Treatment of Unknown Words

Jan Daciuk

Department of Applied Informatics, Technical University of Gdańsk,
Ul. Narutowicza 11/12, PL80-952 Gdańsk, Poland
`jandac@pg.gda.pl`

Abstract. Words not present in the dictionary are almost always found in unrestricted texts. However, there is a need to obtain their likely base forms (in lemmatization), or morphological categories (in tagging), or both. Some of them find their ways into dictionaries, and it would be nice to predict what their entries should look like. Humans can perform those tasks using endings of words (sometimes prefixes and infixes as well), and so can do computers. Previous approaches used manually constructed lists of endings and associated information. Brill proposed transformation-based learning from corpora, and Mikheev used Brill's approach on data for a morphological lexicon. However, both Brill's algorithm, and Mikheev's algorithm that is derived from Brill's one, lack speed, both in the rule acquisition phase, and in the rule application phase. Their algorithms handle only the case of tagging, although an extension to other tasks seems possible. We propose a very fast finite-state method that handles all of the tasks described above, and that achieves similar quality of guessing.

1 Introduction

A morphological analysis of words in a text is needed in many applications. It constitutes a prerequisite for natural language parsing and all applications that use it, and it is also useful in document retrieval. Such analysis is usually lexicon-based, i.e. it requires a morphological lexicon.

Unfortunately, real-world texts contain correct words that cannot be found in a lexicon. It seems impossible to record all words of a living language in a lexicon, as a lexicon is static in nature, and a language is a living thing – new words are coined continually. Another reason for finding words not present in the lexicon is the Zipf's law[Bri93]. The Zipf's law states that the rank of an element divided by the frequency of occurrence is constant. E.g. in the Brown corpus, two percent of different words[1] account for sixty nine percent of the text. About seventy five percent of different words occur five or fewer times in the corpus. Fifty eight percent of different words occur two or fewer times, and forty four percent only occur once. The consequence of the Zipf's law is that by doubling the number of words in the lexicon, one gets only a few percents of the coverage

[1] Eric Brill uses the terms *word type* and *token*

of an arbitrary unrestricted text. Therefore, increasing the size of the lexicon is a very costly effort yielding minute results.

New words are also constructed by derivation on compounding. The number of potential words formed in the same way is huge. Therefore, it is not practical to store all such derivatives and compounds in the lexicon. In many cases there may be many ways to form a new word, and it is not possible to predict which one would be chosen by humans as correct.

Additionally, texts may contain incorrect words. For purpose of e.g. spelling corrections, the morphological categories of a misspelled word may help reduce the list of possible corrections. If the misspelling does not affect the word's flectional ending (and its prefix, if present), these categories may still be easily obtainable from the corrupted version.

Some of previously unseen words eventually make their way into the (morphological) lexicon. In order to do that, we need to give them morphological descriptions. The task of writing them by hand is laborious; it would be much easier to choose a description from a list of feasible descriptions.

Humans can perform all those tasks quite well. The reason they can do that is that they can associate information they want to extract with endings of words (sometimes also with their prefixes and infixes). So can do computers. We propose a fast finite-state technique to accomplish that.

2 Related Work

In the past, various hand-crafted heuristics have been used for the purpose of morphological analysis of unknown words. Later, they have been supplemented by statistical techniques (e.g. [WMS+93]). However, although the probabilities of different endings leading to their corresponding categories were calculated, the endings themselves were chosen manually. A revolutionary approach was proposed by Eric Brill ([Bri93], [Bri95]). The endings, as well as prefixes, are found by the program. Unknown words are first tagged by a naive initial-state annotator that assigns the tags proper noun or common noun on the bases of their capitalization. Then five types of transformations are applied:

Change the tag of an unknown word (from X) to Y if:

1. Deleting the prefix (suffix) x, $|x| \leq 4$, results in a word (x is any string of length 1 to 4).
2. The first (last) (1,2,3,4) characters of the word are are x.
3. Adding the character string x as a prefix (suffix) results in a word ($|x| \leq 4$).
4. Word w ever appears immediately to the left (right) of the word.
5. character z appears in the word.

The result is compared with a tagged corpus. The best-scoring rule is chosen, and applied to the corpus so that it becomes new input data. The learning stops when no rule can increase the score. The transformation type 4 takes into account the context of the unknown word. When the morphological analysis is separated from a contextual tagger, as it is the case with our approach, the tagger must find those rules itself. The transformation types 1 and 3 checks whether adding or deleting characters from/to a word results in another word. In our algorithm those transformations are not present, as well as transformation type 5, which can be treated, if necessary, as a supplementary heuristics. The rules schemata presented above do not account for infixes, which can be treated with our method.

