# Cost Analysis of Classification Using CL and MM

**László Kovács, Péter Barabás**

Department of Information Technology, University of Miskolc, Hungary
{kovacs, barabas}@iit.uni-miskolc.hu

*Abstract: Computational linguistics covers the statistical and logical modeling of languages using computer-based software-hardware tools. An important component in CL systems is the morphological parser. The scope of our study is to build a statistical method to learn the rules of word inflection. The pre-requirement regarding the language is that the language uses words which are sequences of characters. A key factor of the required clustering algorithm is the cost efficiency. After analysis of the alternatives, two methods were selected to perform further refinement and adaptation: the observable Markov Model method and the formal concept analysis method.*

## 1 Classification Problem in CL

Computational linguistics (CL) covers the statistical and logical modeling of languages using computer-based software-hardware tools. An important component in CL systems is the morphological parser. A morpheme is the minimal unit with meaning in the language. 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 application context of the concept 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 [1].

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. The problem of living languages is that the dictionary describing the rules and exceptions is huge and not static. The building and updating these dictionaries is an expensive process. Our goal is to investigate the automated dictionary generation.

The scope of our study is to build a statistical method to learn the rules of word inflection. The pre-requirement regarding the language is that the language uses

words which are sequences of characters. In our language model, the words are built up from characters. During the inflection, a word is mapped to a word (a different word or the same word). The initial word is called stem. The grammar rule describes the generation of the inflected form from the stem.

It can be assumed that the transformation rules depend on the stem form of the concepts. There are distinct and relative few rules in the languages, i.e. the number of rules is much more less than the number of stems. Thus, in our approach, the rule assignment task is considered as a classification method, each stem is assigned to a rule class. The set of possible rule classes is extracted from a training data set. The investigation is based on the following items:

*$W$ : set of words*

*$T = \{(w,w') | w,w' \in W\}$ : a training set, where $w$ is a stem and $w'$ is the transformed form*

*$R = \{r \mid r : W \rightarrow W\}$ : the set of transformation rules, where $r(w)$ results in the transformed form for a $w$ stem.*

*$G : W \rightarrow R$ : the grammar which is treated as a rule classification function.*

The task of rule learning consists of the following sub-problems:

- extraction of transformation rule from the training set
- determining of association rule between the stem and rule class

One of the main requirements on the method is the efficiency. As the set of words $W$ can be very large, the learning algorithm should have a polynomial cost function of low grade. In our approach, the low cost should be achieved through the following steps:

- providing an efficient generalization
- allowing approximation in the rule system

For this purpose, the statistical language processing is an appropriate solution. Considering the classification methods used in data mining, the following alternatives can be used: decision trees, Bayes-methods and ANN methods. On the other hand, our problem area has some special characteristics that are not supported by the classical classification methods, like

- large number of attributes
- incremental learning method
- support of early approximation

Considering the main statistical methods to build morphological parsers are the Hidden Markov Models (HMM), the N-gram models (NGM) and the Finite State

Transducers (FST). The drawback of these algorithms is that these are not intended to perform classification. They are usually used to learn the direct transformation steps. Thus new algorithms for learning the rule classes had to be developed and to be tested. After analysis of the alternatives, two methods were selected to perform further refinement and adaptation: the observable Markov Model method and the formal concept analysis method.

# 2 Classification with Markov Model

## 2.1 N-grams and Markov Models

In statistical language processing N-gram model is widely used. 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 where words have to be identified from noisy, ambiguous input. Our goal is to compute the probability of a word *w* given some *h history,* or *P(w|h)*. A sequence of *N* words is represented as follows [2]:

$$w_1,...,w_n \text{ or } w_1^n \tag{1}$$

The probabilities of entire sequences like $P(w_1,...,w_n)$ can be computed with using the chain rule of probability for decomposition [1]:

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

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 the way to compute the probability of a word a long sequence of preceding words is not known. Estimation by counting word occurrences following a long sequence of words is not the correct way, 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 we can approximate the probability of a word by just a few preceding words.

