Statistical Methods for Morphological Parsers

László Kovács, Péter Barabás

University of Miskolc, IIS, Miskolc-Egyetemváros, Miskolc 3515, Hungary
kovacs@iit.uni-miskolc.hu, barabas@iit.uni-miskolc.hu

1 Introduction

Computational linguistic (CL) covers the statistical and logical modeling of languages using computer-based software-hardware tools. In the 1960s, the research activities in computational linguistic were based mainly on a symbolic approach. The symbolic approach was developed out among others in the works of Chomsky [2], Harris [5] and Shannon [7]. Their methods use mainly the technologies of the parsing and artificial intelligence systems. In the shadow of symbolic approach, a smaller group of researchers stared to work out the stochastic approach. The main pioneer representatives of this direction are among others Browing [1] and Wallace [8]. The methods of the stochastic approach used mainly the Bayesian statistical algorithm to generate the language models. Although, the stochastic approach had a minor role initially, it turned out in the following decades, that it should have a dominant importance in the computational linguistic. Among the most successful stochastic methods can be given the Hidden Markov Model, the N-gram models, the Finite State models, classification methods and the artificial neural networks [6].

A basic element in CL is a formal grammar that describes the language characteristics. Chomsky [3] proposed first to use grammar models based on rewrite rules. In this approach, the language is given with a set of

$$a \to b \tag{1}$$

rules where a and b are strings of the language. Using these rules recursively, the valid sentences of the language can be generated. One of the simplest form of the rewrite rule system is the context-free grammar [6], where the element a is a single symbol, thus the output b is independent from the context of the symbol a. Based on the rule system, it can be verified whether a sentence is valid or not. The parsers [6] are used to perform validation and to determine the construction structure of input sentences. The application of the first parsers to the human languages had minor success. The nature of the human languages posses some elements which can not be covered with strict automation systems. Thus, the main difficulties were the ambiguity in usage of the words, the huge set of exceptions and the very flexible word generation schemes. To cope with these problems, all

of the special cases should be encoded into the model. This requires a major extra cost in time and space. Thus, the applications of the natural language processing (NLP) methods were tailored first to some limited problem areas.

The first industry products of the NLP systems were the language translators. The first translators worked on a word to word basis with a limited success. It is widely assumed, that the translator systems should possess metadata on the context of the text to be translated. The set of the required metadata depends on the purpose of the translation.

An important type of parsers used in NLP systems is the morphological parser. In the most languages, a concept is assigned to several variants of a base word. The context of the concept occurrence determines which variant should be used. A morpheme is the minimal unit with meaning in the language [6]. The key morpheme for a concept is the stem. All of the transformations are defined on the stems. The stems should determine the base concept. The context is given with affixes. The affixes give additional meaning of various kinds. Depending on the location of the affix, the affix unit may be called prefix, suffix, infix or circumfix [6].

The application of affixes may result in a new concept. In this case, the transformation is called derivation. If the output belongs to the same concept family, the transformation is called inflection. In the agglutinative languages, the inflections are more complex, a stem can be extended with ten or more affixes.

There are several languages which grammar is not too difficult, it has no or just a few exceptions. In case of these languages the grammar induction process is more simpler than in languages have intricate grammar like Hungarian language. The problem of living languages is that the dictionary is not persistent and not static. In this case the statistical language processing can be a solution. The main statistical methods to describe a morphological parser are the Hidden Markov Models (HMM), the N-gram models (NGM) and the Finite State Transducers (FST). The grammar of the language is encoded in these methods in the structure and in the parameters of the model.

Based on our experiences, these models provide very high precision for the trained words, but they have some limitations in the case of untrained words. In this paper, the description of the main methods and a short outline of a special morphological parser are given. The HMM, NGM, FST methods and the proposed new model are compared in a test application. In the test application, the parsers should learn the inflection for the accusative. The host language is the Hungarian language which is an agglutinative language.

2 Finite State Transducer

The Finite State Transducer (FST) is a useful tool in the conversion of strings and texts. The input pool of this problem area is a set of string pairs (s_1, s_2). There exists here an apriori unknown function on the set S of strings:

$$f : S \rightarrow S. \tag{2}$$

The input pool is a training set of this function, i.e.

$$s_2 = f(s_1). \tag{3}$$

The FST structure can be used for several purposes in the management of this mapping function [9]. The possible application areas:

- FST as recognizer: it can determine if $s_2 = f(s_1)$ or not
- FST as translator: it generates $f(s_1)$ from s_1.
- FST as generator: it generates $(s_1, f(s_1))$ pairs of words
- FST as relater: it computes the relation between sets.

