STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 1, 2000

TRAINING PROBABILISTIC CONTEXT-FREE GRAMMARS AS HIDDEN MARKOV MODELS

ADRIAN DUDA, GABRIELA ŞERBAN, DOINA TĂTAR

Abstract. It is considerred in this moment that the use of mathematical statistics methods in natural language processing represents a leading topic in NLP. Statistical methods have first been applied in the "speech-recognition" area. While Hidden Markov Model (HMM) is unanimously accepted as a mathematical tool in this area, its advantages have been less used in dealing with understanding natural language. In this paper we propose a method for association of a HMM to a context-free grammar (CFG). In this way, learning a CFG with a correct parsing tree will be realized by learning a HMM.

Key words: probabilistic contex-free languages, hidden Markov models, natural language processing.

1. Hidden Markov Model (HMM).

HMM model is a generalization of Markov chains, being possible that more arrows to go out for a given input. As in an HMM we can have more paths coverred for the same input, it implies that $P(w_{1,n})$ (which is the probability to have as input a sequence made up of n words, $w_1\, w_2 \cdots w_n$, shortly written as $w_{1,n}$) is calculated as the sum of the probabilities on all the possible paths. Probability on a given path is calculated by multiplying the probabilities on each segment (arrow) of that path.

Definition

An HMM is a 4-element structure $< s^1, S, W, E >$, where S is a (finite) set of states, $s^1 \in S$ is the initial state, W is a set of input symbols (words), and E is a set of transitions (labelled arrows). We consider the following order of the elements of the sets S, W, E: $S = (s^1, \cdots s^\sigma)$; $W = (w^1, \cdots w^\omega)$;

$E = (e^1, \cdots e^\epsilon)$.

Let us notice the difference between w_i and w^i: the first one means the i-th element (word) of an input sequence, while the second one is the i-th element of the W set. A transition is defined as a 4-element structure: (s^i, s^j, w^k, p) ,

2000 *Mathematics Subject Classification.* 68Q42, 65C40.

1998 *CR Categories and Descriptors.* F.4.2 [**Theory of computation**]: Mathematical Logic and Formal languages – *Grammars and other rewriting systems*; G.3 [**Mathematics of Computing Probability and Statistics**]: Markov processes.

representing passing from state s^i to state s^j for input w^k, transition evaluated as having the probability p. As for a given input sequence we have more possible paths, the states that it has been passed through is not deductible from input, but *hidden* (this gives the name of the model we focus on). The sequence of states $s_1, s_2, \cdots, s_{n+1}$ that it has been passed through for an input $w_{1,n}$ is marked by us shortly with $s_{1,n+1}$.

Algorithm to find the highest-probability-path. In the followings, we are using *Viterbi* 's algorithm to find the most probable path. Formally written, we have to find

$$argmax_{s_{1,n+1}} P(w_{1,n}, s_{1,n+1})$$

where $w_{1,n}$ is a sequence of input(entrance) words, and $s_{1,n+1}$ is the set of states that has been passed through. The main idea of the algorithm is calculating the most probable path beginning with the empty input sequence, and processing one word at a time, then the next word that comes in the input sequence. At every step, we calculate the most probable sequence of states which ends up with the state s^i, $i = 1, \cdots, \sigma$, where σ is the total number of states of the Markov model. Formally, we denote: $\sigma_i(t+1)$ *is the most probable sequence of states when it was given as input the sequence of words* $w_{1,t}$ *and the final state being* s^i. The highest-probability-path we are looking for is

$$\sigma_i(n+1) = argmax_{s_{1,t+1}} P(w_{1,t}, s_{1,t}, s_{t+1} = s^i)$$

and has as final state s^i. Dynamic programming principle, that is fundamental for Viterbi's algorithm, allows us to make the following remarque: the highest-probability-path to a state s^i, when it is given as input the sequence $w_{1,t}$, is made up of the maximum-probability-path with the input $w_{1,t-1}$, with the final state (let us note) s^k , which is making the multiplication $P(\sigma_k(t))P(s^k \stackrel{w_t}{\to} s^i)$ as maximal, and juxtaposing this so-obtained path with the state s^i. Saying the above in another way, $\sigma_i(t+1)$ is calculated like this:

$$\sigma_i(1) = s^i, i = 1 \cdots \sigma \;\; \sigma_i(t+1) = \sigma_j(t) \circ s^i \;, j = argmax_{k=1,\sigma}(P(\sigma_k(t))P(s^k \stackrel{w_t}{\to} s^i)).$$

In the above formula, "$\circ$" represents concatenation.

Algorithm to calculate the probability of an input sequence. We are mentioning here two algorithms to calculate the probability of an input sequence. Let us note by $\alpha_i(t+1)$ the probability that the input sequence $w_{1,t}$ be accepted, and having s^i as the final state. In other words:

$$\alpha_i(t+1) = P(w_{1,t}, s_{t+1} = s^i), t > 0 \qquad (1)$$

Let us notice that having all $\alpha_i(n+1)$ values calculated, the probability $P(w_{1,n})$ is given by:

$$P(w_{1,n}) = \sum_{i=1}^{n} \alpha_i(n+1).$$

Considering that $w_{1,0}$ is the empty word, which has the acceptance probability 1, we have that $\alpha_j(1) = 1., if\ j = 1, and\ it\ is\ 0\ otherwise$, corresponding to the fact that the initial state of every path is s^1. Calculation of $\alpha_j(t)$ is done starting with $\alpha_j(1)$, $\alpha_j(2)$ and going until $\alpha_j(n+1)$, using the recursive relation:

$$\alpha_j(t+1) = \sum_{i=1}^{\sigma} \alpha_i(t) P(s^i \stackrel{w_t}{\rightarrow} s^j).$$

The probabilities $\alpha_i(t)$ are called *forward* probabilities. It is also possible to calculate *backwards* probabilities, $\beta_i(t)$, with the following definition: $\beta_i(t)$ represents the acceptance-probability of input $w_{t,n}$, if the state at step t is s^i. So:

$$\beta_i(t) = P(w_{t,n} \mid s_t = s^i), t > 1.$$

The probability we are looking for will be

$$\beta_1(1) = P(w_{1,n} \mid s_1 = s^1) = P(w_{1,n})$$

Calculation of β function is done starting with values:

$$\beta_i(n+1) = P(\epsilon \mid s_{n+1} = s^i) = 1, i = 1, \cdots, \sigma.$$

For the recursive case, we have:

$$\beta_i(t-1) = P(w_{t-1,n} \mid s_{t-1} = s^i) == \sum_{j=1}^{\sigma} P(s^i \stackrel{w_{t-1}}{\rightarrow} s^j)\beta_j(t)$$

Training Markov models. The training algorithm of a Markov model used in this paper is the Baum-Welch algorithm (or forward-backward). This one, having given a certain *training* input sequence, it ajusts the probabilities of transitions in the HMM, so that the respective sequence have an as big as possible acceptance probability. Application of the algorithm has as prerequisite an HMM structure already having been defined, and only the probabilities of transitions still remaining to be established. The probabilities of transitions are calculated with the formula [2]

$$P(s^i \stackrel{w^k}{\rightarrow} s^j) = \frac{C(s^i \stackrel{w^k}{\rightarrow} s^j)}{\sum_{l=1,m=1}^{\sigma,\omega} C(s^i \stackrel{w^m}{\rightarrow} s^l)} \qquad (2)$$

The C function in the above formula is calculated like this [2]:

$$C(s^i \stackrel{w^k}{\rightarrow} s^j) = \frac{1}{P(w_{1,n})} \sum_{t=1}^{n} \alpha_i(t) P(s^i \stackrel{w^k}{\rightarrow} s^j) \beta_j(t+1) \qquad (3).$$

