Morphology Induction from Limited Noisy Data Using Approximate String Matching

Burcu Karagol-Ayan, David Doermann, and **Amy Weinberg**
Institute for Advanced Computer Studies (UMIACS)
University of Maryland
College Park, MD 20742
{burcu,doermann,weinberg}@umiacs.umd.edu

Abstract

For a language with limited resources, a dictionary may be one of the few available electronic resources. To make effective use of the dictionary for translation, however, users must be able to access it using the root form of morphologically deformed variant found in the text. Stemming and data driven methods, however, are not suitable when data is sparse. We present algorithms for discovering morphemes from limited, noisy data obtained by scanning a hard copy dictionary. Our approach is based on the novel application of the longest common substring and string edit distance metrics. Results show that these algorithms can in fact segment words into roots and affixes from the limited data contained in a dictionary, and extract affixes. This in turn allows non native speakers to perform multilingual tasks for applications where response must be rapid, and their knowledge is limited. In addition, this analysis can feed other NLP tools requiring lexicons.

1 Introduction

In order to develop morphological analyzers for languages that have limited resources (either in terms of experienced linguists, or electronic data), we must move beyond data intensive methods developed for rich resource languages that rely on large amounts of data for statistical methods. New approaches that can deal with limited, and perhaps noisy, data are necessary for these languages.

Printed dictionaries often exist for languages before large amounts of electronic text, and provide a variety of information in a structured format. In this paper, we propose *Morphology Induction from Noisy Data (MIND)*, a natural language morphology induction framework that operates on from information in dictionaries, specifically headwords and examples of usage. We use string searching algorithms to morphologically segment words and identify prefixes, suffixes, circumfixes, and infixes in noisy and limited data. We present our preliminary results on two data sources (Cebuano and Turkish), give a detailed analysis of results, and compare them to a state-of-the-art morphology learner. We employ the automatically induced affixes in a simple word segmentation process, decreasing the error rate of incorrectly segmented words by 35.41%.

The next section discusses prior work on morphology learning. In Section 3 and 4, we describe our approach and MIND framework in detail. Section 6 explains the experiments and presents results. We conclude with future work.

2 Related Work

Much of the previous work on morphology learning has been reported on automatically acquiring affix lists. Inspired by works of Harris (1955), Dejean (1998) attempted to find a list of frequent affixes for several languages. He used successor and predecessor frequencies of letters in a given sequence of letters in identifying possible morpheme bound-

Proceedings of the Eighth Meeting of the ACL Special Interest Group on Computational Phonology at HLT-NAACL 2006, pages 60–68,
New York City, USA, June 2006.

aries. The morpheme boundaries are where the predictability of the next letter in the letter sequence is the lowest.

Several researchers (Brent, 1993; Brent et al., 1995; Goldsmith, 2001) used Minimum Description Length (MDL) for morphology learning. Snover and Brent (2001) proposed a generative probability model to identify stems and suffixes. Schone and Jurafsky (2001) used latent semantic analysis to find affixes. Baroni et al. (2002) produced a ranked list of morphologically related pairs from a corpus using orthographic or semantic similarity with minimum edit distance and mutual information metrics. Creutz and Lagus (2002) proposed two unsupervised methods for word segmentation, one based on maximum description length, and one based on maximum likelihood. In their model, words consisted of lengthy sequences of segments and there is no distinction between stems and affixes. The Whole Word Morphologizer (Neuvel and Fulop, 2002) uses a POS-tagged lexicon as input, induces morphological relationships without attempting to discover or identify morphemes. It is also capable of generating new words beyond the learning sample.

Mystem (Segalovich, 2003) uses a dictionary for unknown word guessing in a morphological analysis algorithm for web search engines. Using a very simple idea of morphological similarity, unknown word morphology is taken from all the closest words in the dictionary, where the closeness is the number of letters on its end.

The WordFrame model (Wicentowski, 2004) uses inflection-root pairs, where unseen inflections are transformed into their corresponding root forms. The model works with imperfect data, and can handle prefixes, suffixes, stem-internal vowel shifts, and point-of-affixation stem changes. The WordFrame model can be used for co-training with low-accuracy unsupervised algorithms.

