. For Uspanteko a set of 29 texts was used, which had total of 7,000 types.

For English the results were evaluated against the CELEX database (Baayen et al., 1993), which has lists of inflectional variants of for a number of words. The performance of the system was measured based on how the sets of words the system judged to be morphologically related

overlap with the entries of CELEX. A special evaluation metric based on that of Schone and Jurafsky (2000) was designed to accommodate the fact that one word can be in more than one conflation class. I am going to describe it here, because I tried to use it in my experiments.

For each word *w* of the lexicon let $\{X_w\}$ be the set of conflation classes that contain *w* produced by the system and $\{Y_w\}$ the set of gold standard classes that contain *w* (in contrast to Schone and Jurafsky (2000) where there is only one X_w and one Y_w). The numbers of correct (C), inserted (I), and deleted (D) words are calculated by adding up the C, D and I for all possible triples (w, X_w, Y_w):

$$C = \sum_{w \in L} \sum_{X_w} \sum_{Y_w} \frac{|X_w \cap Y_w|}{|Y_w|}$$

$$I = \sum_{w \in L} \sum_{X_w} \sum_{Y_w} \frac{|X_w| - |X_w \cap Y_w|}{|Y_w|}$$

$$D = \sum_{w \in L} \sum_{X_w} \sum_{Y_w} \frac{|Y_w| - |X_w \cap Y_w|}{|Y_w|}. \qquad (2.17)$$

The precision, recall and F-measure are computed the same way as in Schone and Jurafsky (2000):

$$P = \frac{C}{C+I},\ R = \frac{C}{C+D},\ F = \frac{2PR}{P+R} \qquad (2.18)$$

For Uspanteko the set of 100 word clusters produced by the system were reviewed by a Mayan linguist.

The model performance was compared to that of Morfessor and Linguistica. Since both Morfessor and Linguistica provide only segmentation, the authors do the clustering themselves.

Linguistica's output is a set of stem-suffix pair for each word and "it clusters naturally" (Moon et al., 2009). I assume this means that the words sharing the same stem belong to the same cluster and vice-versa.

With Morfessor it is a bit more complicated: the Morfessor produces generally more than two segments per word. Moon et al. (2009) use Morfessor Baseline, which does not label the segments, that's why the authors take the longest segment as a stem and then cluster the words according to that. Again, I assume, they use common stem as the necessary and sufficient condition of two words belonging to one cluster.

According to the authors, the system did better than both Linguistica and Morfessor. The best results were achieved by using the whole corpus trie for candidate generation and document tries for clustering, which came as a surprise to the authors: they expected the best results to come from both candidate generation and clustering being done using the document tries. Another thing that authors consider alarming is that the model performed considerably better on the smaller corpus for English than on the bigger one.

2.6. Linguistica (Goldsmith, 2000)

Like Moon et al. (2009) I compare my results to those obtained with the use of the system called Linguistica. This system is designed to process a corpus ranging from 5,000 to 1,000,000 words and represent every word type as a stem-suffix pair with the suffix possibly being empty. No word clustering is done by this system. The word clusters that I use to compare my system's results to are the sets of words sharing the same stem.

The algorithm used by Linguistica is based on the MDL (Minimum Description Length) analysis (Rissanen, 1989): it tries to find a segmentation that would allow to represent the corpus in the shortest possible way.

2.6.1. The corpus representation via a segmentation

The corpus is never actually represented via a segmentation – there is no need for that; it is enough to be able to calculate the representation length (referred to as the "corpus compressed length") in order to find the best segmentation by minimizing it.

The corpus representation consists of the description of the corpus data with the use of a morphological model and of the description of the model. The model consists of the following items:

1. the list of stems, (2.18)
2. the lists of suffixes, (2.19)

3. the list of signatures.

4. the lists of pointers to the stems, the suffixes, and the signatures.

Let us consider a segmentation: the word types are split into stem-suffix pairs (suffix can be empty). The lists (2.18) and (2.19) of the model contain the stems and the suffixes of the segmentation.

The term signature has two meanings in the article. Here is how it is defined:

a. the "stem signature" is a set of all suffixes, with which the stem co-occurs in the stem-suffix pairs (similar to how I use the term "signature");

b. the "signature" is also the following pair of sets:

1. the stem signature (which is a set of suffixes);
2. the set of stems that share that stem signature.

In other words, a "signature" consists of a set of stems and a set of suffixes (or, rather, pointers to stems and suffixes from the lists (2.18) and (2.19)), where each stem co-occurs with each suffix.

From the signature definition it follows that one stem can belong to only one signature (which is not necessarily true for suffixes). Thus, a word in the corpus can have only one signature to

which its stem and its suffix belong, which phenomena I will refer to as “the signature produces the word”.

A word that is produced by an existing signature is referred to as “analyzed”, otherwise, it is called “unanalyzed”. An analyzed word has only a stem (as itself); it belongs to the list of stems in the model, but, again, there is no signature producing it.

It is convenient to have such a thing as unanalyzed words, because it allows to include in the model only signatures of a certain size or larger, drop all the other splits and consider those words unanalyzed (which is done in the actual algorithm; see below).

2.6.1.1. The compressed data length

The compressed length (in bits) of a word type *w* in the corpus *W* given a model *M* is the inverse $\log_2$ of its probability given the model:

$$L\,(w) = -\log_2 P(w|M), \tag{2.20}$$

which is, a basic principle of the information theory (Goldsmith, 2000 referring to Li and Vitánui, 1997).

Let the word *w* be analyzed, i.e. split into a stem suffix pair $w = s + e$, and $\exists\ signagure\ \sigma$: $(s, e) \in \sigma$. The probability of the word given the model is calculated like so:

$$P(w|M) = P(\sigma)P(s|\sigma)P(e|\sigma) \tag{2.21}$$

The probabilities of the signature, the stem, and the suffix in this formula are estimated as their "empirical frequencies" (Goldsmith, 2000):

$$P(w|M) = \frac{[\sigma]}{|W|}\frac{[s]}{[\sigma]}\frac{[e\ in\ \sigma]}{[\sigma]} = \frac{[s]}{|W|}\frac{[e\ in\ \sigma]}{[\sigma]}、 \tag{2.22}$$

where $[\sigma]$ is the number of tokens produced by σ, $[s]$ the number of tokens with the stem *s*, and $[e\ in\ \sigma]$ the number of tokens produced by σ with the suffix *e*.

Therefore,

$$L\ (w) = -\log_2 P(w|M) = \log_2 \frac{|W|}{[s]} + \log_2 \frac{[\sigma]}{[e\ in\ \sigma]} \tag{2.23}$$

If the *w* is unanalyzed then

$$P(w|M) = \frac{[w]}{|W|} \tag{2.24}$$

where $[w]$ is the number of occurrences of the word type *w* in *W*. Since an unanalyzed word is its own stem, *w* = *s* and $[w] = [s]$, and $P(w|M) = \frac{[s]}{|W|}$.

Even though there is no signature in the model that produces *w*, it can be said that *w* is produced by a trivial signature that contains *w* as the only stem and the empty suffix as the only suffix.

Hence $[\sigma] = [w]$ and $[e\ in\ \sigma] = [w]$, in which case $\log_2 \frac{[\sigma]}{[e\ in\ \sigma]} = 0$, and the formula (1) works for both analyzed and unanalyzed word.

The length of the corpus description is the sum of the lengths of all its word tokens:

$$L(Data) = \sum_{w=s+e}[w](\log_2 \frac{|W|}{[s]} + \log_2 \frac{[\sigma]}{[e\ in\ \sigma]}). \quad (2.25)$$

2.6.1.2. The model length

Recall that the model consists of a suffix list, a stem list, a signature list and lists of pointers to stems, suffixes, and signatures. The length of the model is the sum of the lengths of its components.

The affix list length is calculated as the length of the number of affixes in the list (N), which is $\log_2 N$, plus the sum of the lengths of the affixes. The affix length is calculated as the number of

its characters multiplied by $\log_2 |alphabet|$ (e.g. |*alphabet*| = 26 in the case of English). Thus, the lengths of the lists are:

$$L\,(Stems) = \log_2 |Stems| + \sum_{s \in Stems} \text{characters (s)} \log_2 |alphabet|\,, \tag{2.26}$$

$$L\,(Suffs) = \log_2 |Suffs| + \sum_{e \in Suffs} \text{characters (e)} \log_2 |alphabet|\,. \tag{2.27}$$

A signature contains a stem pointer list and a suffix pointer list. The length of an affix pointer is calculated like so:

$$L\big(ptr\,(s)\big) = -\log_2 \frac{[s]}{|W|} \tag{2.28}$$

for stems, and

$$L\big(ptr\,(e)\big) = -\log_2 \frac{[e\ in\ \sigma]}{[e]} \tag{2.29}$$

for suffixes. Thus, the length of a signature is

$$L(\sigma) = \log_2 |Stems\,(\sigma)| + \log_2 |Suffs\,(\sigma)| + \sum_{s \in Stems(\sigma)} \log_2 \frac{|W|}{[s]} + \sum_{e \in Suffs(\sigma)} \log_2 \frac{[\sigma]}{[e\ in\ \sigma]}\,, \tag{2.30}$$

and the length of the set of signatures (Σ) in the model is

$$L(\sigma) = \log_2|\Sigma| + \sum_{\sigma\in\Sigma} L(\sigma)\ . \qquad (2.31)$$

2.6.2. The algorithm

The algorithm used by Linguistica starts with the initial segmentation, modifies the segmentation iteratively checking whether it decreased the corpus compressed length, and goes on until there is no improvement.

In the initial segmentation every word is split at the peaks of the left segment successor frequency. The term "successor" here refers to a letter that follows the left segment of the split, which is the first letter of the right segment of the split. The word types in the corpus are split in all possible places, provided the left segment is 5 letters or longer. The distinct successors are counted for all the splits in the corpus. The split is accepted if the count has a clear peak.
The signatures are accepted only if they contain at least 5 stems and at least two suffixes.

On every iteration the segmentation is modified in the following ways:

1. a split of a word is added to the model if it consists of a stem and a suffix that are already known;

2. if a word form ends on a known suffix, the preceding segment is considered a stem candidate to be added to the model;
3. the stem-suffix boundary is shifted to the left, thus yielding a pair of new stem and suffix candidates.

Each of these changes may modify the model significantly: segments may be added to or removed from the segment lists, the signatures may be added and/or removed and so on. A modification is accepted if it reduces the compressed corpus length. If none of the possible changes is accepted the algorithm stops.

2.7. Morfessor (Creutz and Lagus, 2005)

Other systems I use to compare my results to are Morfessor Baseline and Morfessor Categories-MAP. Both systems take a corpus as input and segment its words into a set of morpheme-like segments referred to as "morphs", and Morfessor Categories-MAP labels the segments in addition (Morfessor Baseline is used by Morfessor Categories-MAP to obtain the initial segmentation). Unlike Linguistica (Goldsmith, 2000), the number of segments per word produced by the two systems can be (and often is) more than two.

2.7.1. Morfessor Baseline (Creutz and Lagus, 2005)

As mentioned above, Morfessor Baseline takes a corpus as input and segments its words into a set of morphs without labeling them.

The algorithm is based on the Maximum Aposteriori estimate (MAP). I will describe briefly the theory behind the algorithm leaving out the details.

2.7.1.1. The theory

The morphological model the algorithm is looking for is the one that has the highest probability given the corpus:

$$M^* = argmax_M P(M|corpus) = argmax_M P(corpus|M)P(M) \quad (2.32)$$

The model consists of the lexicon of morphs and a description of how the morphs can be combined, the grammar:

$$P(M) = P(L, grammar), \quad (2.33)$$

where $L = \{\mu_1, \dots, \mu_{|L|}\}$ is the morph lexicon.