What can be immediately noticed in this formula is that, for the calculation of $C(s^i \stackrel{w^k}{\rightarrow} s^j)$ we need to know path-probabilities, and so, the probabilities of transitions for the HMM model. Therefore, we start with some 'guessed' probabilities, calculated with the help of the formula (3) the new values of function

$C(s^i \overset{w^k}{\to} s^j)$ and then we adjust the probabilities of transitions using the formula (2). The indicator showing the improvement level of probabilities is the growth of the probability of input sequence $P(w_{1,n})$ compared to the previous estimation. The process of recalculating transition probabilities is finished when these probabilities no more suffers modifications considered as important.

2. Probabilistic context-free grammars.

Definition [3, 2]
A probabilistic context-free grammar (PCFG) is a 5-element structure $< W, N, N^1, R, P >$ where

- $W = \{w^1, \ldots, w^\omega\}$ represents a set of terminal symbols (we also call them words);
- $N = \{N^1, \cdots, N^\nu\}$ represents a set of non-terminal symbols, and N^1 is the initial(start) symbol S;
- R is a set of rules of form $N^i \to \xi^j$, where $\xi^j \in (N \cup W)^*$;
- P is a probability function associating to every rule $N^i \to \xi^j$ a probability $P(N^i \to \xi^j)$ so that the sum of probabilities of the rules having as left-side member (deploying) the same non-terminal is 1.

The probability of a sequence $w_{1,n}$ is equal with to sum of the probabilities of all possible syntactical trees for the analysis of $w_{1,n}$. The probability of a tree is given by the multiplication of the probabilities of the used rules : $P(T) = \prod_{rule\ r\ used\ in\ T} P(r)$.

2.1. Associating an HMM model to a PCFG grammar.

In the followings, we are providing a way of associating an HMM model to a PCFG grammar, so that each derivation tree corespond to a path in HMM. This allows us to calculate the probability of a sequence as the probability of an accepted sequence by an HMM. In order to describe the algorithm of attaching a HMM to a PCFG, we suppose that the PCFG is in Chomsky-normal-form, that is, all the rules have the form:

$$X \to Y\ Z \ or \ X \to a$$

where X, Y, Z are non-terminals, and a is terminal. There are three possible situations to be discussed:

Case I. A rule has the form $p_X : X \to Y\ Z$ and there exist the rules $p_Y : Y \to a$ and $p_Z : Y \to b$. A derivation tree using these rules looks like that given in the figure 1.

The corresponding path in an HMM is shown in the figure 2.

Case II. A rule has the form $p_X : X \to Y\ Z$, and there exist the rules $p_Y : Y \to U\ V$, $p_U : U \to a$, $p_V : V \to b$, $p_Z : Z \to c$. A derivation tree using these rules looks like in the figure 3.

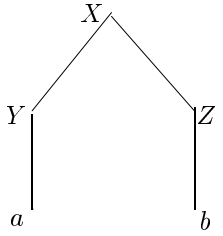

Fig. 1

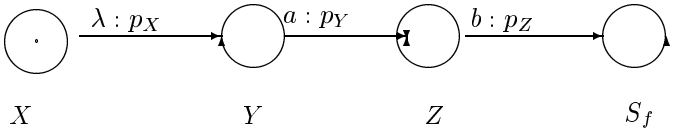

Fig. 2

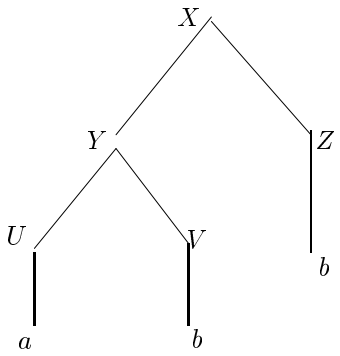

Fig. 3

The corresponding path in an HMM is shown in the figure 4.