Andrei Mikheev ([Mik97]) applied Brill's transformations to the data for a (pre-existing) morphological lexicon. He uses the following template:

$$G =_{x:\{b,e\}} [-S \ + M \ ?I\text{-class} \rightarrow R\text{-class}],$$

where:

- x indicates whether the rule is applied to the beginning or end of a word and has two possible values: b – beginning, e – end;
- S is the affix to be segmented; it is deleted (-) from the beginning or end of an unknown word according to the value of x;
- M is the mutative segment (possibly empty), which should be added (+) to the result string after segmentation;
- i-class is the required POS-class (set of one or more POS-tags) of the stem; the result string after the -S and +M operations should be checked (?) in the lexicon for having this particular POS-class; if I-class is set to be "void" no checking is required;
- R-class is the POS-class to assign (→) to the unknown word if all the above operations (-S +R ?I) have been successful

Compared to Brill's algorithm, Mikheev checks also (optionally) the morphological class (a set of categories) of the resulting word in transformation templates 1 and 3. Also, his algorithm returns all categories of an unknown word, not only the most probable one.

Although Mikheev is not aware of that fact[2], the rules learnt by Brill's algorithm can be transformed into a finite-state automaton. However, that process is complex and time-consuming. The learning process takes time as well. Our algorithm produces the FSA directly from data exploring the links that occur naturally in the format of data we use. In Brill's approach, copied by Mikheev, the length of suffixes and prefixes is a constant. Increasing it means much more computation. In our method, the suffixes are discovered naturally, so there is no need to limit their lengths.

3 Finite-state Approach

Our approach is based on the observation that it is possible to associate endings with required information by inverting inflected words, and sticking the required

[2] [RS95] does not appear among references in [Mik97]

information (an *annotation*) at the end. By performing that operation on all inflected forms in the lexicon, we get a finite set of strings. Therefore, it is possible to convert it into a minimal, deterministic, acyclic, finite-state automaton that we call a *guessing automaton*. To find appropriate annotation for unknown word, we need to invert the word, and then look for it in the guessing automaton.

3.1 Data

The exact format of the annotation depends on the information we want to put into it. In a general case, we need a special symbol that we call an *annotation separator* to separate annotations from the inflected forms. For reasons that we explain later, we also need another special symbol to mark the end of the inverted inflected form; we put that symbol in front of the annotation separator. If the annotation should consist of morphological categories of the inflected form, we simply put them after the annotation separator. Example:

`abmob_+Verb[mode=ind tense=past num=sg person=3]`

If the annotation is to be the base form, a little bit of coding is necessary in order to avoid inflating the automaton. We assume that it is only the ending that is different in the base form as compared with the inflected form. Therefore, we can replace the full base form with a code that says how many characters are to be deleted from the end of the inflected form, and a string consisting of characters that are to be appended to obtain the base form. When no characters are to be deleted, we put 'A' there, one character – 'B', etc. Example:

`abmob_+Ber`

It is possible to put both the base form, and the categories in annotations. Example:

`abmob_+Ber+Verb[mode=ind tense=past num=sg person=3]`

Annotations for morphological data acquisition are more complicated, as the base form may be different form the lexical form, and the lexical form may contain arch-phonemes. Also, they depend on the particular morphology program we use. Therefore, we will not give any examples of that here. The annotation formats we present here are very simple; however, by modifying the annotation format we can successfully handle prefixes and infixes as well.

3.2 Pruning

As word endings normally decide what annotation should be associated with a given word, the automaton has a particular structure. For a given word, the first few states have many outgoing transitions. Then, there is a chain of states linked with each other with single transitions. Passing to annotations, a more complicated transition network appears again. As for any state in the central part, there is only one way leading from the state to the appropriate annotations, it represents no useful information for our purpose. So all states from that part can be pruned, along with the corresponding transitions. Pruning explains the need for the special symbol marking the end of the inverted inflected form (the beginning of the inflected form). We need it because we no longer have full words in the automaton, sometimes entire shorter words may constitute the end part of longer words, and different words may have different annotations, so we need to distinguish them.

Pruning is governed by the following rules:

Rule R 1. The pruning process does not apply to transitions belonging to annotations.

This rule should be obvious, because annotations are what we want to obtain in the recognition phase. The transitions that are pruned belong to the inflected forms, and more precisely: to their beginnings that do not influence the annotations.