The Morfessor Baseline model does not take into account any contextual information for morphs: it assumes that a morph is as likely to be used no matter what morphs precede or follow it. Thus, there is no grammar as such and the model probability is just the probability of the lexicon:

$$P(M) = P(L), \tag{2.34}$$

The probability of the lexicon is calculated as the probability of coming up with $|L|$ morphs:

$$P(L) = |L|!\, P\left(properties(\mu_1), \dots, properties\left(\mu_{|L|}\right)\right), \tag{2.35}$$

where the properties of an individual morph within the paradigm of this algorithm is nothing but its frequency and its form, a string of characters. Assuming independence of strings and frequencies

$$P\left(properties(\mu_1), \dots, properties\left(\mu_{|L|}\right)\right) = P\left(f_{\mu_1}, \dots, f_{\mu_{|L|}}\right) P\left(s_{\mu_1}, \dots, s_{\mu_{|L|}}\right). \tag{2.36}$$

To estimate probability distribution of the morph frequencies Mofessor Baseline uses the non-informative prior:

$$P\left(f_{\mu_1}, \dots, f_{\mu_{|L|}}\right) = \frac{1}{\binom{N-1}{|L|-1}} \tag{2.37}$$

where $N = \sum_{j=1}^{|L|} f_{\mu_j}$ (number of morph tokens in the corpus). There are versions of Morfessor based on Morfessor Baseline that use more explicit frequency distribution with user supplied parameters, but I will not describe those here because they are not used in the experiments.

It is also assumed that all the morphs are independent from each other:

$$P\left(s_{\mu_1}, \ldots, s_{\mu_{|L|}}\right) = \prod_{k=1}^{|L|} P\left(s_{\mu_k}\right) \tag{2.38}$$

and all the characters within the morph are also independent:

$$P\left(s_{\mu_k}\right) = \prod_{i=1}^{l_k} P(c_{ik}) \,, \tag{2.39}$$

where $s_{\mu_k} = c_{1k} \ldots c_{l_k}$, and $P(c_{ik})$ is the character probability distribution over the alphabet estimated by counting its frequency in the corpus.

The probability of a morph being of a particular length assumed to be exponentially distributed:

$$P(l) = \left(1 - P(\#)\right)^{l} P(\#) \tag{2.40}$$

where # is a special end-of-morph character.

With all the independence assumption mentioned above the probability of the corpus given the model is the product of probabilities of all the morph tokens:

$$P(corpus|M) = \prod_{j=1}^{W} \prod_{k=1}^{n_j} P\left(\mu_{jk}\right) , \tag{2.41}$$

where W is the number of tokens in the corpus and $P(\mu_i)$ is estimated by counting its frequency:

$$P(\mu_i) = \frac{f_{\mu_i}}{\sum_{j=1}^{|L|} f_{\mu_j}}. \quad (2.42)$$

2.7.1.2. The algorithm

Morfessor Baseline uses a greedy search algorithm. It starts with the morph lexicon being identical to the set of the word types in the corpus, each word being a morph of its own.

On each iteration the segmentation is modified a certain way and $P(M|corpus)$, or rather its logarithm, is calculated. The process stops when there is no significant improvement compared to the previous iteration.

The algorithm uses the following data structure:

1. Every word type is assigned a binary tree, which is referred to as a split tree; the word itself is the root of the tree. If the word is not split its split tree consists of just the root. Otherwise, the word is split in two; the segments are the children; each segment may also be split in two and so on. The leaves of the split tree are the morphs.

2. The data structure contains all the split trees such that the nodes are shared between the trees. Thus, each node is present in the structure only once; each non-leaf node has two children; any node can have any number of parents (see Figure 1).

3. Each node is associated with its frequency (occurrence count in the corpus). The frequency of each node is exactly the sum of frequencies of all its parents.

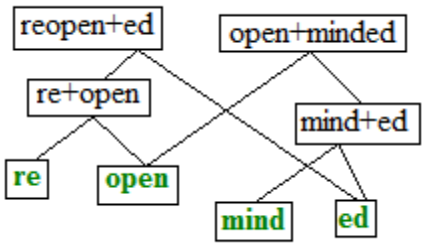

Figure 1. The Morfessor Baseline data structure containing the split trees of the words "reopen" and "openminded"

4. The set of leaves of this structure is the morph lexicon.

As it was mentioned before, initially the data structure is a set of trees that consist only of roots, which are also their leaves; the roots are the word types from the corpus.

On each iteration all the word types from the corpus are sorted randomly and each word is processed by the *resplitnode* procedure. It removes the words split tree from the structure (modifying the frequencies appropriately) and produces a new binary tree for this word that increases $P(M|corpus)$. It is done by first splitting the word in every possible way and calculating the split's contribution to the increase of $\log_2 P(M|corpus)$. The word may contribute best if it is left un-split, in which case *resplitnode* returns and the next word is fed to

it, otherwise the best split is chosen and both segments are processed by *resplitnode* recursively. The recursion stops when the node which is being processed is chosen to be left un-split.

The randomness of the order in which the words are processed in each iteration is important because of the greedy nature of the algorithm. The search is stopped when the increase of $\log_2 P(M|corpus)$ between the iterations is below a certain threshold, whose value is supplied by the user (the default is 0.005 times the number of word types in the corpus).

2.7.2. Morfessor Categories-MAP (Creutz and Lagus, 2005)

This system not only segments each word into a set of morphs but also labels them as members of the following categories: prefix (PRE) suffix (SUF) stem (STM).

An additional category is used in the algorithm: non-morpheme (NON). It is not, however, supposed to be in the system's output.

2.7.2.1. The theory

The theory is based on the maximum aposteriori estimate (MAP):

$$M^* = argmax_{lexicon} P(lexicon|corpus) = argmax_{lexicon} P(corpus|lexicon)P(lexicon) \quad (2.43)$$

is to be optimized, where lexicon is a set of morphs (types) the words are segmented into.

Each word type in the corpus is represented by the HHM with the four categories as states: STM, PRE, SUF, and NON.

Thus,

$$P(corpus|lexicon) = \prod_{j=1}^{W}(P(C_{j1}|C_{j0}) \prod_{k=1}^{n_j} P(\mu_{jk}|C_{jk})P(C_{j(k+1)}|C_{jk}))) \quad (2.44)$$

where W is number of tokens in the corpus, each token is split into n_j morphs (μ_{jk}); each μ_{jk} is assigned a category C_{jk}; $P(C_{j(k+1)}|C_{jk})$ are transition probabilities, $P(\mu_{jk}|C_{jk})$ are emission probabilities, $P(C_{j1}|C_{j0})$ is the transition probability of the transition from the special word boundary category to the first morph category. $P(C_{j(n_j+1)}|C_{jn_j})$ is the transition probability of the transition from the last morph category to the word boundary category.

Each morph is assumed to have a "meaning" and a "form"; the probability of the lexicon $L = \{\mu_1, \dots, \mu_{|L|}\}$ is calculated as follows:

$$P(L) = |L|! \prod_{k=1}^{|L|} P(meaning(\mu_k))P(form\ (\mu_k)) \quad (2.45)$$

It is assumed that a morph can be either a string of characters or a concatenation of two sub-morphs (which is referred to as "having a substructure"). The probability of a morph μ_k having its particular form is this:

$$P\big(form\ (\mu_k)\big) = \begin{cases} \big(1 - P(\sigma)\big) \prod_{j=1}^{len(\mu_k)} P\big(c_{kj}\big), & if\ \mu_k\ does\ not\ have\ substructure \\ P(\sigma)P(C_{k1}|\sigma)P(\mu_{k1}|C_{k1})P(C_{k2}|C_{k1})P(\mu_{k2}|C_{k2}), & otherwise \end{cases}$$

(2.46)

where $P(c_{kj})$ is the probability of the j^{th}character in the k^{th} morph (a special end-of-morph character is used to mark the morph end). The $P(c_{kj})$ are estimated by counting character frequencies in the corpus. $P(\sigma)$ is the probability that the morph has a substructure (again, it is estimated from morph lexicon by dividing the number of complex morphs by the total number of morphs). Finally, $P(C_{k1}|\sigma)$ is the probability that the first morph in substructure is of a given category C_{k1} (estimated by counting the occurrences as well).

The morph's "meaning" is assumed to be formed by the following features: morph frequency (indeed, frequent and infrequent morphs tend to have different semantics), morph length (e.g. longer morphs are more likely to be stems – stems often carry semantic rather than syntactic load, hence there should be many of them, hence they tend to be longer), intra-word right perplexity (it is harder to predict a morph following a prefix), and intra-word left perplexity (it is harder to predict a morph preceding a suffix).

The probability distribution of the morph frequency is assumed to be

$$P\left(f_{\mu_1}, \dots, f_{\mu_{|L|}}\right) = \frac{1}{\binom{N-1}{|L|-1}} \tag{2.47}$$

like in Morfessor Baseline.

The probability distribution of the morph length is deduced from the morph form.

The right perplexity is calculated like so:

$$rghtppl\ (\mu_i) = (\prod_{\nu_j \in right\ of \mu_i} P(\nu_j|\mu_i)\ )^{-\frac{1}{f_{\mu_i}}}, \tag{2.48}$$

where f_{μ_i} is the frequency of μ_i; $\nu_j$immediately follows μ_i ($P(\nu_j|\mu_i)$ is estimated by counting). The left perplexity is calculated analogously.

The perplexity probability distribution is estimated as

$$P(n) \approx 2^{-\log_2 c - \log_2 n - \log_2 \log_2 n - \log_2 \log_2 \log_2 n - \ldots} \tag{2.49}$$

where $c \approx 2.865$ (authors quote Rissanen (1989)).

The only thing lacking at this point is the emission probabilities. According to Bayes theorem,

$$P(\mu|C) = \frac{P(C|\mu)P(\mu)}{P(C)}, \tag{2.50}$$

where

$$P(C) = \sum_{i=1}^{|L|} P(C|\mu_i)P(\mu_i), \tag{2.51}$$

$P(\mu)$ and $P(\mu_i)$ are obtained via maximum likelihood estimate, and the $P(C|\mu)$ are estimated by introducing the $prefixlike,\ suffixlike, and\ stemlike$ properties of a morph:

$$prefixlike(\mu) = (1 + \exp\big(-a(rgtppl\ (\mu) - b)\big)^{-1}$$

$$suffixlike(\mu) = (1 + \exp\big(-a(lftppl\ (\mu) - b)\big)^{-1}$$

$$stemlike(\mu) = (1 + \exp\big(-c(length\ (\mu) - d)\big)^{-1} \qquad (2.52)$$

where b is perplexity threshold, d is a length threshold. Even though these properties are not probabilities (they do not sum up to 1) they can be used to calculate the probability of the NON category given a morph:

$$P(NON|\mu) = \big(1 - prefixlike(\mu)\big)\big(1 - suffixlike(\mu)\big)\big(1 - stemlike(\mu)\big) \qquad (2.53)$$

The probabilities of the remaining categories are proportional to the squares of the corresponding properties and sum up to $1 - P(NON|\mu)$.

2.7.2.2. The algorithm

Morfessor Categories-MAP uses a greedy search algorithm. The initial segmentation is obtained by using Morfessor Baseline (the authors elaborate on the importance of the initial segmentation for this algorithm taken into account its greedy nature). The initial tagging is done using the *stemlike, prefixlike, and suffixlike* properties (2.52).

The morphs are split, then they are joined using a bottom-up strategy, then they are split again, then re-segmentation and re-estimation of probabilities is done using Viterbi algorithm until convergence, then all these steps are repeated and the morph substructures are expanded to the resolution that does not have non-morphemes (morphs of NON category).

The morph splitting is done by, first, sorting them in the order of increasing length, then, trying every possible split. The best split is chosen and different category taggings are evaluated, for which every morph has to be re-evaluated, because changing one morph's tag changes the context for the other morphs thus affecting the transitional probabilities.

Not all the morphs are processed during the morph splitting. It gets interrupted from time to time (the authors do not specify when) and the whole corpus gets retagged using Viterbi algorithm; the probabilities, thus, are re-estimated and the splitting is resumed.

The bottom-up morph joining is done starting with the most frequent morph bigram and proceeding in the order of decreasing frequency. For every morph bigram there are two types of action that can be taken after the morphs are concatenated producing a new morph: 1. all three morphs are kept; the new morph is considered a morph that has a substructure, 2. the new morph is considered not having a substructure and the old morphs are discarded. Both actions are evaluated and the best is chosen. If neither contributes to the total probability increase the old morphs are kept separate. As in the case of the morph splitting, at times the process is interrupted and the corpus gets retagged using Viterbi algorithm, after which the morph joining is continued.

This alternating morph splitting and joining, as well as other things, are done in order to avoid the search getting stuck in the local maxima of the total probability function.