Case III. The rule has the form $p_X : X \to Y\,Z$, and there exist the rules $p_Z : Z \to U\,V$, $p_Y : Y \to a$, $p_U : U \to b$, $p_V : V \to c$. A derivation tree using these rules looks like in figure 5.

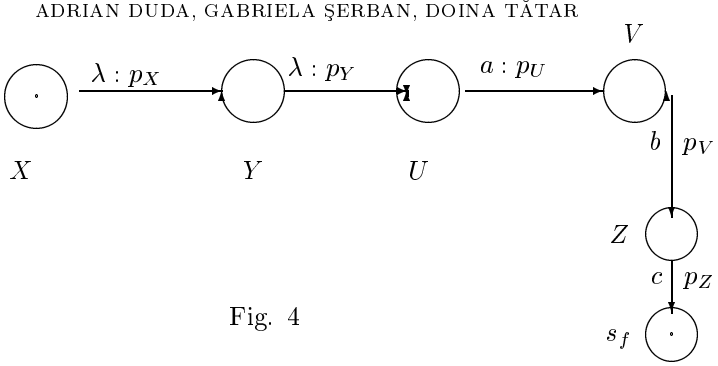

Fig. 4

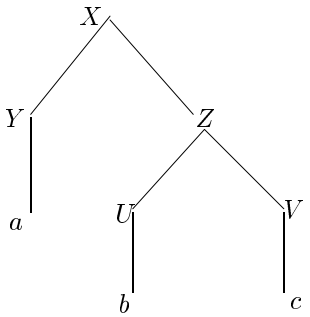

Fig. 5

The corresponding path in an HMM is shown in the figure 6.

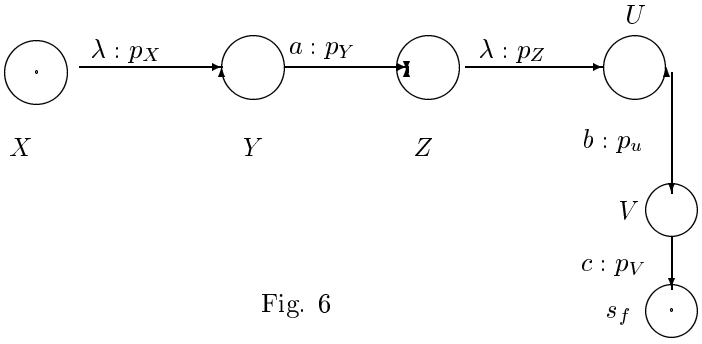

Fig. 6

Definition

A sequence $w_{1,n} \in L(G)$, where G is a PCFG, with the probability $P(w_{1,n})$, if there is a parsing tree T with the root S and the product of rules used for T is $P(w_{1,n})$.

Definition

A sequence $w_{1,n}$ is accepted by a HMM H with the probability $P(w_{1,n})$ if there is a path from S (the start node of HMM) to the final node s_{fin} and the product of probabilities on edges is $P(w_{1,n})$.

Theorem

If G is a probabilistic context-free grammar, H is the HMM associated with G as above and the sequence $w_{1,n} \in L(G)$ with the probability $P(w_{1,n})$, then $w_{1,n}$ is accepted by H.

Proof Let us consider that $w_{1,n} \in G$, where G is in Chomsky normal form. We will prove by induction on the length m of the longest path in the parsing tree of $w_{1,n}$ that $w_{1,n}$ is accepted by H. If the length m is 1, then $w_{1,n} = a$ and in H there is a path (as in Case IV above) from S to s_{fin} labeled by a , of the same probability. We will suppose that the implication is true for each sequence obtained by a parsing tree with the longest path $m-1$ and let as suppose that the sequence $w_{1,n}$ is obtained by a parsing tree T with the longest path m. In this parsing tree the first rule used is of the form $p_S : S \to Y\ Z$. In the tree T Y and Z are roots of parsing tree T_1 and T_2, with the longest path at most $m-1$ and with the frontiers P_1 and P_2. The frontier of T is $w_{1,n}$ so $w_{1,n} = P_1P_2$. Consider that we are in the Case I (the others cases are proved analogously). In this case P_1 is $a = w_1$ and P_2 is $w_{2,n}$. By induction hypothesis P_2 is accepted by a HMM with the start symbol Z. The situation in Case I is as in figure 7.