Monson (2004) concentrated on languages with limited resources. The proposed language-independent framework used a corpus of full word forms. Candidate suffixes are grouped into candidate inflection classes, which are then arranged in a lattice structure.

A recent work (Goldsmith et al., 2005) proposed to use string edit distance algorithm as a bootstrapping heuristic to analyze languages with rich morphologies. String edit distance is used for ranking and quantifying the robustness of morphological generalizations in a set of clean data.

All these methods require clean and most of the time large amounts of data, which may not exist for languages with limited electronic resources. For such languages, the morphology induction is still a problem. The work in this paper is applicable to noisy and limited data. String searching algorithms are used with information found in dictionaries to extract the affixes.

3 Approach

Dictionary entries contain headwords, and the examples of how these words are used in context (i.e. examples of usage). Our algorithm assumes that each example of usage will contain at least one instance of the headword, either in its root form, or as one of its morphological variants. For each headword–example of usage pair, we find the headword occurrence in the example of usage, and extract the affix if the headword is in one of its morphological variants. We should note that we do not require the data to be perfect. It may have noise such as OCR errors, and our approach successfully identifies the affixes in such noisy data.

4 Framework

Our framework has two stages, *exact match* and *approximate match*, and uses three string distance metrics, the *longest common substring* (LCS), *approximate string matching with k differences* (k-DIFF), and *string edit distance* (SED). We differentiate between exact and approximate matches and assign two counts for each identified affix, *exact count* and *approximate count*. We require that each affix should have a positive exact count in order to be in the final affix list. Although approximate match can be used to find exact matches to identify prefixes, suffixes, and circumfixes, it is not possible to differentiate between infixes and OCR errors. For these reasons, we process the two cases separately.

First we briefly describe the three metrics we use and the adaptations we made to find the edit operations in SED, and then we explain how we use these metrics in our framework.

4.1 String Searching Algorithms

Longest Common Substring (LCS) Given two strings $p = p_1...p_n$ and $q = q_1...q_m$, LCS finds the longest contiguous sequence appearing in p and q. The longest common substring is not same as the longest common subsequence because the longest common subsequence need not be contiguous.

There is a dynamic programming solution for LCS[1] that finds the longest common substring for two strings with length n and m in $O(nm)$.

String Edit Distance (SED) Given two strings p and q, SED is the minimum number of edit operations which transforms p to q. The edit operations allowed are insertions, deletions, and substitutions. In our algorithm, we set the cost of each edit operation to 1. A solution based on dynamic programming computes the distance between strings in $O(mn)$, where m and n are the lengths of the strings (Wagner and Fischer, 1974).

Approximate string matching with k differences (k-DIFF) Given two strings p and q, the problem of approximate string matching with k differences is finding all the substrings of q which are at a distance less than or equal to a given value k from p. Insertions, deletions and substitutions are all allowed. A dynamic programming solution to this problem is the same as the classical string edit distance solution with one difference: the values of the first row of the table are initialized to 0 (Sellers, 1980). This initialization means that the cost of insertions of letters of q at the beginning of p is zero. The solutions are all the values of the last row of table which are less or equal to k. Consequently, the minimum value on the last row gives us the distance of the closest occurrence of the pattern.

String Edit Distance with Edit Operations (SED-path) In our framework, we are also interested in tracing back the editing operations performed in achieving the minimum cost alignment. In order to obtain the sequence of edit operations, we can work backwards from the complete distance matrix. For two strings p and q with lengths n and m respectively, the cell $L[n, m]$ of the distance matrix L gives us the SED between p and q. To get to the cell $L[n, m]$, we had to come from one of 1) $L[n-1, m]$ (insertion), 2) $L[n, m-1]$ (deletion), or 3) $L[n-1, m-1]$ (substitution). Which of the three options was chosen can be reconstructed given these costs, edit operation costs, and the characters $p[n], q[m]$ of the strings. By working backwards, we can trace the entire path and thus reconstruct the alignment. However, there are ambiguous cases; the same minimum cost may be obtained by a number of edit operation sequences. We adapted the trace of the path for our purposes as explained below.