The reviewed approaches are quite interesting and produce good results. Goldsmith (2000), Creutz and Lagus (2005), Bernhard (2006), Demberg (2007) produce word segmentation only; while Schone and Jurafsky (2000 and 2001) and Moon et al. (2009) cluster words into conflation classes making use of various types of context: semantic, syntactic, or document boundaries. I attempted to design a simple system that takes a set of word types as an input and produces a set of conflation classes, thus, not relying on any context.

3. Methodology

The system takes a lexicon L, which is a set of word types, as input. The output is a set of possibly overlapping conflation classes (sets of morphologically related words); each of the conflation classes is a subset of L; the union of all the conflation classes equals to L.

The system processes the lexicon in four major phases:

1. Segmentation. Words are segmented into stem-suffix pairs;
2. The suffix types are clustered in a few different ways.
3. The best clustering that yields the best lexicon compression is assumed the optimum.
4. The conflation classes are built based on the optimum clustering.

It is necessary to introduce some terminology. The term "suffix signature" (similar to Goldsmith's (2000) "stem signature") is used in this thesis as "a set of all the stem types the suffix type co-occurs with". Correspondingly, the "stem signature" is a set of all the suffix types the stem type co-occurs with". The co-occurrence of a suffix and a stem types is considered "over a certain segmentation", where "segmentation" is the way the words of the lexicon are split. For instance, within the following segmentation

starling = star + ling

starling = starl + ing

starring = star + ring

boring = bor + ing

the suffix type "-ing" co-occurs with the stem types "starl-" and "bor-" but not "starr-", because the split "starring = starr + ing" is not a part of this particular segmentation. So, the signature of the suffix type "-ing" over the above segmentation is

Sig (-ing) = {starl-, bor-}

Correspondingly,

Sig (star-) = {-ring, -ling}.

By "suffix" and "stem" I will mean "suffix type" and "stem type". By "suffix frequency" or "stem frequency", as they are commonly referred to, I will mean the size of the suffix or the stem signature.

Phase 1. Segmentation (See Appendix A for a segmentation example).

The segmentation used here is rather simplistic but pretty intuitive and fast. The system attempts to find such segmentation where the split of each word maximizes the minimum of |*Sig (stem)*| and |*Sig (suffix)*|:

$$\forall w \in L,\ (s\ ,e) = \ arg\ \max_{sc+ec=w}\ min\ (|Sig(sc)|, |Sig(ec)|)\ , \tag{3.1}$$

where *sc* and *ec* for "stem candidate" and "suffix candidate" correspondingly.

Often there is more than one possible optimum split, in which case, the tie-breaker is used: the sum of suffix and stem frequencies. Let *G* be a set of all possible optimum splits:

$$G = \{(s,e): (s,e) = arg\ \max_{sc+ec=w}\ min(|Sig(sc)|,\ |Sig(ec)|)\ \} \tag{3.2}$$

then G_1 is a subset of G that contains the splits with maximum sum of their signature sizes (frequencies):

$$G_1 = \{(s,e): (s,e) = arg\ \max_{(sc,ec)\ \in G}(|Sig(sc)| +\ |Sig(ec)|)\} \tag{3.3}$$

and, finally, out of G_1 the split with the longest suffix is preferred:

$$(s^*, e^*) = argmax_{(sc,ec)\in G1}\ len\ (ec)\ . \tag{3.4}$$

(See section the section "Algorithm" for more details).

Phase 2. Suffix clustering (See Appendix B for a clustering example)

At this point all the words in the lexicon are split into stem-suffix pairs. Let *E* be a set of all the suffix types involved in the segmentation. Consider two of those suffixes and their signatures over this segmentation. Let those signatures share a certain number of stems. What share of features (the ratio of the number of the stems they have in common over the size of a signature) do they have in common? If the share of common features is significant at least for one of the signatures (and only in this case) these two suffixes are declared to be members of the same cluster:

$$\forall\{e_i, e_j\} \subset E, \frac{|Sig\ (e_i)\ \cap Sig\ (e_j)|}{min(|Sig\ (e_i)|\ , |Sig\ (e_j)|)}\ \geq t \in \mathbb{R} \iff \exists\ cluster : \{e_i, e_j\} \subset cluster \tag{3.5}$$

Where *t* is model parameter referred to as the clustering threshold.

For clustering no more than approximately 5,000 most frequent suffixes and 15,000 most frequent stems are used.

Phase 3. The lexicon compression. Finding the best value of *t.*

Consider a suffix cluster *cl.* For every stem *st* whose signature is a superset of *cl* and for every *sf* from *cl* any word *w = s + e* is in the lexicon:

$$\forall s,\quad \forall cl : cl \subseteq Sig(s) \quad \forall e \in cl \quad w = s + e \ \in L \tag{3.6}$$

because, by definition of the signature,

$$\forall e \in Sig\ (s),\ s + e\ \ \in L. \tag{3.7}$$

Let *scl* be a set of all stems whose signatures are supersets of *cl*:

$$scl := \{s : ecl\ \subseteq Sig(s)\} \tag{3.8}$$

then

$$L_{scl \times cl} = \{w : w = s + e,\ s\ \in scl, e\ \in cl\}\ \subseteq\ L. \tag{3.9}$$

And so, for every suffix cluster *cl* there is a corresponding stem set *scl* and their product $\{w = s + e,\ s\ \in scl, e\ \in cl\}$ is a subset of the lexicon. Thus the lexicon is the union of such products and some remainder:

$$L = \cup_i\ L_{scl_i \times cl_i}\ \cup\ \ R\ . \tag{3.10}$$

Now the lexicon `L` can be represented as the list of stems and suffixes involved in the segmentation, the list suffix clusters and corresponding stem clusters containing indices into the abovementioned lists of stems and suffixes, the list of their products each represented by a pair of indices into the stem and suffix clusters, and, finally, the remainder of the lexicon.

Thus, the clustering is done with different values of *t* and the size of the new representation of the lexicon is calculated for each *t*. The *t* that yields the smallest size is chosen.

The hypothesis is that the clustering that gives the best compression of the lexicon is the best for creation of the conflation classes.

Phase 4. The creation of the conflation classes (See Appendix C for an example of the conflation classes generation)

For any stem *st* and for every suffix cluster *cl* there is a conflation class:

$$\text{class}\ (s, cl) = \{w = s + e,\ e\ \in cl\ \cap \text{Sig}\ (s)\}\ . \tag{3.11}$$

Interestingly, the stem signatures can be taken over any segmentation, not necessarily the one obtained at the Phase 1. This may compensate for limitations produced by the "one split per word" constraint.

For experiments the conflation classes created by three different models:

1. The Tiny model: only the stems used for clustering (approximately the 15000 most frequent) are used for building conflation classes.

2. The Small model: All stems involved in the segmentation obtained at the Phase 1 are used.

3. The Large Model: all possible stems are used.

4. Algorithm and implementation

The system is implemented in Python 2.7. All the input, output and intermediary files are encoded as 'utf-8'.

4.1 Segmentation (See Appendix A. for a segmentation example)

Module: segm.py

input: lexicon file.
line format:

word rest_of_line

First word in each line is assumed to be is a word type of the lexicon; *rest_of_line* is ignored.

output:
file containing segmented words
name: segm.cnfl.
line format:

word stem "+" suffix

line example:

```
starting start + ing
```

The segmentation is implemented as an iterative process.

Step 1. The base segmentation S_0 contains all possible splits for each word of the lexicon, with a single constraint: a stem must only be 5 characters or longer. This constraint is based on the fact that short stems tend to be very ambiguous and, thus, unreliable as features. The base

segmentation provides the base signature sizes (frequencies) for all stems and all suffixes. S_0 is made the current segmentation S.

Step 2. A segmentation S_{new} is obtained using the stem and suffix frequencies from S. Each word is split in S_{new} maximizes

$$min\ (|Sig_S(stem)|, |Sig_S(suffix)|) \quad (4.1)$$

(the tie-breaker used is maximizing

$$|Sig_S(stem)| + |Sig_S(suffix)| \quad (4.2)$$

and, finally, taking the longest suffix).

Step 3. If the number of segments (stems and suffixes) involved in S_{new} is less than that of S S_{new} is made the current segmentation S and Step 2 is repeated, otherwise the segmentation S is declared final and written to the output file.

The Figure 2 contains a pseudo-code describing the algorithm for word segmentation. It uses the concept of a multiset, which is a set of elements not necessarily distinct. The "nde" stands for

```
Input: LEXICON is a set of words (types)
Output: SEGEMENTATIONS is a set of triples (word, stem, suffix),
        where word = stem + suffix

Algorithm:

STEMS = empty multiset
SUFFS = empty multiset

for each WORD in LEXICON
      for each (STM, SUF) such that STM + SUF = WORD and len(STM) > 4
            add STM to STEMS
            add SUF to SUFFS
      end for
end for

: start iterations

NEW_STEMS = empty multiset
NEW_SUFFS = empty multiset

SETMENTATIONS = empty set

for each WORD in LEXICON

      from all (STM, SUF) such that STM in STEMS and SUF in SUFS and
          STM + SUF = WORD

      find such (BEST_STM, BEST_SUF) that min (count(BEST_STM in
          STEMS), count(BEST_SUF in SUFFS)) is the largest

      add BEST_STM to NEW_STEMS
      add BEST_SUF to NEW_SUFFS
      add (WORD, BEST_STM, BEST_SUF) to SEGMENTATIONS

end for


if nde (NEW_STEMS) + nde (NEW_SUFFS) < nde (STEMS) + nde (SUFFS)

      STEMS = NEW_STEMS
      SUFFS = NEW_SUFFS

      go to "start iterations"

end if
```

Figure 2. The word segmentation. The "nde" stands for "number of distinct elements".

"number of distinct elements". In the system multiset is implemented as Python's dict type with distinct elements as keys and their counts as values.

4.2. Collecting suffix signatures. Discarding small signatures and non-frequent features.

Module: suff_signatures_trunc.py
input: segm.cnfl
output:
file containing suffix signatures
name: suff_signatures.cnfl
line format:

$$suff : |Sig(suff)| \quad :: \quad stem_1 : |Sig(stem_1)| \ldots stem_{|Sig(suff)|} : |Sig(stem_2)|$$

line example:

```
ing:3 :: start:4 read:5 writ:3
```

The signatures of all stems and suffixes are collected from the input file; the signatures are sorted in the order of descending size.

The size of the 15000-th stem signature is assigned to the minimum stem signature size, provided it is > 2, otherwise the minimum size is set to 2. The stem signatures whose sizes are lower than the minimum size are removed from the list.

The same is done to suffix signature list only the size 5000-th suffix signature is made the minimum size.

Then the stems whose signatures are not in the stem signature list are removed from the suffix signatures and vise-versa. The sizes of the signatures, thus, change.

The signatures whose became lower than the minimum size are removed again and the process is repeated until no signatures are to be removed.

4.3. Creating the adjacency matrix.

Module: matrix.proj.py
input: suff_signatures.cnfl
output:
file containing the adjacency matrix
name: matrix.cnfl
line format:

$suf_i * suf_1 = value_{i1}$ $suf_i * suf_2 = value_{i2}$ … $suf_i * suf_{i-1} = value_{i,i-1}$

example; top of the file:

```
s*#=0.997
ing*#=0.562 ing*s=0.466
ed*#=0.562  ed*s=0.467  ed*ing=0.956
es*#=0.281  es*s=0.017  es*ing=0.706 es*ed=0.714
```

The suffix signatures are read from the input file and the similarity measure is computed

$$m_{ij} = \frac{|\text{Sig}(e_i) \cap \text{Sig}(e_j)|}{\min(|\text{Sig}(e_i)|, |\text{Sig}(e_j)|)} \quad (4.3)$$

for every pair of suffixes (e_i, e_j).

The output file is a text file containing a lower triangle of the adjacency matrix, since the matrix is symmetrical and contains 1s on the main diagonal. The zeros are omitted.