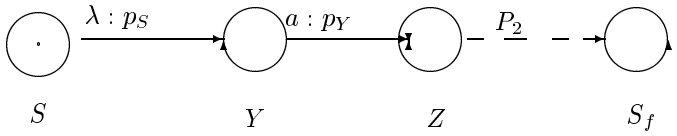

Fig. 7

So, H accepts $a\,P_2 = w_{1,n}$. The probability $P(w_{1,n})$ in T is obtained as the product between p_S , the probability of P_1, and the probability of P_2. In the Case I the probability is: $p_S \times p_Y \times P_2$.

3. Training PCFG - grammars.

The training of the PCFG-grammars is obtained based on the training algorithm for HMM, by passing from a grammar to an HMM, as in the above mentioned theorem. As we have said, Baum-Velch algorithm for training an HMM needs

a given structure to be applied to. Just the same, for a PCFG, it is supposed that the rules have already been defined. Let us consider the phrases used for the training process as *"parenthesis-ed"*, which means, it is defined the way to obtain items from lower-level items (closer to the border of the derivation tree). In order to exemplify, look at the phrase: *" Salespeople sold the dog biscuits"* [2]. Let us now describe the needed steps when training a PCFG, and use the above phrase for better understanding. Parenthesis-ing *(Salespeople (sold (the dog biscuits)))*, generates the folowing rules: $s \to np\,vp$; $np \to noun$; $np \to det\ noun\ noun$; $vp \to verb\,np$. A second possible parenthesis-ing (incorrect) is: *(Salespeople (sold (the dog) biscuits))*. According to this, we have the rules: $s \to np\,vp$; $np \to noun$; $np \to det\ noun$;$vp \to verb\,np\ np$. The overall set of rules we have obtained is shown below, where the sum of probabilities of the rules having the same non-terminal in the left-side member is $s \to np\,vp$: 1.0; $np \to noun$: 0.5; $np \to det\ noun\ noun$: 0.25; $np \to det\ noun$: 0.25; $vp \to verb\,np$. 0.5; $vp \to verb\,np\ np$: 0.5. We are transforming the so-obtained rules to be in Chomsky-normal-form, starting with the rule $np \to noun$: 0.5 and in the next step we are modifying the rules containing more than two non-terminals in their right-side member. After these being done, the final grammar becomes: $s \to np\,vp$: 0.50; $s \to noun\,vp$: 0.50; $np \to det - n\ noun$: 0.50; $np \to det\ noun$: 0.50; $vp \to verb\,np$. 0.20; $vp \to verb\,noun$. 0.20; $vp \to verb - np\ np$: 0.20; $vp \to verb - np\ noun$: 0.20; $vp \to verb - n\ noun$: 0.20; $det - n \to det\ noun$: 1.0; $v - np \to verb\,np$: 1.0; $v - n \to verb\,noun$: 1.0; $noun \to salespeople$: 0.35; $noun \to biscuits$: 0.35; $noun \to dog$: 0.40; $verb \to sold$: 1.0; $det \to the$: 1.0. Let us now consider the first (correct) parenthesis-ing of the phrase: *(Salespeople (sold (the dog biscuits)))*. The derivation-tree of figure 8 corresponds to this situation.

The path in the HMM is given in figure 9.

As far as concerns the second (incorrect) parenthesis-ing, its corresponding derivation-tree looks like in figure 10.

Initially, both of the derivation trees have the same probability (0.00245). After the grammar has been trained with the correctly paranthesised phrase, the correct tree has the probability 0.037037 while the incorrect one has the probability 0.000.