Let *path* be the list of editing operations to obtain minimum distance, and *SED-path* be the SED algorithm that also returns a *path*. The length of the *path* is $max(n, m)$, and $path[j]$ contains the edit operation to change $q[j]$ (or $p[j]$ if $n > m$). *Path* can contain four different types of operations: Match (M), substitution (S), insertion (I), and deletion (D). Our goal is finding affixes and in case of ambiguity, we employed the following heuristics for finding the SED operations leading the minimum distance:

Case 1: If one string is longer than the other, choose I for extra characters

Case 2: Until an M is found, choose I in case of ambiguity

Case 3: If an M is found previously, choose M/S in case of ambiguity

Case 4: If there is an M between two I's, switch this with the last I

Case 1 ensures that if one word has more characters than the other, an insertion operation is selected for those characters.

If there is an ambiguity, and an M/S or I operation have the same minimum cost, Case 2 gives priority to the insertion operation until a match case is encountered, while Case 3 gives priority to match/substitution operations if a match case was seen previously.

Below example shows how Case 4 helps us to localize all the insertion operations. For the headword–candidate example word pair *abirids* → *makaabiríds*, the *path* changes from (1) to (2) using Case 4, and correct prefix is identified as we explain in the next section.

(1) `I M I I I M M M S M M` ⇒ Prefix *m-*
(2) `I I I I M M M M S M M` ⇒ Prefix *maka-*

[1] http://www.ics.uci.edu/ dan/class/161/notes/6/Dynamic.html

5 Morphology Induction from Noisy Data (MIND)

The MIND framework consists of two stages. In the exact match stage, MIND framework checks if the headword occurs without any changes or errors (i.e. if headword occurs exactly in the example of usage). If no such occurrence is found an approximate match search is performed in second stage. Below we describe these two stages in detail.

5.1 Exact Match

Given a list of (noisy) headword–example of usage pairs (w, E), the exact match first checks if the headword occurs in E in its root form.[2] If the headword cannot be found in E in its root form, for each e_i in E, the longest common substring, $LCS(w, e_i)$, is computed.[3] Let e_l be the e_i that has the longest common substring (l) with w.[4] If $w = l$, and for some suffix s and some prefix p one of the following conditions is true, the affix is extracted.

1. $e_l = ws$ (suffix) or
2. $e_l = pw$ (prefix) or
3. $e_l = pws$ (circumfix)

The extracted affixes are added to the induced affix list, and their *exact count*s are incremented. In the third case *p–s* is treated together as a circumfix.

For the infixes, there is one further step. If $w = w'l$ and $e_l = e'_l l$, we compute $LCS(w', e'_l)$. If $e'_l = w's$, for some suffix s, s is added as an infix to the induced affix list. (This means $e_l = w'sl$ where $w = w'l$.)

The following sample run illustrates how the exact match part identifies affixes. Given the Cebuano headword–example of usage pair (*abtik*) — (*naabtikan sad ku sa bátá*), the word *naabtikan* is marked as the candidate that has the longest common substring with headword *abtik*. These two words have the following alignment, and we extract the circumfix *na–an*. In the illustration below, straight lines represent matches, and short lines ending in square boxes represent insertions.

```
      a b t i k
  ▮ ▮ | | | | | ▮ ▮
  n a a b t i k a n
```

5.2 Approximate Match

When we cannot find an exact match, there may be an approximate match resulting from an error with OCR or morphophonemic rules[5], and we deal with such cases separately in the second part of the algorithm. For each e_i in E, we compute the difference between headword, and example word, *k-DIFF*(w, e_i). The example word that has the minimum difference from the headword is selected as the most likely candidate (e_{cand}). We then find the sequence of the edit operations performed in achieving the minimum distance alignment to transform e_{cand} to w using SED-path algorithm we described above.[6]

Let $cnt(X)$ be the count of X operation in the computed path. If $cnt(I) = 0$, this case is considered as an approximate root form (with OCR errors). The following conditions are considered as possible errors and no further analysis is done for such cases:

$$\begin{aligned} cnt(M) &= 0 \;\; || \\ cnt(M) &< max(cnt(S), cnt(D), cnt(I)) \;\; || \\ cnt(M) &< cnt(S) + cnt(D) + cnt(I) \end{aligned}$$

Otherwise, we use the insertion operations at the beginning and at the end of the path to identify the type of the affix (prefix, suffix, or circumfix) and the length of the suffix (number of insertion operations). The identified affix is added to the affix list, and its *approximate count* is incremented. All the other cases are dismissed as errors. In its current state, the infix affixes are not handled in approximate match case.