4.4. Clique-clustering (see Appendix B. for a clustering example).

Modules: cluster1.py followed by cluster2.py
input: matrix.cnfl
clustering threshold $t \in (0,1]$
output:
file containing suffix clusters
name: clusters<*threshold_value*>.cnfl
file name example: clusters0.95.cnfl

line format:

$number\ size\ \ suf_1{:}n_1\ suf_2{:}n_2\ \ldots suf_{size}{:}n_{size}$

where n_i denotes the number of clusters containing suf_i (not used by the system).

line example:

```
10 6  fying:4 fy:4 fies:3 fied:2 fication:1 fier:1
```

The suffixes are considered vertices of a graph for which the adjacency matrix values determine whether any two vertices:

$$\left(e_i\ and\ e_j\ are\ connected\right) = \begin{cases} true, & m_{ij} \geq t \\ false, & m_{ij} < t \end{cases} \qquad (4.4)$$

All the cliques (sets of vertices where each is connected with all the rest) are extracted from the graph. Figure 3. contains a pseudo-code that describes (omitting certain details) the algorithm of extracting all cliques from a graph. It starts from GRAPH being a set of all vertices, from which the cliques are iteratively extracted and added to the set of cliques CLIQUES.

```
Input: GRAPH is a set of all vertices forming a graph
Output: CLIQUES is a set of all cliques from GRAPH

Algorithm:

SEEDS = GRAPH
CLIQUES = empty set

do while SEEDS in not empty

     SEED = arbitrary vertex from SEEDS
     CLIQUE = {SEED}


     for each vertex E in GRAPH \ {SEED}
           if E connected to all vertices in CLIQUE
                 add E to CLIQUE


     CLIQUE_CORE = empty set

     for each vertex C in CLIQUE
           if C not connected to any vertex in GRAPH \ CLIQUE
                 add C to CLIQUE_CORE


     GRAPH = GRAPH \ CLIQUE_CORE
     SEEDS = SEED \ CLIQUE

     add CLIQUE to CLIQUES

end do
```

Figure 3. The clique-clustering of a graph.

The extracted cliques are the sought clusters. They are written into the output file.

Note: the case of t = 0 should be a trivial case when all suffixes are elements of a single cluster. Since the adjacency matrix file does not contain zero values, however, the algorithm given t = 0 will cluster suffixes whose signatures have non-empty intersection (it will replace $m_{ij} \geq t$ with $m_{ij} > t$ in (4.4)).

4.5 Collect stem signatures and generate words.

Modules: generate.py
input:
clusters<*threshold_value*>.cnfl
suff_signatures.cnfl
output:
file containing words, their segmentations and the IDs of clusters that generated the words
name: generated<*threshold_value*>.cnfl
file name example: generated0.95.cnfl

line format:

$word :: stem + suffix \quad cluster\_numeric\_ID$

line example:

```
woodsman :: woods + man 481
```

The clusters are read in to form the cluster set:

$$CL = \{cl: |cl| \geq 2\}, \tag{4.5}$$

the clustered suffixes are collected:

$$E = \{e: \exists cl \in CL : e \in cl\}, \quad (4.6)$$

the stems are collected from the signatures of the clustered suffixes:

$$S = \{s : s \in Sig(e), \forall e: \exists cl \in CL : e \in cl\}, \quad (4.7)$$

the stem signatures are created:

$$\forall s \in S \; Sig(s) = \{e : s \in Sig(e)\}, \quad (4.8)$$

and for every stem the word form is generated as the concatenation of the stem and all suffixes from all clusters that are subsets of the stem's signature:

$$\forall s \in S, \forall cl \in CL : cl \subseteq Sig(s) \;\; \text{generate } w = s + e, \; \forall e \in cl \quad (4.9)$$

Each generated word together with the stem, the suffix, and the numeric ID of the suffix cluster the suffix belongs to is written to the output file (see "line example" above).

4.6. Evaluation: finding the compression ratio

Module: eval.py

input:
lexicon file
generated<*threshold_value*>.cnfl
output:
prints out the compression ratio to the console

The lexicon is read in and its 'size' is calculated by adding the lengths of all the words:

$$old\_size = \sum_{w \in L} |w| \tag{4.10}$$

The set of word forms with the stems, suffixes and cluster IDs are is read in:

$$F = \{(w, s, e, n) : s + e = w \wedge \exists cl_n : e \in cl_n\} \tag{4.11}$$

and the word set is formed:

$$W = \{w : \exists (w, s, e, n) \in F\} \tag{4.12}$$

as well as the sets of stems and suffixes involved:

$$ST = \{s : \exists (w, s, e, n) \in F\},\ SF = \{e : \exists (w, s, e, n) \in F\}, \tag{4.13}$$

and the set of suffix clusters

$$CL_{sf} = \{cl_{sf}(n) = \{e : \exists (w, s, e, n) \in F\}\} \tag{4.14}$$

and corresponding stem clusters

$$CL_{st} = \{cl_{st}(n) = \{s : \exists(w, s, e, n) \in F\}\} \quad (4.15)$$

Then the size of the "compressed file" is computed (without actually generating the file).

For any *n* that is a cluster number ($\forall n : \exists(w, s, e, n) \in F$) a set of word forms can be represented as a "product" of *n*-th stem class $cl_{st}(n)$ and *n*-th suffix class $cl_{sf}(n)$:

$$W(n) \coloneqq \{w : w = s + e,\ s \in cl_{st}(n),\ \ e \in cl_{sf}(n)\}, \quad (4.16)$$

and the set of all generated word forms *W* is the union of such sets:

$$W = \cup_n\ W(n), \quad (4.17)$$

Not all the words of the lexicon *L* are necessarily among the generated forms, therefore, the "compressed" files should contain those words that are not:

$$L = W\ \cup\ L \backslash W. \quad (4.18)$$

Thus, the "compressed file" will consist of the list of suffixes, the list of stems, the list of suffix and stem clusters (each cluster is a list of 2 byte pointers to the list of suffixes or stems), and the

list of words from the lexicon that were not among the generated word forms. And the size of the "compressed file" or the "new size" will be:

$$new\_size = \sum_{s \in S}|s| + \sum_{e \in E}|e| + \sum_{cl \in CL_{st}} 2|cl| + \sum_{cl \in CL_{sf}} 2|cl| + \sum_{w \in L \setminus W}|w| \quad (4.19)$$

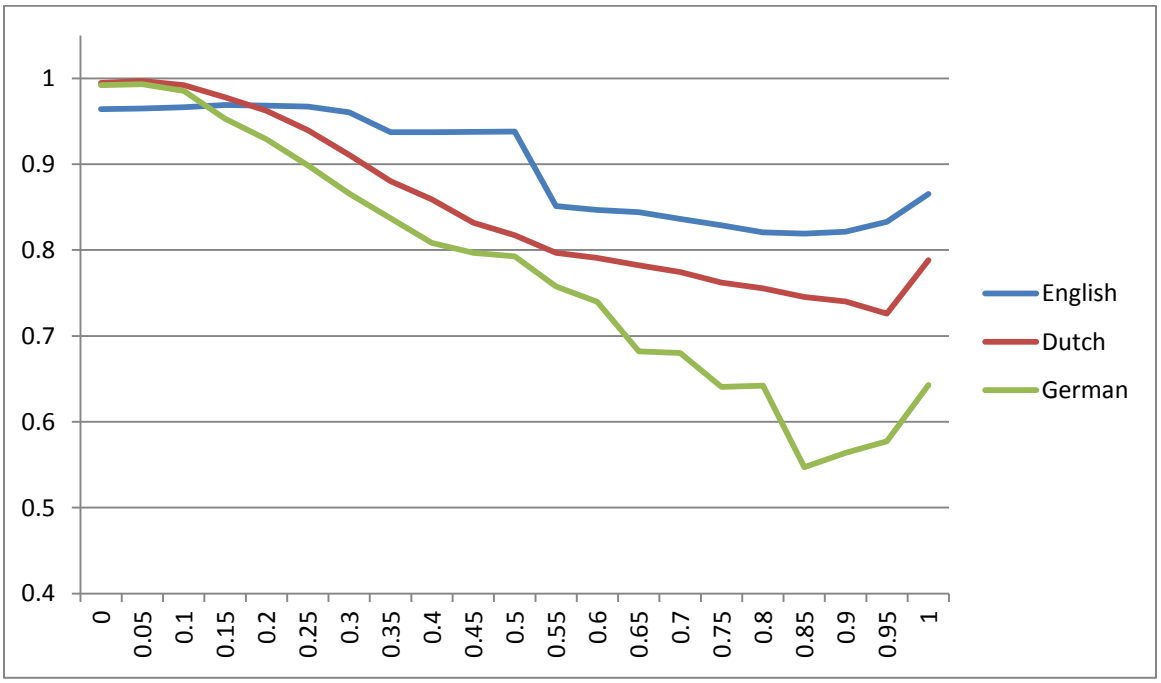

Figure 4. Compression Ratio for English, Dutch and German lexicons.

The output is the "compression ratio":

$$ratio = \frac{new\_size}{old\_size} \quad (4.20)$$

The Figure 4. shows the graphs of the compression ratio for different values of the clustering threshold t . The compression ratio values were computed for 21 values of t using the CELEX lexicons of English, Dutch, and German. This was done as a part of the experiments (see section Experiments). As we can see the compression ratios for all three languages have more or less clear global minimums.

The hypothesis is that the smallest *ratio* indicates the best suffix clustering, which, in turn, yields the best set of conflation classes.

4.7. Choosing the best clustering threshold

To get the best ratio the clustering (subsection 4.5) and evaluation (subsection 4.6) is done for all the values of $t \in (0,1]$ starting from 0.05 with the increment of 0.05. The suffix clusters that produced the lowest ratio are chosen for building the conflation classes, for which clusters<*best_t* >.cnfl is renamed to clusters.cnfl.

4.8. Generating stem signatures for the three models: the Tiny, the Small, and the Large (See Appendix C. for an example of conflation class generation using different models)

4.8.1. The tiny model

Module: gather_stems_tiny.py.

input:

cluster.cnfl
suff_signatures.cnfl

output: stem signature file

name: stems_tiny.cnfl)
line format:

$stem{:}\, len(stem) \;::\; suff_1\ suff_2 \ldots,$

line example:

```
lumin:5 :: aries ary osity ous ously
```

The suffix clusters are read in to form the cluster set:

$$CL = \{cl{:}\, |cl| \geq 2\}, \tag{4.21}$$

the clustered suffixes are collected:

$$E = \{e{:}\, \exists cl\ \in CL : e \in cl\}, \tag{4.22}$$

the stems are collected from the signatures of the clustered suffixes:

$$S = \{s : s \in Sig(e), \forall e{:}\, \exists cl\ \in CL : e \in cl\}, \tag{4.23}$$

the stem signatures are created:

$$\forall s \in S\ Sig(s) = \{e : s \in Sig(e)\}. \tag{4.24}$$

The stem signatures are sorted in the order of descending signature length and written to output file.

The Tiny model uses only the stems that are used as the suffix features for the suffix clustering. These are approximately 15000 the most frequent stems involved in the segmentation. Thus, only those words that were segmented into one of those stems and one of the clustered suffixes are grouped into the conflation classes. Each word from the rest of the lexicon is considered its own conflation class.

4.8.2. The Small model

Module: gather_stems_small.py.
input:
cluster.cnfl
segm.cnfl (output of the segmentation (subsection 4.1))

output: stem signature file
name: stems_small.cnfl)
line format:

$stem{:}\, len(stem)$ :: $suff_1\ suff_2 \ldots,$

line example:

```
lumin:5 :: aries ary osity ous ously
```

The Small model uses the stems that are involved in the segmentation phase (subsection 4.1).

As in the Tiny model the suffixes are collected from the suffix cluster file:

$$CL = \{cl: |cl| \geq 2\}, \tag{4.25}$$

$$E = \{e: \exists cl \ \in CL : e \in cl\} \tag{4.26}$$

The stems, however, are collected from the file created at the segmentation stage: segm.cnfl. This file has words with the splits assigned to them, e.g.:

```
luminaries lumin + aries
```

In other words, file contains the set of triples of this sort:

$$F = \{(w, s, e) : w = s + e\} \tag{4.27}$$

The stems that co-occur in F with the clustered suffixed are collected:

$$S = \{s: \exists (w, s, e) \in F : e \in E\}, \tag{4.28}$$

and their signatures are generated:

$$\forall s \in S, \ \ Sig(s) = \{e : \exists (w, s, e) \in F : e \in E\}, \tag{4.29}$$

and those of the size two or greater are written out in the output file in the same format as in the case of the Tiny model.