Rule R 2. A transition can be removed only if the pruning process has already visited all transitions that can be reached through the target state, except the transitions representing annotations. In other words, the states are visited (and the outgoing transitions pruned if possible) in the postorder method.

This means that we traverse the automaton recursively in depth, cutting unneeded transitions on the way back.

Rule R 3. A transition cannot be removed if the target state has a transition that does not belong to annotations, but cannot be removed.

This means that the target state has transitions that distinguish between different annotations, i.e. they lead to different sets of annotations. We do not want to lose that distinction.

Rule R 4. A state can always be replaced with an equivalent one (i.e. a state with the same transitions leading to the same states).

The automaton should be kept minimal.

Rule R 5. A state with all transitions leading to one state (the target state) can be removed with all transitions, and transitions that point to it should be replaced by transitions pointing to the target state.

This is the rule that actually cuts transitions. Note that it also applies to states with one only transition.

The consequence of the rules R2 and R5 is that we cannot remove states (and transitions that lead to them) if they have transitions leading to different sets of annotations. This is because the rule R2 ensures that all transitions of a visited state lead to states that either cannot be removed, or that have an outgoing transition labeled with the annotation separator.

The process described above leads to the construction of a finite state automaton that contains all required information. However, the automaton is still big, and an effort can be made to further reduce its size. Looking at its contents, we can see many states with a majority of transitions leading to one state (we will use the term *default state*), but with other transitions as well. The target state must have one outgoing transition, and it must be labeled with the annotation separator. We can treat less frequent transitions as exceptions, assuming that *all* other transitions, even those that had not appeared in our lexicon, lead to the default state. Acting under this assumption, we can replace the frequent transitions and the default state with the transition leading from the default state. A limit can be imposed on the ratio of frequent/less frequent transitions to trigger the pruning.

There is a difference between rules R1, R2, ... R5, and this rule. Rule R6 introduces a generalization. While still 100% of words present in the lexicon are annotated correctly, R6 may select one annotation among many as the correct one, and hide other possibilities. This may speed up an annotation process, but it can also introduce errors: some correct (but less probable) possibilities may not be shown, as the lexicon may not contain data that associates their endings with their correct annotations.

Rule R 6. If for a given state the number of transitions leading to one state (the default state) is greater or equal to the number of all other transitions multiplied by a small integer, and the default state has only one outgoing transition and it is labeled with annotation separator, then the default state can be removed, and all transitions that lead to it should be replaced by the transition leaving the default state.

Sometimes, it is impossible to devise a rule that associates an ending with the correct annotation, because the choice is lexicalized, i.e. it depends on a particular word, and it seems arbitrary from the morphological point of view. For example, in Polish, there is a rule that transforms adjectival endings *-sny* in lexemes into *-śniejszy* in comparatives and superlatives. There is, however, another rule that transforms endings *-śny* into *-śniejszy* in comparatives and superlatives. So there is no other way of knowing what the lexeme might be from a comparative or superlative ending other than a dictionary lookup. R6 introduces artificial divisions, e.g.:

-raśniejszy → -raśny
-maśniejszy → -maśny
-waśniejszy → -waśny
-ośniejszy → -ośny
-bleśniejszy → -bleśny
-uśniejszy → -uśny

jaśniejszy → jasny
-iaśniejszy → -iasny
-dośniejszy → -dosny
-ześniejszy → -zesny
-oleśniejszy → -olesny

while the right answer is that both annotations must be considered:

-śniejszy → -śny
-śniejszy → -sny

To cope with that situation, we introduce a new rule that strives to accommodate such cases. We will use the term *first annotated state* to name a state that is a target of a transition labeled with the annotation separator (a state that begins an annotation or a set of annotations).

Rule R 7. If for a given state the number of first annotated states that are reachable from the given state does not exceed a given limit, then:

- replace the first annotated states by their union;
- replace all the states and transitions between the chosen state and the union of the first annotated states by a single transition labeled with the annotation separator.

Note that it is possible to introduce a lower limit on the number of states to be removed in order to insure that we are dealing with a case such that the one described above (*sny* and *śny*). The rule can then work in parallel with R6.

It is worth noting that while the rule R6 introduces very detailed distinctions, R7 discards details. For the guesser, the result of applying R7 is that one gets more choices than without having applied R6 or R7. As to the lexicon size, R7 removes small differences between similar word forms, making it possible to infer more general and compact relations between endings and annotations.

Please note that although no annotation possibility is lost, and the automaton is much smaller, the answers for known words are no longer 100% accurate. The correct answer appears always, but it may be accompanied by other, incorrect possibilities. In many cases exceptions are merged with regular rules. A lower limit imposed on the number of states to be removed by this rule can solve the problem.