The following sample shows how approximate match works with noisy data. In the Cebuano input

[2] Headwords consisting of one character are not checked.

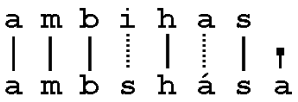
[3] In order to reduce the search space, we do not check the example words that are shorter than the headword. Although there are some languages, such as Russian, in which headwords may be longer than the inflected forms, such cases are not in the scope of this paper.

[4] Note that the length of the longest common substring can be at most the length of the headword, in which case the longest common substring is the headword itself.

[5] At this initial version, MIND does not make any distinctions between noise in the data such as OCR errors, and morphophonemic rules. Making this distinction will be one of our future focuses

[6] Computing k-difference, and the edit path can be done in parallel to reduce the computing time.

pair (*ambihas*) — (*ambshása pagbutang ang duha ka silya arun makakitá ang maglingkud sa luyu*), the first word in the example of usage has an OCR error, *i* is misrecognized as *s*. Moreover, there is a vowel change in the word caused by the affix. An exact match of the headword cannot be found in the example of usage. The k-DIFF algorithm returns *ambshása* as the candidate example of usage word, with a distance 2. Then, the SED-path algorithm returns the path *M M M S M S M I*, and algorithm successfully concludes that *a* is the suffix as shown below in illustration (dotted lines represent substitutions).



6 Experiments

6.1 Dictionaries

The BRIDGE system (Ma et al., 2003) processes scanned and OCRed dictionaries to reproduce electronic versions and extract information from dictionary entries. We used the BRIDGE system to process two bilingual dictionaries, a Cebuano-English (CebEng) dictionary (Wolff, 1972) and a Turkish-English (TurEng) dictionary (Avery et al., 1974), and extract a list of headword-example of usage pairs for our experiments. The extracted data is not perfect: it has mistagged information, i.e. it may include some information that is not the headword or example of usage, or some useful information may be missing, and OCR errors may occur. OCR errors can be in different forms: Two words can be merged into one, one word can be split into two, or characters can be misrecognized.

Dictionary Dictionary	# of pages	# of hw-ex pairs	# of words
Cebuano-all	1163	27129	206149
Turkish-all	1000	27487	111334
Cebuano-20	20	562	4134
Turkish-20	20	503	1849

Table 1: Details of Data from Two Dictionaries Used in Experiments

Along with the headword–example of usage pairs from more than 1000 pages, we randomly selected 20 pages for detailed analysis. Table 1 provides details of the data from two dictionaries we use in our experiments.

Both Cebuano and Turkish are morphologically rich. Cebuano allows prefixes, suffixes, circumfixes, infixes, while Turkish is an agglunative language. The two dictionaries have different characteristics. The example of usages in CebEng are complete sentences given in italic font while TurEng has phrases, idioms, or complete sentences as examples of usages indicated in bold font.

6.2 Protocol

We ran our algorithm first on all of the data and then on a randomly selected 20 pages from each dictionary. We manually extracted the affixes from each of the 20 pages. We then evaluated the MIND results with this ground truth. During the evaluation, even if the number of an affix in the ground truth and result are same, if they were extracted from different words, this is counted as an error. We also examined the cause of each error in this data.

We then compare our results from the whole TurEng data with the state-of-the-art Linguistica (Goldsmith, 2001) algorithm. Finally, we used the suffixes extracted by MIND and Linguistica to segment words in a Turkish treebank.