The Small model uses the stems involved in the segmentation that co-occur with the clustered suffixes. Correspondingly, only those words that are split into stem-suffix pair used by the model are grouped into the conflation classes. As for the rest of the lexicon, each word there forms its own conflation class.

4.8.3. The Large model

Module: gather_stems_large.py.

input:
cluster.cnfl
segm.cnfl (output of the segmentation (4.1))

output: stem signature file
name: stems_large.cnfl)
line format:

$stem{:}\, len(stem)$ :: $suff_1\ suff_2 \ldots$,

line example:

```
lumin:5 :: aries ary osity ous ously
```

The Large model uses the stems of all the words that end on a clustered suffix:

$$S = \{s{:}\, \exists w, \in L : w = s + e\ \wedge\ e \in E\}, \tag{4.30}$$

where L is the lexicon and E is the set of the clustered suffixes:

$$E = \{e{:}\, \exists cl\ \in CL : e \in cl\}, \tag{4.31}$$

$$CL = \{cl: |cl| \geq 2\}. \quad (4.32)$$

As in the Tiny and the Small models the clustered suffixes are collected from the suffix clusters file, while the lexicon is read from the segmentation file (segm.cnfl).

The idea behind the large model is that since we are using the clusters of the high frequency suffixes picked from the suffixes involved in the segmentation, using all possible stems might capture the morphological tendencies missed by the segmentation because of the "one split per word" constraint limitation.

4.9. Building conflation classes (See appendix C. for an example of conflation class generation)

Module: confl.py.
input:

cluster.cnfl
stems_tiny.cnfl, stems_small.cnfl, or stems_large.cnfl
the name of the output file

output: file containing conflation classes

name: class_tiny.cnfl, class_large.cnfl, class_small.cnfl depending on the model

File format: Each conflation class contains one word per line. The classes are separated by the blank lines. Each line also contains the stem used in building the class and the size of the class

Example (contains three conflation classes):

```
seventh stem:seven cluster:1441 len:2
sevenths stem:seven cluster:1441 len:2
```

```
seventeen stem:sevent cluster:132 len:4
seventeens stem:sevent cluster:132 len:4
seventeenth stem:sevent cluster:132 len:4
seventeenths stem:sevent cluster:132 len:4

seventy-eight stem:seventy-eight cluster:234 len:2
seventy-eights stem:seventy-eight cluster:234 len:2
```

The conflation classes are built for all the three models using different stem signature files as the input for each. It also takes as input the suffix cluster file. The conflation classes are generated the following way.

$$\forall s \, \forall cl \; class\,(s, cl) = \{w\colon \; w = s + e, \; e \; \in cl \; \cap \text{Sig}\,(s)\} \; . \qquad (4.33)$$

I.e., for every stem *s* and for every suffix cluster *cl* there is a conflation class if Sig *(s)* and *cl* have a non-empty intersection.

The Figure 5. contains a pseudo-code describing (in general terms) the generation of the conflation classes given a set of stems (STEMS), suffix clusters (SUFF_CLUSTERS) and stem

signatures. Note: the duplicate classes are discarded when newly generated conflation class (CONFL_CLASS) is added to the conflation class set (CONFL_CLASSES).

4.10. The subset removal

Module: confl_kill_subs.py.

input:

classes_tiny.cnfl, classes_large.cnfl, or classes_small.cnfl depending on the model

```
Input: STEMS is a set of stems,
       Sig(STEM) for each STEM in STEMS, and
       SUFF_CLUSTERS is a set of suffix clusters

Output: CONFL_CLASSES is a set of conflation classes

Algorithm:

CONFL_CLASSES = empty set

for each STEM in STEMS
     for each SUFF in Sig(STEM)

           CONFL_CLASS = empty set

           for each SUFF_CLUST in SUFF_CLUSTERS

                 if SUFF in SUFF_CLUST
                       WORD = STEM + SUFF
                       add WORD to CONFL_CLASS
                 end if


                 add CONFL_CLASS to CONFL_CLASSES

           end for
     end for
end for
```

Figure 5. The generation of the conflation classes.

output: file containing conflation classes with subsets removed. One class per line:
name: cl_tiny.cnfl, cl_small.cnfl and cl_large.cnfl depending on the model
Line format:

$class\_size$ :: $wd_1{:}n_1$ ... $wd_k{:}n_k$

where n_i is the number of classes that contain wd_i.

Example (two classes):

```
3 :: imitated:3 imitates:3 imitator:2
2 :: jihad:1 jihads:1
```

Often among the generated conflation classes some are subsets of others. These subsets are removed.

To make the subset removal algorithm efficient a stem trie is used, capitalizing on the fact that all the words in a conflation class share the same stem since any conflation class is generated by a stem and a suffix cluster:

$$class\ (stem, cluster)\ =\ \{wd: wd = stem + sf,\ sf\ \in cluster\ \cap \text{Sig}\ (stem)\}. \qquad (4.34)$$

Thus, each class has its stem and if one conflation class is a subset of another, the stem of one of the classes will be the beginning of the other's stem:

$$class(stem_1, cluster_1) \subseteq class(stem_2, cluster_2)$$

$$\Downarrow$$

$$\exists str : stem_1 = stem_2 + str\ \vee\ stem_2 =\ stem_1 + str \qquad (4.35)$$

where *str* is some string of characters.

This module (confl_kill_subs.py.) takes as input the output file produced by the previous step. Recall that each conflation class in that file also has its stem recorded.

```
Input: STEM_CLASS_PAIRS is a set of pairs (STEM, CONFL_CLASS) where all
       words in CONFL_CLASS share STEM

Output: NEW_STEM_CLASS_PAIRS is a set of pairs (STEM, CONFL_CLASS) where
        none of the clsasses is a subset of another class

Algorithm:

STEM_TRIE = empty trie

NEW_STEM_CLASS_PAIRS = empty set

for each (STEM, CONFL_CLASS) in STEM_CLASS_PAIRS:


      LINEAGE_STEM_SET = STEM_TRIE.get_ansestors (STEM) +
                                    STEM_TRIE.get_descendents (STEM)

      SUBSET = false

      for each ST in LINEAGE_STEM_SET
            for CONFL_CL such that (ST, CONFL_CL) in NEW_STEM_CLASS_PAIRS

                  if CONFL_CLASS subset CONFL_CL
                        SUBSET = true
                        break

                  else if CONFL_CL subset CONFL_CLASS
                        remove  (ST, CONFL_CL) from NEW_STEM_CLASS_PAIRS
                  end if

            end for

            if SUBSET
                  break
      end for

      if not SUBSET:
            add (STEM, CONFL_CLASS) to NEW_STEM_CLASS_PAIRS
            add STEM to STEM_TRIE
      end if

end for
```

Figure 6. The subset removal.

The Figure 6. has a pseudo-code describing the algorithm used for the subset removal. It takes the set of conflation classes and the corresponding stems

(STEM_CLASS_PAIRS) as given and produces the new set of classes and their stems as the result (NEW_STEM_CLASS_PAIRS).

4.11. Adding singletons

Module: add_singletons.py.
input:

name: cl_tiny.cnfl, cl_small.cnfl and cl_large.cnfl depending on the model

output: file containing conflation classes with subsets removed. One class per line:

name: cl_tiny.cnfl, cl_small.cnfl and cl_large.cnfl depending on the model

Line format: same as in output of the subsection 4.10.

Since every model groups only a subset of the lexicon into conflation classes the rest of the lexicon is added to the conflation classes files as a set of classes containing one word each.

5. Experiments

The set of classes containing morphologically related words was extracted of from the CELEX morphology database and was used as the gold standard. The lexicon for the experiment was collected from the gold standard data set.

The lexicons collected for different languages were of significantly different sizes:

English	73,396 words.
German	311,984 words.
Dutch	301,985 words.

The system was run on the three lexicons and the obtained sets of conflation classes were evaluated against the gold standard.

5.1. The evaluation

The evaluation was a bit of a problem. The evaluation metric used by Schone and Jurafsky (2000 and 2001) uses the concepts of correct, inserted and deleted words in the class set that is being evaluated against the gold standard class set.

Let L be the corpus, $\{X_1, X_2, \dots, X_n\}$ the class set to be evaluated and $\{Y_1, Y_2, \dots, Y_m\}$ the gold standard class set. The numbers of correct words (C), inserted words (I) and deleted words (D) are calculated according to the following:

$$C = \sum_{w \in L} \frac{|X_w \cap Y_w|}{|Y_w|}$$

$$I = \sum_{w \in L} \frac{|X_w| - |X_w \cap Y_w|}{|Y_w|}$$

$$D = \sum_{w \in L} \frac{|Y_w| - |X_w \cap Y_w|}{|Y_w|}, \quad (5.1)$$

where X_w and Y_w are classes that contain the word $w, \forall w \in L$.

Then precision (P), recall (R) and F-measure (F) are calculated as follows:

$$P = \frac{C}{C+I}, \; R = \frac{C}{C+D}, \; F = \frac{2PR}{P+R} \quad (5.2)$$

Unfortunately, this metric relies on every word being an element of only one class. In our situation this is not the case both for the class set to be evaluated and for the gold standard.

The metric suggested by Moon et al. (2009) is a generalization of the Schone and Jurafsky (2000) metric. It is designed to handle words belonging to more than one class:

$$C = \sum_{w \in L} \sum_{X_w} \sum_{Y_w} \frac{|X_w \cap Y_w|}{|Y_w|}$$

$$I = \sum_{w \in L} \sum_{X_w} \sum_{Y_w} \frac{|X_w| - |X_w \cap Y_w|}{|Y_w|}$$

$$D = \sum_{w \in L} \sum_{X_w} \sum_{Y_w} \frac{|Y_w| - |X_w \cap Y_w|}{|Y_w|} .$$

$$P = \frac{C}{C+I},\ R = \frac{C}{C+D},\ F = \frac{2PR}{P+R} \qquad (5.3)$$

Rather unfortunately, this metric does not satisfy the important basic rule: if a set of classes, where at least one word belongs to more than one class, is evaluated against itself the F-measure produced will be less than 100%. Indeed, let us consider an example of a three word lexicon {be, art, arts} divided into the two classes : {be, art} and {art, arts}. Let us evaluate this class set against itself.

$$X_1 = Y_1 = \{\text{be, art}\}$$
$$X_2 = Y_2 = \{\text{art, arts}\} \qquad (5.4)$$

Obviously, if I in (5.3) is greater than 0 the precision and, thus, the F-measure will be less than 100%.

According to (5.3)

$$I = I_{be} + I_{art} + I_{arts}$$
$$I \geq I_{art} = \sum_{X_{art}} \sum_{Y_{art}} \frac{|X_{art}| - |X_{art} \cap Y_{art}|}{|Y_{art}|} =$$
$$= \frac{|X_1| - |X_1 \cap Y_1|}{|Y_1|} + \frac{|X_1| - |X_1 \cap Y_2|}{|Y_2|} + \frac{|X_2| - |X_2 \cap Y_1|}{|Y_1|} + \frac{|X_2| - |X_2 \cap Y_2|}{|Y_2|} = 1 > 0 . \qquad (5.5)$$

If the gold standard class set evaluated against itself yields an F-measure that is less than 100% then what do the F-measures yielded by other class sets mean?

The problem with metric of Moon et al. (2009) is that in order to calculate C, I, and D for a word *w* it takes all possible classes pairs (X_w, Y_w) and then adds up the results, while, in my opinion, it has to take only pairs of the most similar classes. The question is how to establish what pairs are the most similar.

My suggestion would be to gauge the similarity of classes by using the cosine similarity measure:

$$\frac{|X_w \cap Y_w|}{|X_w||Y_w|} \tag{5.6}$$

Pairing up the classes can be done in many different ways. I propose the following relatively simple algorithm.