Structurally, the FST is an extension of the Finite State Automate (FSA). Similar to FSA, the FST consists of the following elements:

- Q: finite set of states
- E: finite set of directed edges (transition function between the states)
- A: finite set of symbols (alphabet).

There are two special subsets of Q: the set containing the initial state and the set of final states. The alphabet is built up from two sets: an AI alphabet of input symbols and an AO alphabet of output symbols. The transition edge is marked with a (q, i:o, r) triplet where q denotes the start node, the r is the end node, the i is a symbol from AI and the o denotes an element from AO.

The phonological rewrite rules can be expressed as regular expression with the help of finite state machines. The FST can be used to handle the two-level morphology. The $\{s_1\}$ set of input strings is the lexical level, and the output level is called surface level. The FST is based on a lexicon containing all of the required grammar transformation rules. If the background lexicon does not exist as it is true in general case, the system should learn the FST network. It is known that learning of a general FST is intractable. In the practice, a special case of the general FST, the sequential FST can provide an acceptable solution. The Sequential Transducer (ST) is deterministic and they preserve increasing length input-output prefixes. This restriction is very dominant, thus the model can be used only for very simple languages.

To provide more general solution, the subsequential transducer (SST) model was proposed. In this model, a new model element, the state output function was

introduced. The output function results in an output strings for every state. The final output string is the concatenation of the edge output with the state function output. A good comparison of the ST and SST structure is given in [4] paper. In the onword form of the transducers, the output substring is assigned to the edges in such a way that that they are as close to the initial state as they can be. One of the most efficient algorithm to generate the SST structure the OSTIA method (Onword Subsequential Transducer Inference Algorithm).

The OSTIA algorithm is based on the following core steps.

- generating a simple structured ST,
- merging some nodes,
- pushing some output elements toward the initial state,
- eliminating all of the non-deterministic elements,

The OSTIA method generalizes the training examples as much as possible, this may lead to over-generalization.

3 N-grams and Markov Models

In statistical language processing, N-gram model is a widely used method. N-gram models are essential in speech recognition, handwriting recognition, machine translation, spelling correction, part-of-speech tagging, natural language generation and in any task which we have to identify words in noisy, ambiguous input. The goal is to compute the probability of a word *w* given some *h* history, or P(w|h). The sequence of *N* words will be presented in [9] paper as

$$w_1, \ldots, w_n \text{ or } w_1^n . \tag{4}$$

The computation the probabilities of entire sequence can be performed using the chain rule of probability for decomposition [9]:

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1}) = \\ = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1}) \tag{5}$$

The chain rule shows that the joint probability of a sequence can be computed from the probability of words given previous words. But using the chain rule doesn't really help us, because we don't know the way how to compute the probability of a word a long sequence of preceding words. It can not be estimated by counting word occurrences following a long sequence of words, because the language is creative and can be produce sequences never seen before. Thus instead

of computing the probability of a word given its entire history, the probability of a word by just a few preceding words is approximated.

The bigram model is such an N-gram model where the probability of a word is approximated by the preceding word [9]:

$$P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-1}\right), \tag{6}$$

whereas in trigram model this probability is:

$$P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-2}^{n-1}\right). \tag{7}$$

In general we can make the following approximation:

$$P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-N+1}^{n-1}\right) \tag{8}$$

This property that the probability of a word depends only on the previous word is called Markov property stated as [10]:

$$P\left(\xi_{t+1} = s_{i_{t+1}} \mid \xi_1 = s_{i_1},\ldots,\xi_t = s_{i_t}\right) = P\left(\xi_{t+1} = s_{i_{t+1}} \mid \xi_t = s_{i_t}\right), \tag{9}$$

where $\xi_1,\ldots,\xi_n$ are random variables.