4. The application.

The application is written in Borland Pascal. The application has three parts:

- the first part(A) reads a HMM from a text file
- the second part(B), having as input a HMM and a given entry sequence, finds the probability and also the most probable paths for the entry sequence
- the third part(C), executes the training of the given HMM, for a given entry sequence.

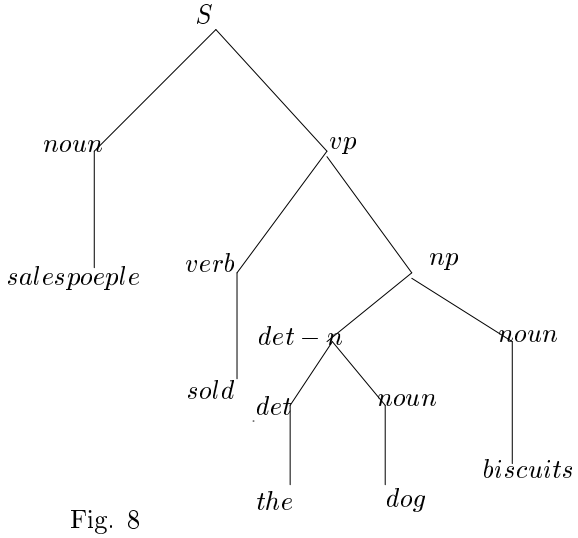

Fig. 8

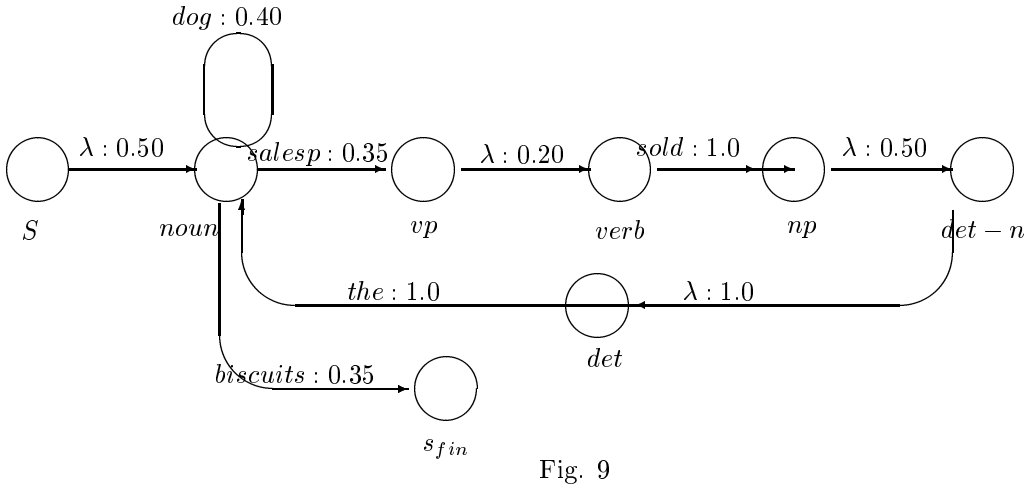

Fig. 9

The algorithms used in the second and the third part of the application are described above.

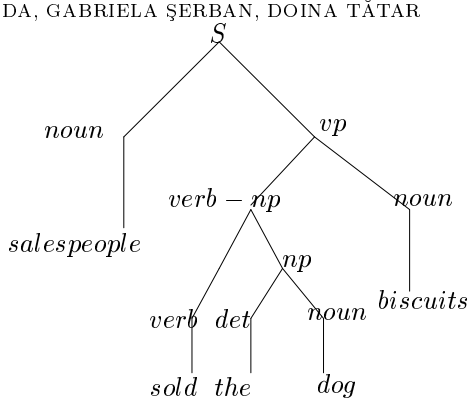

Fig. 10

The application will be sent at request by the second author. In the following we will describe shortly this application.