Let *w* be a word contained by more than one class both in the set to be evaluated and in the gold standard set:

$$w \in X_1, \dots, X_n$$
$$w \in Y_1, \dots, Y_k \tag{5.7}$$

The similarity measure between classes is:

$$m_{ij} = \frac{|X_i \cap Y_j|}{|X_i||Y_j|},\ 1 \leq i \leq n,\ 1 \leq j \leq k. \tag{5.8}$$

For convenience, let us introduce the two following sets of indices:

$$Ind(X) = \{1, \dots, n\},$$

and

$$Ind(Y) = \{1, \dots, k\} \tag{5.9}$$

The first most similar class pair would be such (X_{i_1}, Y_{j_1}), that

$$(i_1, j_1) = argmax_{i \in Ind(X),\ j \in Ind(Y)}\, m_{ij} \tag{5.10}$$

The classes X_{i_1}, Y_{j_1} are now out of the game and we look for the most similar pair among the remaining classes:

$$(X_{i_2}, Y_{j_2}),\ \ (i_2, j_2) = argmax_{i \in Ind(X)\setminus\{i_1\},\ j \in Ind(Y)\setminus\{j_1\}}\, m_{ij}. \tag{5.11}$$

Same for the third pair:

$$(X_{i_3}, Y_{j_3}),\ (i_3, j_3) = argmax_{i \in Ind(X)\setminus\{i_1, i_2\},\ j \in Ind(Y)\setminus\{j_1, j_2\}}\, m_{ij}\ , \tag{5.12}$$

and so on until we run out of classes on either side, at which point we will have obtained the set of class pairs:

$$best\_pairs(w) = \{(X_{i_1}, Y_{j_1}), \dots, (X_{i_l}, Y_{j_l}),\ \ l = \min(n, k)\}. \tag{5.13}$$

If $n \neq k$ there still remain unpaired classes among X_i or Y_j depending on whether $n > k$ or $n < k$. Suppose $n > k$, in which case $l = k$, and the set of unpaired classes is

$$U = \{X_i,\ i \in Ind(X)\backslash\{i_1, \dots, i_k\}\}, \tag{5.14}$$

(if $n < k$ the set of remaining classes would be $\{Y_j,\ j \in Ind(Y)\backslash\{j_1, \dots, j_n\}\}$, and the situation would be perfectly symmetrical).

Then the set of indices of the remaining classes is

$$Ind(U) = Ind(X)\backslash\{i_1, \dots, i_k\} = \{i_{k+1}, \dots, i_n\}. \tag{5.15}$$

For each i form $Ind(U)$ we can find $j(i)$ such that

$$j(i) = argmax_j m_{ij} \tag{5.16}$$

and pair X_i with $Y_{j(i)}$.

This gives us the set of pairs

$$rem\_pairs(w) = \{(X_{i_{k+1}}, Y_{j(i_{k+1})}), \dots, (X_{i_n}, Y_{j(i_n)})\}, \tag{5.17}$$

and, thus, all the classes are paired up.

Now, to calculate the C_w , I_w , and D_w we need to sum the numbers of correct, inserted and deleted words over the obtained class pairs:

$$C_w = \sum_{(X,Y)\in best\_pairs(w)} \frac{|X\cap Y|}{|Y|} + \sum_{(X,Y)\in rem\_pairs(w)} \frac{|X\cap Y|}{|Y|},$$

$$I_w = \sum_{(X,Y)\in best\_pairs(w)} \frac{|X|-|X\cap Y|}{|Y|} + \sum_{(X,Y)\in rem\_pairs(w)} \frac{|X|-|X\cap Y|}{|Y|},$$

$$D_w = \sum_{(X,Y)\in best\_pairs(w)} \frac{|Y|-|X\cap Y|}{|Y|} + \sum_{(X,Y)\in rem\_pairs(w)} \frac{|Y|-|X\cap Y|}{|Y|}, \quad (5.18)$$

And, finally,

$$C = \sum_w C_w$$

$$I = \sum_w I_w$$

$$D = \sum_w D_w$$

$$P = \frac{C}{C+I},\ R = \frac{C}{C+D},\ F = \frac{2PR}{P+R}\ . \quad (5.19)$$

5.2. Results

The results were compared to those yielded by the approach of Moon et al. (2009), Linguistica (Goldsmith, 2000), Morfessor Baseline (Creutz and Lagus, 2005) and Morfessor Categories-MAP (Creutz and Lagus, 2005).

My implementation of the approach of Moon et al. (2009) includes all the steps they use except for utilizing the document boundaries, since the input here is a lexicon only.

Moon et al. (2009) compare their model performance to that of Linguistica and Morfessor. I compare the performance of my system to that of Lingusitica and Morfessor the same way Moon et al. (2009) do it: for Linguistica, which provides segmentation only, the following clustering is used: the words sharing a stem belong to the same cluster and vice-versa. The same clustering method is used for the output of Morfessor.

Since Morfessor Baseline often breaks the words into more than two segments, the longest segment is considered a stem; Moon et al. (2009) do the same with the output of Morfessor Baseline for their evaluation.

Morfessor Categories-MAP labels the segments in addition. Thus, the stems are known by their labels. Again, the words sharing a stem form a cluster.

The results were also compared to the baseline set of classes, which represents a zero-analysis scenario: each word forms a separate class. The baseline classes, obviously, have a 100% precision.

For English the system produced just over 40,000 classes. Most of the classes look acceptable, with the members that are definitely morphologically related, e.g.

```
hospital hospitalization hospitalizations hospitalize hospitalized hospitalizes hospitalizing
```

or

```
penal penalization penalizations penalize penalized penalizes penalizing
```

Some classes are not as well formed. For instance, the very first four classes (the classes are sorted in the order of decreasing size) contain the hyphenated words sharing their first part. The following is the first class in the list:

```
well-advised well-balanced well-behaved well-chosen well-connected well-defined well-disposed well-earned well-established well-founded well-groomed well-grounded well-informed well-intentioned well-known well-meaning well-preserved well-read well-thought-of well-timed well-turned
```

These are definitely not morphologically related words. This is a reflection of a general problem the system has with the compound words. If we look at the clusters we see that all theses "advised", "balanced", "behaved", etc., are considered suffixes by the system. The system segmentation splits every word into a stem-suffix pair, which is generally wrong for compound words. The segmentation approach is based on an intuition that a word should be split where the stem and the suffix both have the largest frequencies (recall that $min(freq(stem), freq(suffix))$ is maximized in order to find the perfect split). If we look at the word "well-advised" in the English lexicon and count the frequencies, it will be obvious that it can only be split like so: "well-" + "advised". Indeed, the frequency of "#well-" is 50 and the frequency of

"advised#" is 6; any other split would produce a much smaller $min(freq(stem),\ freq(suffix))$. (6.1)

Another problem is that this system does not deal with prefixes. So the pair of morphologically related words like "compress" and "decompress" will never be put into the same class. They are, however, in the same class in the gold standard set of classes.

Yet another problem is the noise produced by the low coverage of the lexicon. The system, for example, created such classes:

```
subjection subjectively
```

and

```
subjection subjective
```

All these four words are morphologically related, yet, the system placed them in two different classes because there was not enough evidence in the lexicon that "subjective" and "subjectively" were related.

5.3. Discussion

The system showed better results than the other systems in all but one experiment. The comparison, however, is not entirely fair: Linguistica and Morfessor were not designed for word clustering, while Moon et al. (2009), requires document boundaries for the best performance.

The evaluation results of the Moon et al. (2009) algorithm is reflected in the 'Moon et al. (2009)' row of the Table 1. The 'Moon et al. (2009) purged' row reflects the evaluation results of the Moon et al. (2009) classes after the subset removal (which is not part of their algorithm), i.e. all the classes that are subsets of other classes were removed. Evidently, the subset removal improves the results significantly; 'Moon et al. (2009) purged' does better for English than my system.

	English			German			Dutch		
	P	R	F	P	R	F	P	R	F
Baseline	1	0.31	0.47	1	0.1	0.19	1	0.31	0.47
Tiny Model	0.9	0.49	0.64	0.94	0.35	0.51	0.97	0.42	0.59
Small Model	0.9	0.49	0.64	0.94	0.36	0.52	0.97	0.44	0.6
Large Model	0.87	0.57	0.69	0.92	0.46	0.61	0.95	0.53	0.68
Moon et al. (2009)	0.88	0.51	0.64	0.9	0.29	0.44	0.79	0.43	0.56
Moon et al. (2009) purged	0.88	0.62	0.73	0.8	0.48	0.6	0.77	0.6	0.68
Linguistica	0.68	0.58	0.63	0.6	0.38	0.47	0.28	0.49	0.36
Morfessor baseline	0.2	0.79	0.32	0.18	0.55	0.27	0.05	0.69	0.09
Morfessor Categories-MAP	0.14	0.87	0.24	0.07	0.72	0.13	0.03	0.79	0.05

Table 1. Results of the experiments: Precision, Recall and F-measure values yielded by different systems for English, German and Dutch. My system's results are represented by the "Tiny Model", "Small Model", and "Large Model" rows.

As expected, the Large model worked better than the Small, which, in turn, showed better results than the Tiny model (except for English where the Tiny and the Small mode sets of classes are the same due to the smallness of the lexicon). The Large model was supposed to compensate for the "one split per word" constraint limitation, which was likely to limit the recall. Indeed, the larger set of stems seems to drop the precision somewhat but increase the recall enough to increase the F-measure.

Apparently, both Morfessor and Linguistica, as a general rule, yield considerably higher recall and lower precision than my system and that of Moon et al. (2009). This, together with their lower F-measure due to the precision being too low, is a clear sign of "over-joining" of the conflation classes.

For instance, Morfessor Categories-MAP mistakenly (probably due to the lexicon's low coverage) takes "mon" as a stem in a number of words, resulting in "money", "demon" and all their variants (like "demonic", "moneylender", "monetize", etc.) being in the same conflation class.

Linguistica does show a higher precision since it produces only one stem per word. Yet it also makes segmentation errors, e.g., it puts a stem "st" in words "st", "stable", "state", "stew", "std", "stern" and many others, all of which end up in one conflation class.

All of that is hardly surprising: as it was mentioned above neither Morfessor nor Linguistica are built to cluster words – they only do segmentation. The way Moon et al. (2009) and I use these systems for word clustering (all the words sharing a stem are members of the same conflation class) makes the price of their segmentation/labeling mistakes too high, while both my system's suffix clustering phase and that of Moon et al. (2009) act as major statistical filters for segmentation errors.

The hypothesis that the suffix clustering yielding the best compression also yields the best conflation class set turned out to be entirely wrong. I built the Tiny model conflation classes for all the values of the clustering threshold and realized that not only the highest F-measure was produced by a different value of the threshold than the one that produces the lowest compression ratio, but that the compression ratio behavior does not in the least correspond to that of the F-measure. It is very visible in the Figure 7, which shows the graphs of the precision (P), recall (R) and F-measure (F) yielded by different threshold values compared against the compression ratio graph "flipped" vertically (the purple graph reflects (1.4 – compression_ratio) for better pattern visibility).

I also ran the Large model for English for all the values of *t*, to see whether the Large model F-measure behaves in a similar way to that of the Tiny model. The Figure 8. shows the precision, the recall and the F-measure of the Large model for English.

It is visible that the Large model graph and Tiny model graph have certain similarities: the precision is increasing monotonically, the recall is monotonically decreasing, while the F-measure shows a tendency of growing (slightly, in the case of the Tiny model) and then falling with increasing speed as *t* comes closer to 1.

I also ran the same experiments on Dutch and German lexicons. Unfortunately, since those lexicons are much larger than the English, running the Large model for all the values of *t*, especially for the ones lower than 0.5 would be too time consuming. The Figures 9 and 10 show the Tiny model result graphs and the "flipped" compression ratio.

Again, it is visible that the compression ratio behavior does not correspond to the behavior of the F-measure. The similarities in the system performance for the different languages are apparent: the precision increases more or less monotonically while the recall is going down as the clustering threshold increases.