The bigram model is such an N-gram model where the probability of a word is approximated by the preceding word [2]:

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1}) \tag{3}$$

whereas in trigram model this probability is:

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-2}^{n-1}). \tag{4}$$

In general we can make the following approximation:

$$P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-N+1}^{n-1}\right) \tag{5}$$

This property that the probability of a word depends only on the previous word is called Markov property stated as [3]:

$$P\left(\xi_{t+1} = s_{i_{t+1}} \mid \xi_1 = s_{i_1},\ldots,\xi_t = s_{i_t}\right) = P\left(\xi_{t+1} = s_{i_{t+1}} \mid \xi_t = s_{i_t}\right) \tag{6}$$

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

In a Markov chain we can attribute each state 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 $\eta_t$, $t=1\ldots T$, which is the emitted signal in time t. This determines a Markov model.

In Markov models we can define [3]:

- 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 point of time a state transition occurs depending on the transition probabilities and after each transition, one of the signals associated with the current state is emitted. In general a state in the model depends on the previous state of the model, called first-order Markov model. When a state depends on more preceding states these are called as higher-order Markov models, such as 2nd-order, ... Nth-order Markov model. Bigram model is a first-order Markov model, whereas trigram model is a second-order and n-gram model is a n-1th-order Markov model.

## 2.2 Algorithm for Classification

The rule detection can be defined as a classification method where words which are inflected in the same way belong to the same class. The classes are not predefined but they are extracted from the training samples. The training samples consist of word pairs such as a stem and its well transformed inflected form. A class can be signed as

$$C_{x,y} = \left\{(w,w') \mid \exists w_o \in W : w = w_0.x, w' = w_0.y\right\} \tag{7}$$

The number of classes depends on the training samples. For each class an observable Markov Model is generated. The states of the model are the unigrams,

bigrams, trigrams, etc. of the stems. To choose a correct-order Markov Model is an optimization problem. The higher-order is the model the more accurate can be the result and the more costly is the algorithm. E.g. in case of a 1st-order MM the number of states is maximum 35, in case of a 2nd-order MM it is $35^2$, in case of a Nth-order MM it is $35^N$. The number of cells of the transition matrix is the square of the states whereby can be seen that the cost of count depends on the good choice of the MM.

The training algorithm is the following:

1 get the first pair from the samples

2 determine the class from the word pair

3 if the class is not exist establish the class and the model

4 add the stem to the model of the current class

   a) break the word into n-grams (n is a predefined constant)

   b) add the n-grams as states to the model

   c) recalculate values in the transition matrix

5 get the next pair from the samples and go to step 2

The numbers of states of the models are varying because it depends on the training samples like the number of classes. It is not a difficult algorithm accordingly the cost of the training is much more lower than in case of more difficult nets or in FCA, but depends on the order of model.

In testing phase a stem is given as input to the classifier and a class as the inflection rule is resulted as output. The classifier works with observable Markov models since the output of the process is the set of states at each instant of time, where each state corresponds to a physical (observable) event [4]. The stem have to be broken into n-grams which will be the observation sequence O as O = $\{w_1,w_2,w_3,\ldots w_n\}$. We wish to determine the probability of O, given the models. We have a model for each class. In first approach we can count the $P(O|model_i)$ probability for all class.

$$P(O|model_i) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \ldots \cdot P(w_n|w_{n-1}) \quad (8)$$

The maximum probability belongs to the winner model:

$$P_{max} = max\ \{P(O|model_i\} \quad (9)$$

The algorithm can be refined with eliminating classes which does not fit to the testing stem ($w_T$). The testing classes are the follows:

$$C^T = \left\{C_{x,y} \mid \exists w_0 \in W : w_T = w_0.x\right\} \quad (10)$$

### 2.3 Results

Summarized can be said that the teaching phase is a simple phase, it can be refined in the future. Our experiences show that the error rate of inflection of training words is under 5%. The error rate of inflection of non-trained words is higher and its main reason is the count of observation probability. If any of the factors of probability is equal to zero then the probability will be 0. It can occur often because any of the n-grams of untrained words can be missing of the states. One of the solutions can be that the factor of the missing transition will be a small number instead of zero.