6.3 Analysis

Dict.	Affix	Sample words
C	mu-	galing/mugaling hikúhíkú/muhikùhíkù
E	nag-	kisdum/nagkisdum kugkugl/nagkugkug
B	mi-	iktin/miiktin kírus/mikárus
U	i-	kunsuylu/ikunsuylu pazíha/iparíha
A	na-	píl/napíl ulatl/naúlat
N	gi-	buga/gibuga dálit/gidádit
O	gi-an	labuk/gilabukan íkug/giikúgan
	-un	gihay/gihayun gáyung/gayúngun
	-a	pisar/pisara sirnpul/simpúla
T	-ı	ad/adı ilaç/ilaeı
U	-i	heves/hevesi ilim/ilmi
R	-a	saz/saza sonsuz/sonsuza
K	-e	deniz/denize zmim/mime
I	-ına	etraf/etrafına kolay/kolayına
S	-ya	hasta/hastaya orta/ortaya
H	-ü	üst/üstü zyüz/yüzü
	-ini	bel/belini zevk/zevkini
	-ine	derin/derinine iç/içine

Table 3: Sample Affixes Extracted from Two Dictionaries

Table 2 shows result of MIND runs. The total number of affixes and number of different types of

	Cebuano				Turkish			
	Whole dict.		20 pages		Whole dict.		20 pages	
Total	26106		542		27314		502	
Root form	5727		180		18416		345	
Prefix (diff. type)	10300	(180)	197	(26)	6	(6)	0	(0)
Suffix (diff. type)	1315	(253)	16	(8)	6983	(447)	128	(59)
Infix (diff. type)	25	(11)	0	(0)	1	(1)	0	(0)
Circumfix (diff. type)	717	(221)	18	(11)	9	(9)	0	(0)
App. Root form	1023		14		103		1	
App. Prefix (diff. type)	1697	(116)	23	(9)	8	(8)	1	(1)
App. Suffix (diff. type)	2930	(199)	63	(19)	168	(100)	5	(5)
App. Circumfix (diff. type)	1060	(207)	14	(5)	20	(20)	0	(0)
Couldn't decide	1159		13		765		15	

Table 2: Total Number and Different Types of Affixes Extracted from Two Dictionaries Using MIND

affixes (in parenthesis) are presented for two dictionaries, CebEng and TurEng, and two data sets, the whole dictionary and 20 randomly selected pages. The top part of the table gives the exact match results and the bottom part shows the approximate match results. For Cebuano, approximate match part of the framework finds many more affixes than it does for Turkish. This is due to the different structures in the two dictionaries. We should note that although MIND incorrectly finds a few prefixes, circumfixes, and infixes for Turkish, these all have count one. Table 3 contains some of the most frequent extracted affixes along with their exact and approximate counts, and samples of headword–example of usage word pairs they were extracted from. Each word is segmented into one root and one suffix, therefore when a word takes multiple affixes, they are all treated as a compound affix.

Dictionary	GT cnt.	Res.cnt.	Misses	Additions
Cebuano	311	314	17	14
Turkish	155	142	8	10

Table 4: Detailed Analysis of Affixes from 20 Pages

Table 4 shows the number of affixes in ground truth and MIND results along with number of missed and incorrectly added affixes on 20 of these pages of data. MIND only missed 5% of the affixes in the ground truth in both data sets.

We also examined the causes of each miss and addition. Table 5 presents the causes of errors in the output of MIND with an example for each cause. We should emphasize that a valid affix such as Turkish suffix *-mı* is counted as an error since the suffix *-ını* should be extracted for that particular headword–example of usage pair. An OCR error such as the misrecognition of *a* as *d*, causes both the miss of the prefix *mag-* and incorrect addition of *mdg-* for Cebuano. There are some cases that cannot be correctly identified by the framework. These usually involve dropping the last vowel because of morphophonemic rules. For the Cebuano dictionary, merge and split caused several errors, while Turkish data does not have any such errors. Main reason is the different structure and format of the original dictionaries. In the Cebuano dictionary, an italic font which may result in merge and split is used to indicate example of usages.

For the Cebuano data, five invalid suffixes, three invalid prefixes, and two invalid circumfixes are found, while one valid suffix and one valid circumfix are missed. For the Turkish data, three invalid suffixes, one invalid prefix, and two valid suffixes are found while two valid suffix are missed. When we look at the invalid affixes in the data, most of them (six of the Cebuano, and all of the Turkish ones) have count one, and maximum count in an invalid affix is five. Therefore, if we use a low threshold, we can eliminate many of the invalid affixes.