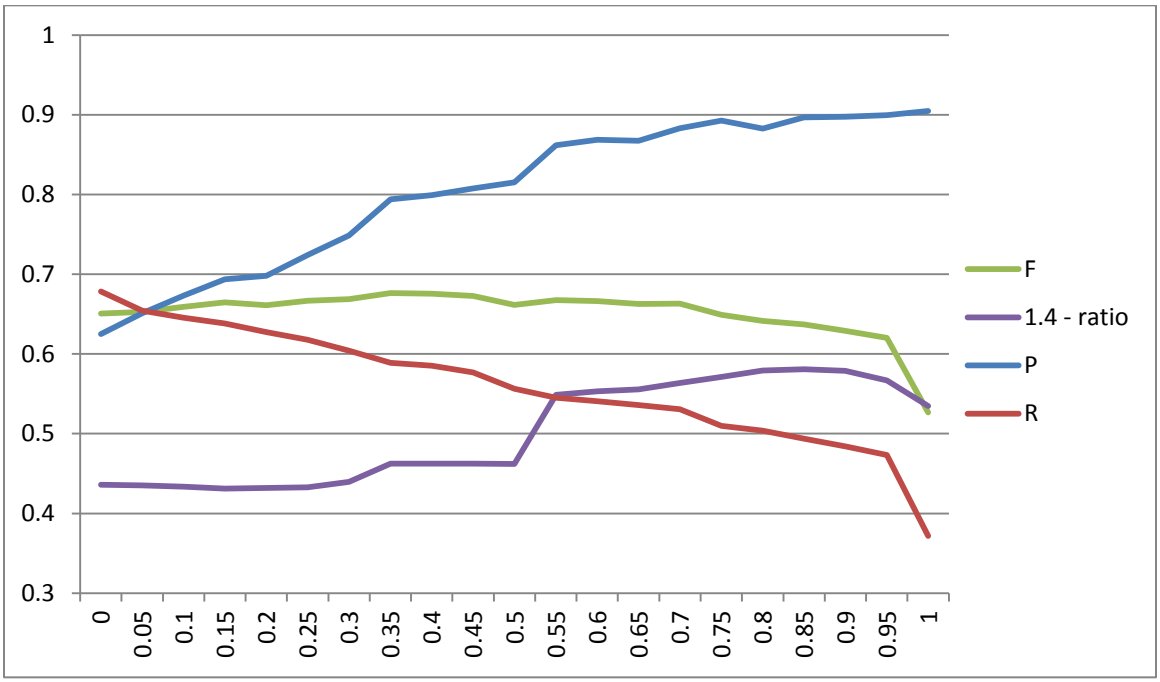

Figure 7. The Tiny Model English classes Precision, Recall, F-measure, and Compression Ratio ("flipped and lifted").

Thus, there are no criteria so far for choosing the optimum clustering threshold value, since in real life situation there is no gold standard. This issue has to be researched more in the future. As we can see from the figures, with a better criteria for choosing the clustering threshold the results produced by the system would have been higher.

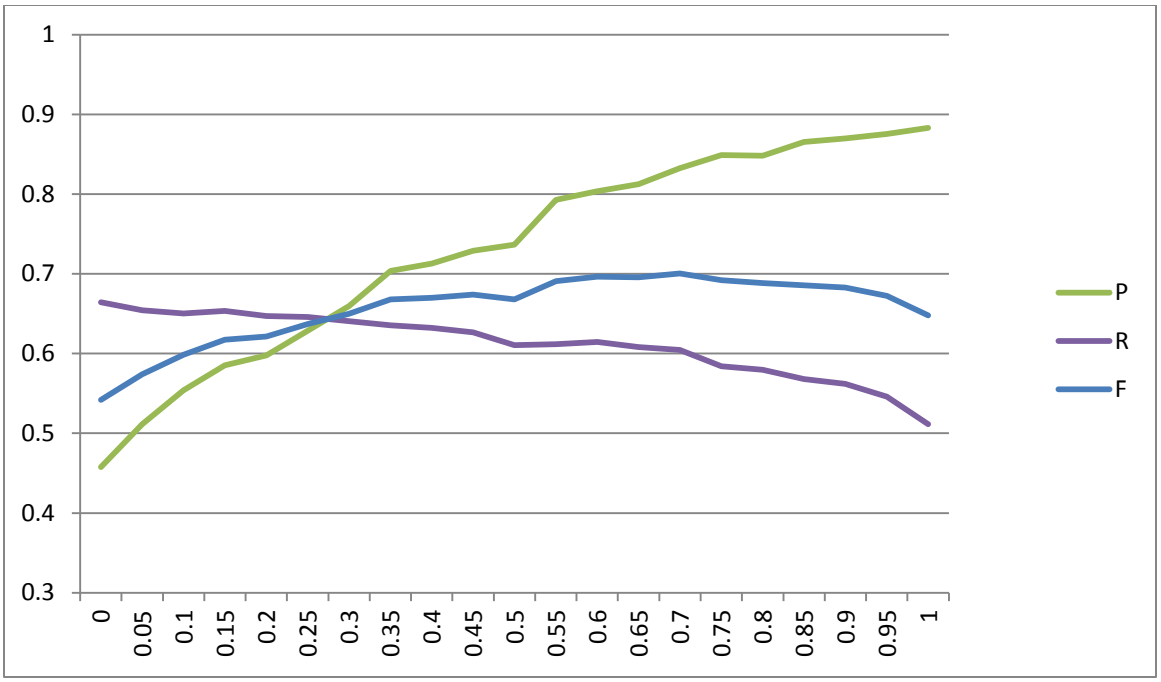

Figure 8. The Large Model English classes Precision, Recall and F-measure.

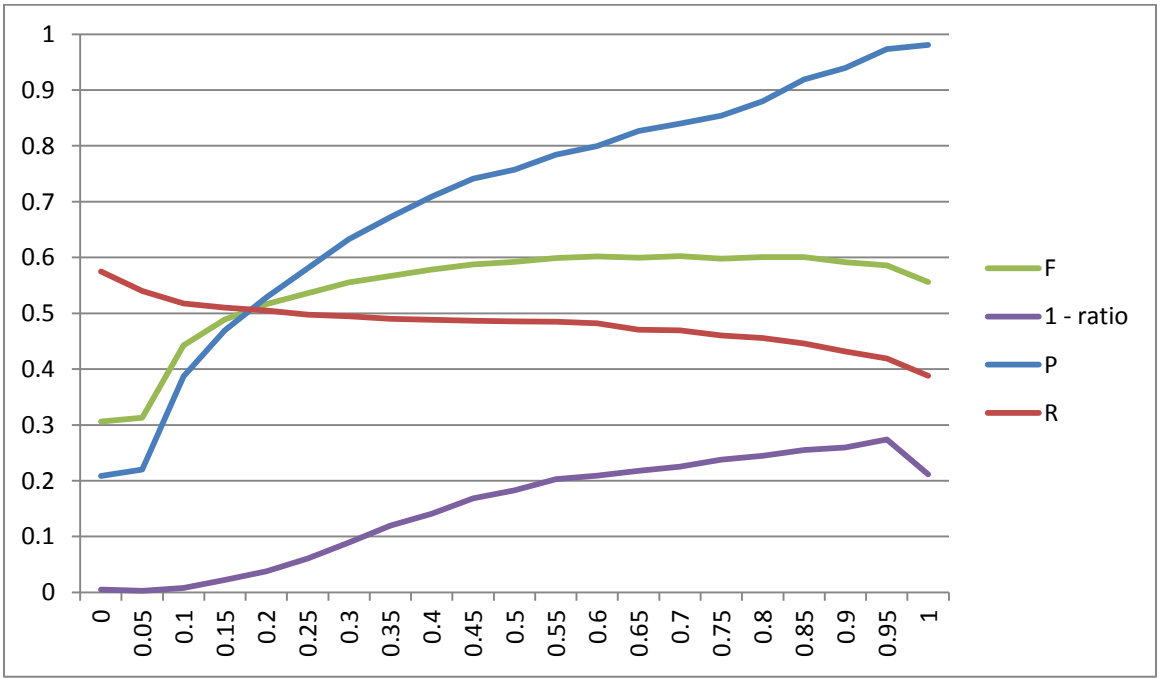

Figure 9. The Tiny Model Dutch classes Precision, Recall, F-measure, and "flipped" Compression Ratio.

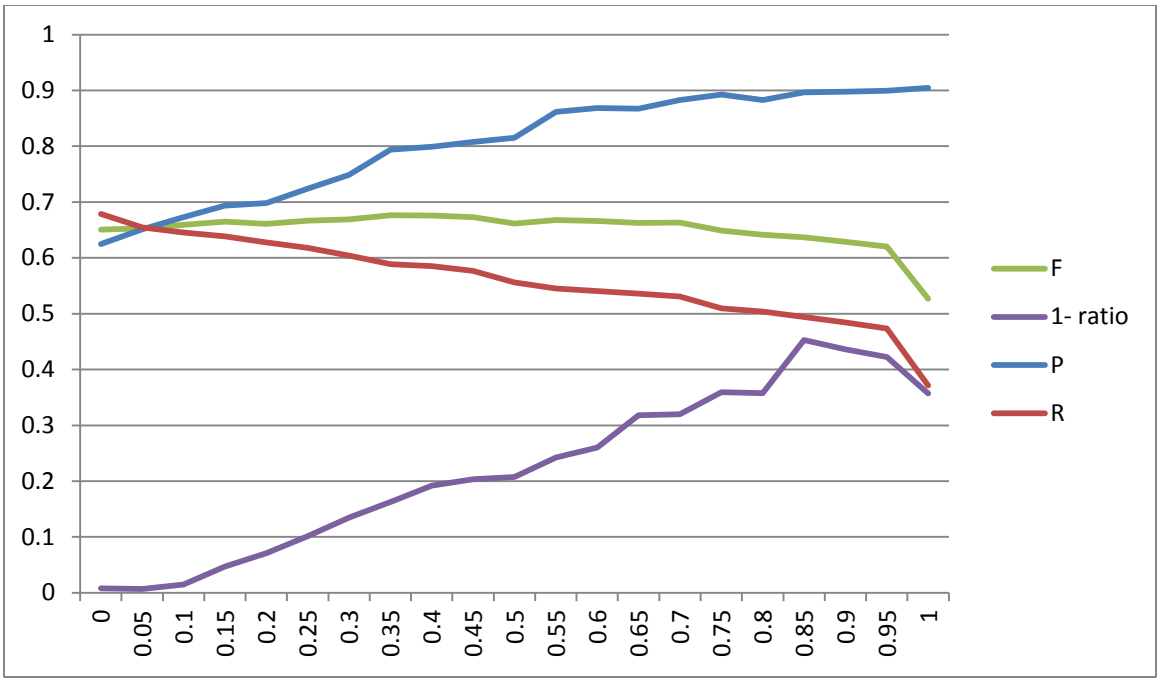

Figure 10. The Tiny Model German classes Precision, Recall, F-measure, and "flipped" Compression Ratio.

6. Conclusion and future work

We considered a system that uses clustering to create a set of conflation classes given a lexicon in a written language. It was tested on Dutch, English, and German CELEX lexicons and the resulting conflation classes were evaluated against those derived from the CELEX morphological database (Baayen, et al., 1993).

The same experiments were done with the use of Linguistica (Goldsmith, 2000), Morfessor, both Baseline and Categories-MAP (Creutz and Lagus, 2005), and using the approach of Moon et al. (2009). The results were compared with those of the present system. The system results were higher in all but one experiment: Moon et al. (2009) with the subsets removed showed a better result for English. It should be noted, however, that the subset removal, even though a sensible thing to do, is not part of the algorithm of Moon et al. (2009), at least it was not specified in their article.

Some unresolved issues, however, remain. The proposed way to choose the value of the clustering threshold – a crucial model parameter used by the system – proved to be completely wrong. This is one of the most important issues to be resolved in the future.

Another future item is to try this algorithm on different corpora while using the same gold standard. Evaluating, in this case, is to be done for the intersection of the lexicons of the corpus and the CELEX database. That is the way it is done by Schone and Jurafsky (2000 and 2001) and by Moon et al. (2009).

The presented system also does not deal with prefixes, which is a serious shortcoming. Gathering stem and suffix candidates should be done after the words of the lexicon are stripped of prefixes, as it is done by Schone and Jurafsky (2000). Additionally, considering circumvention (Schone and Jurafsky, 2001) seems to have a lot of potential.

Additionally, the system does poorly with the compound words. For example, the words “well-advised”, “better-advised”, and “best-advised” should be in the same conflation class, while the system assumes that words within the same conflation class must share a stem and puts “well-advised” together with “well-balanced”. It must be said, however, that in this case, the decision would be based on the semantic information, which is not available when dealing just with the lexicon.

Finally, since the approach is based solely on intuition, developing a theory that would back it up is something that definitely needs to be done.