In a Markov chain, each state is attributed with a finite set of signals. After each transition, one of the signals associated with the current state is emitted. Thus, we can introduce a new sequence of random variables η_t, $t=1\ldots T$, which is the emitted signal in time t. A Markov model consists of the following elements [10]:

- a finite set of states $\Omega = \{s_1,\ldots,s_n\}$;
- an signal alphabet $\Sigma = \{\sigma_1,\ldots,\sigma_m\}$;
- a n x n state transition matrix $P = [p_{ij}], where\ p_{ij} = P\left(\xi_{t+1} = s_j \mid \xi_t = s_i\right)$;
- an n x m signal matrix $A = \{a_{ij}\}, where\ a_{ij} = P\left(\eta_t = k_j \mid \xi_t = s_i\right)$
- an initial vector $v = [v_1,\ldots,v_n], where\ v_i = P\left(\xi_1 = s_i\right)$

In every discrete point of time, a state transition occurs depending on the transition probabilities and after each transition. Then one of the signals associated with the current state is emitted. If a state in the model depends only on the previous state, the model is called first-order Markov model. When a state depends on more preceding states, a higher-order Markov model is used. Bigram model is a first-order Markov model, whereas trigram model is a second-order and N-gram model is a $(n-1)^{th}$-order Markov model.

If the sequence of states can not be directly observed and only the sequence of emitted signals can be measured, the model is called hidden Markov Model. The

sequence of states can be then evolved with use of signal matrix, which defines the probability of an emitted signal in a given state. Let $O = O_1O_2...O_T$ be a sequence of observation, where O_i is an emitted signal. Hidden Markov models have three prototypical tasks:

1 estimate the probability of observation sequence, or *P(O)*,

2 determine the most probable state sequence to the given signal sequence,

3 determine the model parameters $\lambda = (P, A, v)$ to maximize the probability of a given signal sequence.

4 Problem of Generalization

One of the main characteristics of a learning system is the capability of generalization. If the training set is incomplete, the system may meet untrained cases in the production phase. It can be assumed that the unknown input object has some common attributes with the other input objects met in the training phase. Thus, the input objects are similar to each others in some extends. An efficient learning system is able to detect the common characteristics of an input object and it can infer the most probable output object. The capability of generalization makes the learning system to work from incomplete training set too. Of course, in the case of untrained input objects, the learning system can give only a probabilistic result which may differ from the real output value. With comparison of the calculated and of the measured output values for the untrained objects, the generalization quality of the learning method can be determined. A good generalization facility is very useful from a practical viewpoint. It has the following benefits:

- it can work with incomplete training data
- the training cost can be reduced
- it is better suitable for unexpected problems

From the viewpoint of generalization, the FST method can not provide a good performance. The initial FST tree is a suffix tree of the trained data. During the generalization phase, the connected nodes with deterministic output edges are merged into a common node. As the FST is based on a sequential processing principle, the first decision steps are made at the beginning of the words. If the training pool has examples with similar prefix parts, the parsing path is lead into that direction. This decision is most of the times incorrect, as mostly not the initial part of the words determine the inflection rule but some other parts of the word. The performed generalization enables only a limited permutation of the character sequence. As a result, the FST could not provide good outputs in our test result.

The HMM method, on the other hand, manages the words in a more flexible way. The basic units of the parsing are the N-grams. To learn the inflection transformation, the stems are considered as the output words and the inflected forms are the internal N-grams as the internal states. Based on this required structure, the set of all possible N-grams in the inflected forms should be known apriori. As this assumption can not be met in general, only words having known N-grams can be inferred. Another restriction of the HMM method is that the length of the inflected form should be no less than the length of the stem forms. This restriction comes from the internal structure of the HMM as in the Trellis probability matrix each time step is assigned to one internal state.

Based on these experiences, the outline of a more flexible parsing system was worked out in the frame of the project. The proposal considers the inflection transformation as a classification problem. The inflection process is treated as function:

$$t: w \rightarrow w' \tag{10}$$

The t function can be described with a transformation mask. Based on the experiences, there exist a set of transformation masks and several stem words may share the same mask. Thus, the task of the parser is to select the appropriate mask. In this sense, each stem word belongs to one transformation mask. The mapping of the stem words to transformation mask is regarded as a classification problem. In order to provide a flexible classification, a classification lattice is built up from the training examples. This lattice has a similar role to the concept lattices, thus it serves to provide an efficient generalization. The generalized masks are generated with the mask intersection operation. Each parent node has a more general mask than the child's mask. A problem of this kind of generalization is that a lot of noise masks are generated. A noise mask means here that it reflects not the real dependency, it depends on the random distribution of the selected training examples. To select the core dependencies, the different mask subparts are weighted based on their location within the mask. Another new element of the proposed system refers to the cost-effective management of the lattice. In the initial lattice, the lookup of a new stem word in the lattice finishes if it reaches a node with unambiguous class assignment. In this case all of the child nodes belong to this class. It is possible to transform the initial lattice to an exception-centered storage where the children of the same class are eliminated, only the exception child nodes are stored in the lattice. According to the test experiences this kind of parsing provides a better generalization results than the previous methods.