6.4 Comparison to Linguistica

We compared our system with *Linguistica*, a publicly available unsupervised corpus-based morphology learner (Goldsmith, 2001). Linguistica induces paradigms in a noise-free corpus, while MIND makes use of string searching algorithms and allows one to deal with noise at the cost of correctness. MIND emphasize segmenting a word into its root and affixes. We trained Linguistica using two different data sets from TurEng[7]: 1) Whole headword-

[7]We would like to do the same comparison in Cebuano. For the time being, we could not find a treebank and native speakers

Reason	Cebuano		Turkish	
OCR	8	M→lbi	11	ım→mı or ım
Algorithm	8	(uluy, giuylan)→ not gi-an, -lan is found	7	(alın, alnında)→ not -ında, -da is found
Merge	9	ímung giláug→ímunggiláug	0	-
Split	1	nag-kúgus→nag- kúgus	0	-
Other	5	apr.→april Headword is an abbreviation	0	-

Table 5: The Distribution of the Causes of Errors in 20 Pages with Samples

example of usage sentence pairs, and 2) Headword-candidate example words that our algorithm returns. In the first case (Ling-all), Linguistica uses more data than our algorithm, so to avoid any biases resulting from this, we also trained Linguistica using the headword and candidate example word (Ling-cand). We only used the suffixes, since Turkish is a suffix-based language. The evaluation is done by a native speaker.

Figure 1 presents the analysis of the suffix lists produced by Linguistica using two sets of training data, and MIND. The suffix lists are composed of suffixes the systems return that have counts more than a threshold. The results are presented for six threshold values for all of the data. We use thresholding to decrease the number of invalid affixes caused primarily by the noise in the data. For the MIND results, the suffixes over threshold are the ones that have positive exact counts and total counts (sum of exact and approximate counts) more than the threshold. Although Linguistica is not designed for thresholding, the data we use is noisy, and we explored if suffixes with a corpus count more than a threshold will eliminate invalid suffixes. The table on the left gives the total number of suffixes, the percentage of suffixes that have a count more than a threshold value, the percentage of invalid suffixes, and percentage of missed suffixes that are discarded by thresholding for the whole TurEng dictionary. The number of affixes MIND finds are much more than that of Linguistica. Furthermore, number of invalid affixes are lower. On the other hand, the number of missed affixes is also higher for MIND since, for this particular data, there are many affixes with counts less than 5. 41% of the affixes have an exact count of 1. The main reason for this is the agglunative nature of Turkish language. The effect of thresholding can also be examined in the graph on the right in Figure1 which gives the percentage of valid suffixes as a function of threshold values. MIND takes advantage of thresholding, and percentage of valid suffixes rapidly decrease for threshold value 1.

for Cebuano.

System	Th.	Total	Over Th.	Invalid	Missed
Ling-cand	0	6	100.00	0.00	0.00
Ling-all	0	4	100.00	0.00	0.00
MIND	0	60	96.67	1.72	0.00
Ling-cand	1	6	66.67	0.00	33.33
Ling-all	1	4	100.00	0.00	0.00
MIND	1	60	41.67	0.00	53.33
Ling-cand	2	6	50.00	0.00	50.00
Ling-all	2	4	75.00	0.00	25.00
MIND	2	60	18.33	0.00	76.67

Table 6: Total Number and Percentage of Over the Threshold, Invalid, and Missed Suffixes Found by Linguistica and MIND for Different Threshold Values for 20 pages of Turkish Data

Table 6 presents the same results for 20 pages from TurEng for three threshold values. MIND performs well even with very small data and finds many valid affixes. Linguistica on the other hand finds very few.