In our test application the classifier has been trained with 4100 pairs of words. For testing we choose 100 words randomly. The following table summarizes the results:

| **N-grams** | **Accuracy** | **Time cost** | |
|---|---|---|---|
| | | **Training** | **Testing (per word)** |
| Unigrams | 90%-95% | 0.14s | 0.016s |
| Bigrams | 95%-100% | 1s | 0.5s |
| Trigrams | 97%-100% | 11.3s | 10s |

## 3 Classification with Concept Lattice

In a wide area of soft computing, the methods of formal concept analysis (FCA) are increasingly used to discover the object clusters and the generalization relationships inherent in the corresponding attributes. Wille [5] proposed first to consider the elements in a Galois lattice as concepts. The concept lattice is based on a binary relationship between the objects and attributes. The edges of the lattice represent the generalization relationship. The FCA is used nowadays as a powerful tool in a wide variety of soft computing areas among others for data analysis, information retrieval, knowledge discovery and software engineering.

A *K context* in FCA is a triple *K(G,M,I)* where *G* and *M* are sets and *I* is a relation between *G* and *M*. The *G* is called the set of *objects* and *M* is the set of *attributes*. The cross table *T* of a context *K(G,M,I)* is a matrix form description of the *relation I*. For every $O \subseteq G$, and for every $A \subseteq M$ a *derivation operator* is defined:

$$\begin{aligned} O' &= \{ a \in M \mid g\, I\, a \text{ for } \forall\, g \in O \} \\ A' &= \{ g \in G \mid g\, I\, a \text{ for } \forall\, a \in A \} \end{aligned} \tag{11}$$

The pair *C(O,A)* is a *concept* of the *K* context if

$$O' = A,\ A' = O \tag{12}$$

are satisfied. In this case, the $O$ is called the *extent* and $A$ is the *intent* of the $C$ concept. Considering the $\Phi$ set of all concepts for the $K$ context, an *ordering relation* can be introduced for the concept set in the following way:

$$C_1 \leq C_2$$

if (13)

$$O_1 \subseteq O_2$$

where $C_1$ and $C_2$ are arbitrary concepts. Based on this ordering, the $(\Phi, \leq)$ is a lattice, called *concept lattice*. According to the Basic Theorem of concept lattices, $(\Phi, \leq)$ is a complete lattice, i.e. the infimum and supremum exist for every set of concepts. The following rules hold true for every family $(O_i, A_i)$, $i \in I$ of concepts:

$$\begin{aligned} &\vee_{i \in I} (O_i, A_i) = (\cap_{i \in I} O_i, (\cup_{i \in I} A_i)''), \\ &\wedge_{i \in I} (O_i, A_i) = ((\cup_{i \in I} O_i)'', \cap_{i \in I} A_i) \end{aligned} \tag{14}$$

where $O''$ denotes the *closure* of the set $O$ and it is defined as derivation of the derivated set.

One of the first proposals to apply a concept lattice for classification problem is given in [7]. In this model, one of the attributes is marked as class label. The main goal of the classification algorithm is the assignment of a class label to the objects based on the object's attribute values. A classification rule describes the dependency of the class label from the attribute values. The goodness of a rule is measured with the confidence and generality properties, where

$$\begin{aligned} &\text{conf}(A \Rightarrow c) = |(A \wedge (a_c=c))'| \ / \ |A'| \\ &\text{generality}(A \Rightarrow c) = |A'|/|G| \end{aligned} \tag{15}$$

The higher is the confidence value the more accurate is the classification rule. A consistent classification rule is a classification rule with a confidence value 1, i.e.

$$|(A \wedge (a_c=c))'| = |A'| \tag{16}$$

A concept $(O,A)$ is called consistent concept if it implies a unique class label and the confidence value is equal to 1. The most general consistent concept is a consistent concept where neither of the super concepts is consistent. It can be shown that the set of most general consistent concepts is covering $G$ the universe. Thus the set of most general consistent concepts is enough to perform the classification process.