Appendix A. An example of word segmentation

I will illustrate how the word segmentation works using this small lexicon:

```
thinking                                                    (A.1)
thinks
thinker
think
bringing
bring
brings
bringer
staring
stare
stared
stares
drinked
drinker
drinking
drinks
sink
sunk
sinks
sinker
```

The first step is to split the words in all possible places and count the segment frequencies. Note that even though in this example I disallow only empty stems, in real experiments disallowed are all the stems shorter than 5 letters.

It turns out that the initial segmentation contains 55 stem and 56 suffix types (further referred to as "stems" and "suffixes") including the empty suffix ('#'). I am not showing them here because the lists are too long and do not carry much information.

For each word out of all the possible splits we choose the one that maximizes the minimum of suffix and stem frequencies. The Table 2 shows how the word "thinking" is segmented.

Stem	stem frequency (fstm)	suffix	suffix frequency (fsuf)	min (fsuf, fstm)	winner (max)
t	4	hinking	1	1	
th	4	inking	2	2	
thi	4	nking	2	2	
thin	4	king	2	2	
think	4	ing	5	**4**	4
thinki	1	ng	5	1	
thinkin	1	g	5	1	
thinking	1	#	20	1	

Table 2. The segmentation of the word "thinking". `"think" = "think" + "ing"` is the best split

As we can see the split `"think" = "think" + "ing"` clearly wins in this case.

The Table 3 contains an example in which the tie breaker has to be used. Recall that the tie breaker is the sum of the suffix and the stem frequencies. In this example there are four splits that have the same winning *min (fsuf, fstm)*. Thus, split with the largest sum of frequencies is the winner: `"sinks" = "s" + "inks"`.

Stem	stem frequency (fstm)	suffix	suffix frequency (fsuf)	min (fsuf, fstm)	winners (max)	Tie breaker (fsuf + fstm)	winner (max)
s	8	inks	3	**3**	3	**11**	11
si	3	nks	3	**3**	3	6	
sin	3	ks	3	**3**	3	6	
sink	3	s	5	**3**	3	8	
sinks	1	#	20	1			

Table 3. The segmentation of the word "sinks". The tie breaker is used (fsuf +fstem).

Note, that if the tie breaker still produces a tie, the split with the longer suffix is preferred.

After all the words have been split in such a way, we obtain the following segmentation:

```
thinking = think + ing                                          (A.2)
thinks = think + s
thinker = think + er
think = think + #
bringing = bring + ing
bring = bring + #
brings = bring + s
bringer = bring + er
staring = star + ing
stare = stare + #
stared = stared + #
stares = stare + s
drunk = dr + unk
drinker = dr + inker
drinking = drink + ing
drinks = drink + s
sink = sink + #
sunk = s + unk
sinks = s + inks
sinker = s + inker
```

Now the system collects the segments from the segmentation and counts their frequencies. Apparently, this segmentation contains 9 stems and 7 suffixes. Here they are (every segment is followed by its frequency after the ":"):

```
stems

bring:4, dr:2, drink:2, s:3, sink:1, star:1, stare:2, stared:1, think:4


suffixes

#:5, er:2, ing:4, inker:2, inks:1, s:4, unk:2
```

As the next step, the system re-splits the words according to the new frequencies using only the segments from (A.2). There will be no new stems or suffixes, but some of the existing ones might disappear. The new segmentation is:

```
thinking = think + ing                                                    (A.3)
thinks = think + s
thinker = think + er
think = think + #
bringing = bring + ing
bring = bring + #
brings = bring + s
bringer = bring + er
staring = star + ing
stare = stare + #
stared = stared + #
stares = stare + s
drunk = dr + unk
drinker = dr + inker
drinking = drink + ing
drinks = drink + s
sink = sink + #
sunk = s + unk
sinks = sink + s
sinker = s + inker
```

and the new segments (and their frequencies) are:

```
stems

bring:4, dr:2, drink:2, s:2, sink:2, star:1, stare:2, stared:1, think:4,


suffixes

#:5, er:2, ing:4, inker:2, s:5, unk:2,
```

There are 9 stems and 6 suffixes now – one suffix ("`inks`") from the previous segmentation (A.2) was dropped.

The system will keep re-splitting the words and re-counting frequencies until the number of segments in the segmentation does not change. In this example the above segmentation happens to be the final one.

Appendix B. A clustering example

This example will be using the segmentation (A.3) obtained in the Appendix A.

Let us gather the segmentation suffix signatures (sets of segmentation stems the suffixes co-occur with)

```
Sig(#) = {bring, think, stare, sink, stared}
Sig(s) =  {bring, think, stare, drink, sink}
Sig(ing) = {bring, think, drink, star}
Sig(unk) =  {s, dr}
Sig(inker) = {s, dr}
Sig(er:2) =  {bring, think}
```

(B.1)

Now calculate the similarity between suffixes. Recall that the similarity measure is the following:

$$\mu(e_i, e_j) = \frac{|Sig\ (e_i) \cap Sig\ (e_j)|}{min\ (|Sig\ (e_i)|, |Sig\ (e_j)|)} \quad \text{(B.2)}$$

	#	inker	er	unk	ing
inker	0				
er	1	0			
unk	0	1	0		
ing	0.5	0	0	0	
s	0.8	0	1	0	0.75

Table 4. The similarity measures between suffixes. The similarity measure is commutative, hence the empty cells.

The Table 4. shows the similarity measure values for all the suffix pairs.

The clustering is done according to the principle that two suffixes $\mu(e_1, e_2)$ are members of one cluster if and only if $\mu(e_1, e_2) \geq t$, where *t* is a clustering parameter a.k.a. clustering threshold. Here are the only 4 possible different clusterings for different values of *t*:

```
t = 0.5  {#, ing, s, er}, {unk, inker}                                    (B.3)
t = 0.75 {#, s, er}, {ing, s, er}, {unk, inker}
t = 0.8  {#, s, er}, {ing, er},{s, er}, {unk, inker}
t = 1.0  {#, er}, {s, er}, {ing, er},{s, er}, {unk, inker}
```

Appendix C. An example of conflation class generation

We will use the clusters obtained in the clustering example in the Appendix B (B.3). We will omit here the file compression phase done by the system to obtain the best clustering threshold. Instead, we will just choose *t = 0.5* for simplicity. Thus, we have the following set of clusters:

```
t = 0.5  {#, ing, s, er}, {unk, inker}
```

The thing to do at this point is to gather the stem signatures. There are two sources of stem signatures that are used: the suffix signatures gathered from the segmentation in the example in the Appendix B (B.1) and the lexicon (A.1). The Small and Tiny models (which, in this case, are the same due to the smallness of the lexicon) use the former, while the latter is used by the Large model. The stem signature source is the only difference between the models.

Let us consider the Small model first. The stem signature is a set of all suffixes whose signatures contain this stem. Note: for the conflation class generation we need the stem signatures that contain at least two suffixes. The stem signatures, thus, are:

```
Sig(bring) = {#, er, ing, s}
Sig(think) = {#, er, ing, s}
Sig(stare) = {#, s}
Sig(drink) = {ing, s}
Sig(s) =  {inker, unk}
Sig(sink) = {#, s}
Sig(dr) = {inker, unk}
```

The conflation classes are generated by taking intersections of every stem signature and every suffix cluster:

```
{bring bringer bringing brings}
```

```
{think thinker thinking thinks}
{stare stares}
{drinking drinks}
{sinker sunk}
{sink sinks}
{drinker drunk}
```

At the end the left out lexicon words are added to the list as the one-word classes:

```
{stared}
{staring}
```

The Large model gathers the stem signatures from the lexicon, while suffixes are taken from the suffix clusters. If a lexicon word ends with a suffix from the suffix clusters, the remainder of the word forms a stem, whether it is in the suffix signatures or not, and the suffix now belongs to the stems signature. For instance, the word `drinker` ends with `er` which is a suffix from a cluster:

```
drinker = drink + er
```

Thus, `drink` is a Large model stem and its signature contains `er`, even though `drink` is not in the signature of the suffix `er`.

The Large model attempts to compensate the "one split per word" constraint limitation. Indeed, while the final segmentation (A.3) for the word `drinker` happens to be

```
drinker = dr + inker
```

the Large model adds another one to the game:

```
drinker = drink + er
```

The list of the Large model stem signatures is:

```
Sig(bring) = {#, er, ing, s}
Sig(think) = {#, er, ing, s}
Sig(drink) = {er, ing, s}
Sig(sink) = {#, er, s}
Sig(stare) = {#, s}
Sig(dr) = {inker, unk}
Sig(s) = {inker, unk}
```

and the conflation classes are:

```
{bring, bringer, bringing, brings}
{think, thinker, thinking, thinks}
{drinker, drinking, drinks}
{sink, sinker, sinks}
{stare, stares}
{sinker, sunk}
{drinker, drunk}

##### added singetons #############
{stared}
{staring}
```

Even though the conflation classes obtained here have some resemblance to the real conflation classes, they definitely do not match them completely. This can be explained by the smallness of the lexicon, which is the only source of information for the algorithm. For example, the words `stare, stared,` and `staring` are put in the different conflation classes simply because there is not enough evidence in the lexicon that they are related. Another note would be that in this case the Small and Large model conflation class sets do not differ all that much. In cases of large lexicons the difference between the models is usually quite significant.

References

Bernhard, D., 2006. *Unsupervised Morphological Segmentation Based on Segment Predictability and Word Segments Alignment.*

Chapin, P., & Norton, L.1968. A Procedure for morphological analysis. *Information system language studies number eighteen*, MTP-101. Massachusetts. (The Mitre Corporation)

Creutz, M.& Lagus, K. 2005. Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0. *Publications in Computer and Information Science, Report A81*, Helsinki University of Technology, March.

Creutz, M.& Lagus, K. 2005. Inducing the Morphological Lexicon of a Natural Language from Unannotated Text. *In Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, Espoo, Finland, 15-17 June.

Demberg, V., 2007. *A Language-Independent Unsupervised Model for Morphological Segmentation.* Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 920–927, Prague, Czech Republic, June 2007. c 2007 ACL.

Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, Vol. 41, pages.391-407.

Goldsmith, J. 1997/2000. Unsupervised learning of the morphology of a natural language. *Univ. of Chicago.http://humanities.uchicago.edu/faculty/goldsmith*.

Kazakov, D. 1997. Unsupervised learning of naïve morphology with genetic algorithms. In W. Daelemans, et al., eds., *ECML/Mlnet Workshop on Empirical Learning of Natural Language Processing Tasks*, Prague, pages. 105-111.

Keshava, S., Pitler, E., 2006. *A simpler, intuitive approach to morpheme induction.* Proceedings of 2nd Pascal Challenges Workshop, pages 31–35, Venice, Italy.

Koskenniemmi, K. 1983. Two-level Morphology: A General Computational Model for Word Form Recognition and Production. *Ph.D. thesis*, University of Helsinki.

Kurimo, M., Creutz, M., Varjokallio, M., Arisoy, E., Saraclar, M., 2006. *Unsupervsied segmentation of words into morphemes – Challenge 2005: An introduction and evaluation report*. Proc. of 2nd Pascal Challenges Workshop, Italy.

Li, M. and Vitányi,P. 1997. An Introduction to Kolmogorov Complexity and its Applications

(2nd ed.). Springer, New York.

Lovins, J.B.1969, Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, Vol. X, pages 22 – 31.

Oflazer, K., M. McShane, and S. Nirenburg. 2001. Bootstrapping morphological analyzers by combining human elicitation and machine learning. *Computational Linguistics*, 27(1): pages 59–85.

Porter, Martin F. 1980. An Algorithm for Suffix Stripping, *Program*, 14(3): 130–137

Rissanen, J. 1989. Stochastic Complexity in Statistical Inquiry. World Scientific Publishing Co, Singapore.

Schone, P. & Jurafsky, D. 2000. Knowledge-free induction of morphology using latent sematic analysis. *In CoNLL-2000 and LLL-2000.*

Schone, P. & Jurafsky, D. 2001. Knowledge-free induction of inflectional morphologies. *In NAACL'01*, pages 1–9.

Shalonova, K. & Golénia, B. 2010. Weakly Supervised Morphology Learning for Agglutinating Languages Using Small Training Sets. *Proceedings COLING '10 Proceedings of the 23rd International Conference on Computational Linguistics.* Pages 976-983