6.5 Stemming

To test the utility of the results, we perform a simple word segmentation, with the aim of stripping the inflectional suffixes, and find the bare form of the word. A word segmenter takes a list of suffixes, and their counts from the morphology induction system (Linguistica or MIND), a headword list as a dictionary, a threshold value, and the words from a treebank. For each word in the treebank, there is a root form (rf), and a usage form (uf). The suffixes with a count more than the threshold are indexed according to their last letters. For each word in the treebank, we first check if uf is already in the dictionary, i.e. in the headword list. If we cannot find it

System	Th.	Total	% Over Th.	% Invalid	% Missed
Ling-cand	0	116	100.00	18.10	0.00
Ling-all	0	274	100.00	34.67	0.00
MIND	0	499	89.58	13.20	3.61
Ling-cand	1	116	98.28	17.54	0.86
Ling-all	1	274	94.89	32.69	1.46
MIND	1	499	50.50	4.37	33.07
Ling-cand	2	116	92.24	16.82	5.17
Ling-all	2	274	87.96	31.12	4.74
MIND	2	499	38.48	4.17	44.49
Ling-cand	3	116	91.38	16.98	6.03
Ling-all	3	274	85.40	31.20	6.57
MIND	3	499	28.86	2.78	53.31
Ling-cand	4	116	81.03	12.77	11.21
Ling-all	4	274	81.39	30.94	9.12
MIND	4	499	25.65	3.13	56.51
Ling-cand	5	116	80.17	12.90	12.07
Ling-all	5	274	79.56	31.19	10.58
MIND	5	499	23.25	2.59	58.72

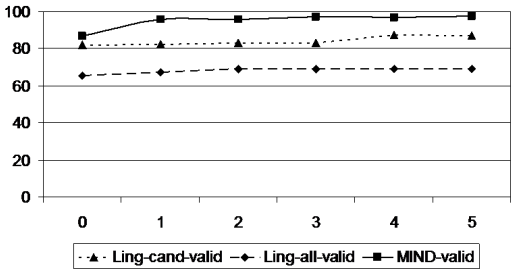


Figure 1: Total Number and Percentage of Over the Threshold, Invalid, Missed and Valid Suffixes Found by Linguistica and MIND for Different Threshold Values

in the dictionary, we repeatedly attempt to find the longest suffix that matches the end of uf, and check the dictionary again. The process stops when a dictionary word is found or when no matching suffixes can be found at the end of the word. If the word the segmenter returns is same as rf in the treebank, we increase the correct count. Otherwise, this case is counted as an error.

In our stemming experiments we used METU-Sabanci Turkish Treebank[8], a morphologically and syntactically annotated treebank corpus of 7262 grammatical sentences (Atalay et al., 2003; Oflazer et al., 2003). We skipped the punctuation and multiple parses,[9] and ran our word segmentation on 14950 unique words. We also used the headword list extracted from TurEng as the dictionary. Note that, the headword list is not error-free, it has OCR errors. Therefore even if the word segmenter returns the correct root form, it may not be in the dictionary and the word may be stripped further.

The percentage of correctly segmented words are presented in Figure 2. We show results for six threshold values. Suffixes with counts more than the threshold are used in each case. Again for MIND results, we require that the exact match counts are more than zero, and the total of exact match and approximate match counts are more than the threshold. For Linguistica, suffixes with a corpus count more than the threshold are used. For each threshold value, MIND did much better than Ling-cand. MIND outperformed Ling-all for thresholds 0 and 1. For the other values, the difference is small. We should note that Ling-all uses much more training data than MIND (503 vs. 1849 example of words), and even with this difference the performance of MIND is close to Ling-all. We believe the reason for the close performance of MIND and Ling-all in segmentation despite the huge difference in the number of correct affixes they found due to the fact that affixes Ling-all finds are shorter, and more frequent. In its current state, MIND does not segment compound affixes, and find several long and less frequent affixes. These long affixes can be composed

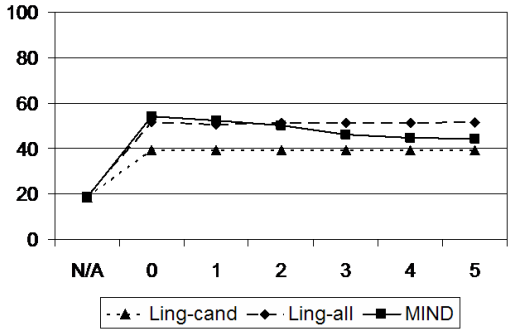


Figure 2: Percentage of Correctly Segmented Words by Different Systems for Different Threshold Values

[8] http://www.ii.metu.edu.tr/ corpus/treebank.html

[9] Multiple parses are the cases where a suffix is attached not to a single word, but to a group of words. The suffix *-ti* in *takip etti* is attached to *takip et*.

by shorter affixes Linguistica finds.