3.3 Recognition

It is mostly endings that decide what annotations a word may have. To get an annotation, we invert the unknown word, put the special symbol (word beginning marker) at the end, and look for such string in the automaton. Sooner or later, we come to a state that has no transition labeled with the subsequent letter of the string. That state may have a transition labeled with the annotation separator. If it has one, then the right language of that state is the set of annotations for the word. If not, we recursively look at the descendants of the state, searching for states with transitions labeled with the annotation separator, and adding the right languages of the states they lead to to the resulting annotation of the unknown word.

4 Results

Experiments were carried out on French morphology from ISSCO, Geneva, Switzerland. The data for the morphological lexicon was divided in 10 parts, 9/10 were used to construct guessing automata, and 1/10 as a source of words whose annotations were to be guessed. This was done ten times, i.e. for each pair (9/10, 1/10). Standard measures of recall, precision, and coverage were used to evaluate

Table 1. Recall and precision for predictions of properties of unknown words

	categories only			categories and base forms		
	R1-R5	R1-R6	R1-R5,R7	R1-R5	R1-R6	R1-R5,R7
recall	94.57	93.92	97.80	93.43	92.74	95.93
precision	90.03	91.64	64.81	88.45	90.34	61.42

Table 2. Recall, precision, and coverage for predictions of morphological descriptions of unknown words

	R1-R5	R1-R6	R1-R5,R7
recall	93.22	92.47	94.67
precision	86.26	88.27	66.31
coverage	99.94	99.96	99.99

the results. Recall and precision were calculated for each item, and the average of all items was calculated for each part.

Table 1 shows recall and precision for prediction of morphological categories, and both morphological categories and base forms. The coverage is 100% in all cases. For the rule R6, we chose that there should be twice as many transitions leading to the default state than to other states. For the rule R7, we merged only two states at a time, they had to be the only children of a given state, and we did not count the transitions that led to them. By setting a threshold on the minimum number of transitions leading to the states that are to be merged by the rule R7, we can raise precision, but lower the coverage (e.g. to 95.99% and 75.39% for categories only, and to 94.48% and 74.10% for both categories and base forms).

The results show that if we want to minimize the number of choices and raise precision, we should use rules R1-R6. This is the case of simple POS-tagging. In cases where we want to make sure that we do not miss any possibility, we should use rules R1-R5 and R7.

Mikheev ([Mik97]) claims achieving 95.24% recall, 85.16% precision, and 92.66% coverage on categories only (he did not consider base forms). Note that we have 100% coverage, so his recall and precision should probably be multiplied by 0.9266 in order to be compared directly to ours. However, he performed experiments on English words, and it is not clear what the impact of the chosen language is on the results. He also used smoothing on a corpus. Since we have access neither to his programs, nor to the data he used, the only way we could compare our results to his would be to emulate his approach on our data. Our experiments required ca. 14 minutes to build all 10 guessing automata for one set of rules, and 2 minutes 15 seconds to evaluate the results on Pentium II 350 MHz with 128 MB memory running under Linux.

Table 2 shows the results for guessing morphological descriptions of words. The coverage is no longer 100%; this is probably caused by existence of arch-

Table 3. Size of the automaton as function of rules

	categories only			morphological descriptions		
	R1-R5	R1-R6	R1-R5,R7	R1-R5	R1-R6	R1-R5,R7
states	19800	18507	10123	39832	38301	27608
transitions	65635	48419	32729	99535	78976	61077

phonemes. The format of descriptions was the one used by mmorph tool ([PR95]) from ISSCO, Geneva.

The choice of rules influences the size of the automaton. Table 3 shows that relation for guessing only categories, and for guessing morphological descriptions. We can see that R6 cuts mostly transitions, but R7 cuts both transitions and states.

References

[Bri93] Eric Brill. *A Corpus-Based Approach to Language Learning*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, USA, 1993.

[Bri95] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, December 1995.

[Mik97] Andrei Mikheev. Automatic rule induction for unknown-word guessing. *Computational Linguistics*, 23(3):405–423, September 1997.

[PR95] Dominique Petitpierre and Graham Russell. MMORPH - the Multext morphology program. Technical report, ISSCO, 54 route des Acacias, CH-1227 Carouge, Switzerland, October 1995.

[RS95] Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253, June 1995.

[WMS+93] Ralph Weischedel, Marie Meteer, Richard Schwartz, Lance Ramshaw, and Jeff Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382, 1993.