The input data are read from a text file, which contains the given HMM, in fact **the number of states, the set of states, the initial state** and **the set of transitions**. We have to specify that the number of entries and the set of the entries is not read from the input file, but is automatically calculated from the set of transitions. We assume that each transition is identified by three components $s1, p, s2$, where s1 and s2 are states, and p is the probability of the transition from s1 to s2 and also, a state is identified by a character (of course, this assumption is not restrictive, if is necessary, a state could be identified by a string).

Constants.

•$MaxNrStari = 25$
- the maximum number of states
•$MaxNrIntrari = 15$
- the maximum number of entries
•$MaxNrDrum = 10$
- the maximum number of paths

Data types.

```
sir=array[1..MaxNrStari] of char
```

- defines the type of the set of states of the HMM (each state is represented as a character)

```
tranzitii=array[1..MaxNrStari,1..MaxNrIntrari,1..MaxNrStari] of real
```

- defines the type of the set of transitions (the structure of a transition was described above)

`mat=array[1..MaxNrStari,1..MaxNrIntrari] of real`

- defines the type of a matrix with $MaxNrStari$ lines and $MaxNrIntrari$ columns, for representing the data type of the probabilities $\alpha_i(t)$

`sirs=array[1..MaxNrDrum] of string;`

- represents the type corresponding to the array of paths in the HMM (a path is represented as a *string* - an array of characters).

Global variables.

•s - a variable of type sir; represents the set (array) of states
•w - a variable of type sir; represents the set (array) of entries
•p - a variable of type $tranzitii$; represents the set (array) of transitions
•$sigma$- a variable of type $integer$; represents the number of states
•ni - a variable of type $integer$; represents the number of entries
•si - a variable of type $integer$; represents the index of the initial state in the set of states
•$alfa$ - a variable of type mat; represents the matrix containing as elements the probabilities $\alpha_i(t)$ (for a given entry sequence)
•$beta$ - a variable of type mat; represents the matrix containing as elements the probabilities $\beta_i(t)$ (for a given entry sequence)
•y - a variable of type $string$; represents an entry sequence (we assume that the length of this sequence is less or equal than $MaxNrIntrari$)

The algorithm performs the following steps:

Part A

• reads the input data(the HMM) from the text file.

Part B

• reads an entry sequence
• determines for the given entry y, the probabilities $\alpha_i(t)$ and $\beta_i(t)$ (for all $i \in [1..sigma]$ and $t \in [1..length(y)+1]$
• using α and β (calculated at the preceding step), on determine the probability of the entry sequence y
• determines and displays the most probable paths for the entry sequence

Part C

• reads the training entry sequence
• trains the HMM for the entry sequence, using the Baum-Welch algorithm

Subprograms used.

Part A

(P) $procedure\, citire(var\, sigma : integer; var\, s, w : sir; var\, p : tranzitii; var\, si : integer; var\, ni : integer)$

- reads the input data (the number of states, the set of states, the set of entries, the set of transitions, the initial state, the number of entries) from a text file

Part B

(F) $function\, apare(x : string; s : sir; ns : integer) : integer$

- determines the index of the string x in the array s having the dimension ns

(F) $function\, alfa\_j\_tplus1(alfa : mat; j, t : integer; y : string) : real$

- calculates $\alpha_j(t+1)$ for the entry sequence y

(F) $function\, beta\_i\_tminus1(alfa, beta : mat; i, t : integer; y : string) : real$

- calculates $\beta_i(t-1)$ for the entry sequence y

(P) $procedure\, calcul\_alfa\_beta(var\, alfa, beta : mat; y : string)$

- using the two above described functions, calculates the probabilities $\alpha_i(t)$ and $\beta_i(t)$ (for all $i \in [1..sigma]$ and $t \in [1..length(y)+1]$

(P) $procedure\, det\_prob\_intrare(y : string; var\, pro : real)$

- calculates the probability p of the entry sequence y, as the sum of the elements from the last column (length(y)+1) of the matrix $alfa$

(P) $procedure\, det\_drum\_cel\_mai\_probabil(y : string; var\, n : integer; var\, z : sirs; var\, max : real)$

- determines the most probable paths for the entry sequence y, each path having the probability max (n represent the number of paths, z represent the array of the most probable paths)

(P) $procedure\, afisare\_drum\_cel\_mai\_probabil(y : string; n : integer; z : sirs; pro : real)$

- displays the most probable paths for the entry sequence y (the paths retained by the above described procedure)

Part C.