5 Test Experiments

The initial test included only 140 training examples. This is only a very thin subset of the larger example pool containing about 90000 examples. This larger training

pool was generated from an online dictionary with manual entering of the inflection forms.

The set of untrained words contains ten elements:

$$U = \{gatya, labda, tanár, krumpli, fej, korong, csapat,köd, ló, korom\}$$

The inflected forms of the words are the followings:

$$R = \{gatyát, labdát, tanárt, krumplit, fejet, korongot, csapatot, ködöt, lovat, kormot\}$$

In the initial FST test, the following result was obtained:

$$R_{FST} = \{g, l, tanítót, k, fecskét, k, csattot, kö, lócát, k\}$$

This means a 0% precision value. The actual FST graph contained 686 nodes.

In Hungarian language 36 (including space) different character can be found, thus the number of possible trigrams is 46656. The 140 training examples contain 570 different trigrams which is 1.2 percent of all of the possibilities. As test results show the inflected form of the teaching nouns was generated with 100% precision value and working with 570 nodes was very fast.

Test results are contrary in case of testing with set *U* of words. The precision value was 0%, because all of the inflected words were in form *faaa* where number of *a* is equal to the length of the estimated inflected form-1. This poor result has more reasons:

- HMM contains only those states and signals which derived from teaching datas (570 different trigrams), thus in test words there are such trigrams which are not present among HMM states and signals,
- a state which is not present among HMM states has no signals and a signal which is not present among HMM signals has no states,
- the initial values in state transition matrix and signal matrix are zero for all cells.

The base problem is that there are a few states and signals in HMM and if trigrams of testing words cannot be found among the signals of HMM, the result will be poor.

Another test examines the behavior of the model trained by words consisting such trigrams which can be found among signal values of HMM. The number of teaching nouns is 62. These words came from a webcorpus. The number of good inflections is 42, it means 67.6% precision value. It seems to be much better, but the teaching data contained all of the well inflected words. Thus the precision of inflection did not improve.

With initializing the state transition matrix the precision of inflection got better. In a web corpus the most frequent 100000 words are given. The state transition matrix was initialized by the trigrams of these words. It means approximately

15000 states. It is a bit lot, thus in testing the most frequent (occurs more than 100) 1800 trigrams is used. The number of good infections would be 49, it means that the precision increased with 11% and 7 new words could be inflected well.

In case of such a few teaching data the last column of the Viterbi matrix can contain more maximum value. The precision would be better when the most probable path was given as output ended with one of a state with maximum probability. The number of good infection increased to 53, it means 85% precision value, 4 newer words could be inflected.

In summary, the achievement is as follows: from 62 words 20 were untrained and at the end of refinement 11 of them have been inflected well, it means 55% precision value. The precision can be increased by estimations of signal matrix and naturally by the increasing of teaching data.

Regarding the new proposed method, the initial test yielded in the following output words:

$$R_{prop} = \{gatyát, labdát, tanárt, krumplit, fejt, korongot, csapatot, ködt, lót, koromet\}$$

This means a 60% precision value. The missed guesses are here more similar to the real values than in the other methods. The actual graph contains only 87 nodes.

It can be seen that the proposed method provides a promising performance. Thus in the project, the further and deeper investigation of the HMM and of the new method will continue.

References

[1] Browing, M.: Null Operator Constructions, Ph.D. thesis, MIT, 1987

[2] Chomsky, N.: Aspects of the Theory of Syntax, Cambridge, MIT Press, 1965

[3] Chomsky, N.: Formal properties of grammar, 1963

[4] Gildea, D., Jurafsky, D.: Automatic Induction of Finite State Transducer for Simple Phonological Rules, Meeting of ACL, 1995

[5] Harris, Z.: Methods in Structural Linguistics, University of Chicago Press, 1951

[6] Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing, MIT Press, 1999

[7] Shannon, C.: Prediction and Entropy of Printed English, Bell System Technical Journal, 1951

[8] Wallace, C.: Seneca Morphology, International Journal of American Linguistic, 1960

[9] Jurafsky D., Martin J. H.: An introduction to Speech Recognition, Computational Linguistics and Natural Language Processing, 2006

[10] Krenn, B., Samuelsson C.: The Linguistic's Guide to Statistics, 1997