## 3.1 Algorithm for Lattice Building

Our investigation is focused on the incremental lattice building. The best known incremental method is the proposal of Godin [6]. The building of the concept set is based on the following rule: every new concept-intent after inserting a new object

will be the result of intersecting the attribute set of the new object with some intent set already present in the concept set. The algorithm consists of the following main steps. First, the concepts are partitioned into buckets based on the cardinality. Next, the buckets are processed in ascending cardinality order. Every intents in the current bucket are intersected with the intent set of the new object. If the result set is not present in the concept set, it will be added. The cost estimation for the algorithm can be given by

$$O(N\sigma + CNDM), \tag{17}$$

where

- $N$ : number of objects,
- $M$ : number of attributes,
- $C$ : number of concepts,
- $D$ : number of candidate parent concepts.

The Godin's method does not filter out the consistent concepts. Only few proposals were given to determine the most general consistent concepts. The PRISM [8] algorithm is one of these methods using a relative simple and heuristic algorithm. The algorithm uses a simple loop to verify all attribute set - class assignment to calculate the confidence and generality values.

In our investigation, the base Godin's method was extended in order to perform an efficient classification. The method extensions are based on the following considerations:

- cost reduction of concept intersection can be achieved with application of lattice-based candidate generation (instead of full scan)
- bucket-based implementation of candidate verification during the parent generation phase
- applying a lexical ordering in the context generation
- attribute level processing of concepts
- introduction of exception concepts
- elimination of included concepts
- reduction of the lattice into a decision tree format
- introduction of generalized attributes (attribute lattice)

All of these steps are tailored to the problem areas, namely to processing of a large set ($w$,$w'$) pairs. According to our and other experiments (for example [Godin]), the time cost of base lattice building algorithm is

$$O(N^{\alpha}) \tag{18}$$

where the $\alpha$ factor lies between 2 and 3, depending among others on the |M| value. In our test environment, the base algorithms could be used only for some thousands training words. The proposed algorithm reduces the cost factor to a value between 1 and 2. In the following table, the base CL building method, the Godin method and the proposed method are compared.

| N (words) | C (lattice size) | Base method | Godin method | proposed m. |
|---|---|---|---|---|
| 100 | 530 | 0.6 | 0.7 | 0.4 |
| 500 | 4000 | 38.0 | 12.1 | 2.4 |
| 1000 | 9300 | 143.1 | 60.4 | 9.0 |
| 1500 | 16790 | 334.3 | 154.4 | 17.3 |
| 2000 | 28500 | 655.4 | 345.2 | 31.4 |
| 2500 | 40500 | 1241.2 | 608.4 | 51.2 |
| 3000 | 53000 | 1871.3 | 933.2 | 74.3 |

The test result show high efficiency of the proposed method. These tests were executed with algorithm without size reduction. In the next version of the algorithm, a size limitation module is implemented where the nodes with low importance factor are eliminated. The base elements of the importance factor are:

- frequency of usage
- homogeneity
- time of last usage
- compactness

These steps can yield in constant lattice size and cost value.

**References**

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

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

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

[4] Lawrence R. Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, 1990, pp. 267-296

[5] Wille R.: Restructuring Lattice Theory: an Approach Based on Hierarchies of Concepts, Reidel, 1992

[6] Godin R., Missaoui R., Alaoui H.: Incremental Concept Formation Algorithms Based on Galois Lattices, Computational Intelligence, pp. 246-267, 1995

[7] Ganter B., Wille R.: Formal Concept Analysis: Mathematical Foundations, Springer Verlag, 1999

[8] Zhao Y., YAo Y.: Classification Based on Logical Concept Analysis, 2006