7 Conclusion and Future Work

We presented a framework for morphology induction from noisy data, that is especially useful for languages which have limited electronic data. We use the information in dictionaries, specifically headword and the corresponding example of usage sentences, to acquire affix lists of the language. We presented results on two data sets and demonstrated that our framework successfully finds the prefixes, suffixes, circumfixes, and infixes. We also used the acquired suffix list from one data set in a simple word segmentation process, and outperformed a state-of-the-art morphology learner using the same amount of training data.

At this point we are only using headword and corresponding example of usage pairs. Dictionaries provide much more information. We plan to make use of other information, such as POS, to categorize the acquired affixes. We will also investigate how using all the words in example of usages and splitting the compound affixes in agglunative languages can help us to increase the confidence of correct affixes, and decrease the number of invalid affixes. Finally we will work on identifying morphophonemic rules (especially stem-interval vowel shifts and point-of-affixation stem changes).

Acknowledgments

The partial support of this research under contract MDA-9040-2C-0406 is gratefully acknowledged.

References

Nart B. Atalay, Kemal Oflazer, and Bilge Say. 2003. The annotation process in the Turkish Treebank. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora–LINC*, Budapest, Hungary, April.

Robert Avery, Serap Bezmez, Anna G. Edmonds, and Mehlika Yaylalı. 1974. *Redhouse İngilizce-Türkçe Sözlük*. Redhouse Yayınevi.

Marco Baroni, Johannes Matiasek, and Harald Trost. 2002. Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 48–57.

Michael R. Brent, Sreerama K. Murthy, and Andrew Lundberg. 1995. Discovering morphemic suffixes: A case study in minimum description length induction. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, pages 28–36, Hillsdale, NJ.

Michael R. Brent. 1993. Minimal generative models: A middle ground between neurons and triggers. In *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, Ft. Laudersdale, FL.

Mathias Creutz and Krista Lagus. 2002. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*.

H. Dejean. 1998. Morphemes as necessary concepts for structures: Discovery from untagged corpora. In *Workshop on Paradigms and Grounding in Natural Language Learning*, pages 295–299.

John Goldsmith, Yu Hu, Irina Matveeva, and Colin Sprague. 2005. A heuristic for morpheme discovery based on string edit distance. Technical Report TR-2205-04, Department of Computer Science, University of Chicago.

John Goldsmith. 2001. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198.

Zellig Harris. 1955. From phoneme to morpheme. *Language*, 31:190–222.

Huanfeng Ma, Burcu Karagol-Ayan, David Doermann, Douglas Oard, and Jianqiang Wang. 2003. Parsing and tagging of bilingual dictionaries. *Traitement Automatique Des Langues*, pages 125–150.

Christian Monson. 2004. A framework for unsupervised natural language morphology induction. In *Proceedings of the Student Research Workshop: ACL 2004*, pages 67–72.

Sylvain Neuvel and Sean A. Fulop. 2002. Unsupervised learning of morphology without morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 31–40.

Kemal Oflazer, Bilge Say, Dilek Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish Treebank. In Anne Abeillé, editor, *Building and Using Parsed Corpora*. Kluwer Academic Publishers.

Patrick Schone and Daniel Jurafsky. 2001. Knowledge-free induction of inflectional morphologies. In *Second Meeting of the NAACL*, pages 183–191.

Ilya Segalovich. 2003. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine. In *Proceedings of MLMTA*, Las Vegas, NV.

P.H. Sellers. 1980. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, 1:359–373.

Matthew G. Snover and Michael R. Brent. 2001. A bayesian model for morpheme and paradigm identification. In *Proceedings of the 39th Annual Meeting of the ACL*, pages 482–490.

Robert A. Wagner and Michael J. Fischer. 1974. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173.

Richard Wicentowski. 2004. Multilingual noise-robust supervised morphological analysis using the wordframe model. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology*, pages 70–77, Barcelona, Spain.

John U. Wolff. 1972. *A Dictionary of Cebuano Visaya*. Southeast Asia Program, Cornell University, Ithaca, New York.