(F) $function\, calcul\_c\_i\_k\_j(i, k, j : integer; y : string) : real$

- calculates the value of the numbering function C for the states i, j and the transition y[k] (y is the entry sequence), using the relation (3) given in subsection 1.1; this function uses the values $\alpha_i(t)$ and $\beta_i(t)$ calculated in part B

(P) $procedure\, antrenare\_hmm$

- trains the HMM using a training entry sequence and the Baum-Welch algorithm

Examples.

Part B.

Let us consider the following input file

3 - the number of states
s - the first state
b - the second state
f - the third state
s - the initial state
s 0 s 0.05 - the following lines contain the transitions
s 1 s 0.05
s 0 b 0.9
b 1 s 0.3
b 0 s 0.5
s 1 f 0.1
b 0 f 0.1
b 1 f 0.1

If the entry sequence is *001*, then the results are
• the probability of the entry sequence is *0.0859*
• the probability of the most probable path for the entry sequence is *0.0450*
• the most probable path for the entry sequence is $sbsf$

Part C.

Let us consider the following input file, which codifies the HMM described in section 3.

9
s - the state "S"
a - the state "noun"
b - the state "vp"
c - the state "verb-np"
d - the state "verb"
e - the state "np"
h - the state "det-n"
g - the state "det"
f - the final state "s-fin"
s - the initial state
s l a 0.5 - the transitions
a d a 0.4 - "d" codifies "dog"
a S b 0.35 - "S" codifies "Salespeople"
a b f 0.35 - "b" codifies "biscuits"
b l c 0.2 - "l" codifies "λ"
b l d 0.2
c l d 1.0
d s e 1.0 - "s" codifies "sold"
e l h 0.5

e l g 0.5
h l g 1.0
g t a 1.0 - "t" codifies "the"
a S f 0.35
d s f 1.0
a d f 0.4
s l e 0.5
a d b 0.40

The sequence which codifies the correct sentence *(Salespeople(sold(the dog biscuits)))* (1) is *lSlsslltdb*.

The sequence which codifies the incorrect sentence *(Salespeople(sold(the dog) biscuits))* (2) is *lSllsltdb*.

The sequence which codifies the incorrect sentence *(The dog(sold(the dog biscuits)))* (3) is *lltdlsslltdb*.

The results obtained for this HMM are the following

• in the given HMM, without training, the sentences 1 and 2 have the same probability *0.00245* and the sentence (3) has the probability *0.0014*

• after training the HMM for the first sentence (1), the probability for the sentence 1 is *0.037037*, the probability for the sentence 2 is *0.00* and the probability for the sequence (3) is *0.00*

References

[1] J.Allen : " Natural language understanding", Benjamin/Cummings Publ. , 2nd ed., 1995.
[2] E. Charniak: "Statistical language learning", MIT Press, 1996.
[3] D. Jurafski, J. H. Martin: "Speech and Language Processing", Prentice Hall, 2000.
[4] S.J.Russell, P.Norvig: "Artificial intelligence.A modern approach", Prentice-Hall International,1995.
[5] D. Tatar: "Unification grammars in natural language processing", in "Recent topics in mathematical and computational linguistics", ed. Academiei, Bucuresti, 2000, pg 289-300.

Faculty of Mathematics and Computer Science, "Babeş-Bolyai" University, Cluj-Napoca, Romania
E-mail address: gabis|dtatar@cs.ubbcluj.ro