N-best Hypotheses

[INDEX]

1. Introduction

1.1 Project Goal

1.2 Sources of Interest

1.3 Current Work

2. Implemented Three Algorithms

2.1 General Viterbi Algorithm

2.1.1 Basic Ideas

2.1.2 Implementation

2.2 N-Best List in HMM

2.2.1 Basic Ideas

2.2.2 Implementation

2.3 BMMF (Best Max-Marginal First)

2.3.1 Basic Ideas

2.3.2 Implementation

3. Some Experiment Results

4. Future Work

5. References

1. Introduction

It's very useful to find the top N most probable figures of hidden variable sequence. There are some important methods in recent years to find such kind of N figures:
(1) General Viterbi Algorithm
(2) MFP (Max-Flow Propagation), by Nilson and Lauritzen in 1998
(3) N-Best List in HMM, by Nilson and Goldberger in 2001
(4) BMMF (Best Max-Marginal First), by Chen Yanover and Yair Weiss in 2003
In this project my task is to realize part of these algorithms and test the results.

1.1 Project Goal

The project goal is to totally understand the basic algorithm ideas and try to realize the most recent three of them (MFP, N-Best List in HMM, BMMF) and test them.

1.2 Sources of Interest

Many applications, such as protein folding, vocabulary speech recognition and image analysis, want to find not just the best configuration of the hidden variables (state sequence), but rather the top N.
Normally the recognizers in these applications are based on a relatively simple model. The existence of N-best list helps to combine additional knowledge sources into the operating process. Even without additional knowledge sources, the N-best list can be used to improve the rate of correctness for the original model.
The researchers have been trying to find better algorithms to solve this N-best list problem. So we are interested in the basic ideas behind some recent algorithms and try to implement them. This is especially useful in teaching and researching.

1.3 Current Work

The original goal has been partly changed by me because of some actual difficulties in implementing these algorithms. Instead of implementing MFP (Max-Flow Propagation), I chose to implement the General Viterbi Algorithm. The reason lies in that, given limited time, the paper about MFP [3] is nearly impossible to thoroughly understand and to realize it. So my current work is having implemented the General Viterbi Algorithm, N-Best List in HMM, BMMF (Best Max-Marginal First), and tested them using the language identification problem in our Assignment 5.

2. Implemented Three Algorithms

This section talks about my implemented (and/or tested) three algorithms. The part of each algorithm includes two sub-sections: one for the basic ideas behind it, the other for the implementation scheme and details.

2.1 General Viterbi Algorithm

2.1.1 Basic Ideas

AS we know, the Viterbi Algorithm is to find the best state sequence Q={$q_1q_2q_3…q_T$} for the given observation sequence O={$O_1O_2O_3…O_T$}. We define the quantity $\delta_t(i) = \max_{q1,q2,..,qt-1} P[q1q2...qt = i, O_1O_2...O_t \mid \lambda]$. This value stands for the best score (highest probability) along a single path, at time t, which explains the first observations and ends in state S_i. To get the next one when reaches time t+1, we recursively define $\delta_{t+1}(j) = [\max_i \delta_t(i)a_{ij}] \cdot b_j(O_{t+1})$. We also define the array $\psi_t(j)$ to keep track of the argument maximize $\delta_t(i)$. The Viterbi Algorithm uses a forward-backward procedure to figure out the best state sequence.

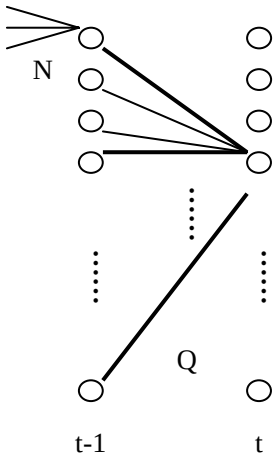
The General Viterbi Algorithm is the direct generalization of Viterbi algorithm to obtain the N best state sequences. The change is that for each time index t and for each state q we have to keep the N best subsequences terminating at this state. And at each step we must select the N best subsequence scores from all candidates.

2.1.2 Implementation

The basic program structure of the General Viterbi Algorithm is similar with that of the original Viterbi Algorithm. It includes four phases: (1) initialization, (2) recursion, (3) termination, (4) path backtracking.

But the form of δ and ψ are little more complicated. They are now matrices of 3 dimensions, i.e. Q*T*N matrices. (Where Q is the number of states, T is the length of observation sequences, and N is the number of best state sequences.)

For each time t and each state q, we must keep N largest δ values. But the strategy of induction to calculate δ value at (t+1) changes: suppose Media is the matrix of Q*T*N δ candidates in time t, we should select the top N values from Media. This scene is illustrated in the following figure: (Bold lines means where for each state the top N largest δ you will get from Q*N δ candidates.)



This kind of the top N-th selection of a matrix is not provided by Matlab build-in library. I implemented it as the nLargest2 function.
The backtracking procedure to find the N-best state sequences can also be realized by using the information stored in ψ at each step.

My program list for the General Viterbi Algorithm is as follows:
(1) general_viterbi.m: the function of General Viterbi Algorithm;
(2) nLargest.m: the function computes the nth largest element in a nonnegative column vector;
(3) nLargest2.m: the function computes the nth largest element in a nonnegative matrix, return result and indices;
(4) test.m: test file for the General Viterbi Algorithm;
(5) other files: all are about the language segmenting HMM data from Assignment 5, including hmm.mat, multinomial_prob.m, normalise.m, segment.mat, stream2text.m.

2.2 N-Best List in HMM

2.2.1 Basic Ideas

N-Best List in HMM Algorithm takes the idea of MFP (Max-Flow Propagation) into a HMM environment.
The entire information used to compute the N-best list from the HMM is encapsulated in two types of entities:

(1) $f_t(s) = \max_{\{x|x_t=s\}} P(x, y)$

(2) $f_{t,t+1}(s, s') = \max_{\{x|(x_t, x_{t+1})=(s,s')\}} P(x, y)$

Actually according to Theorem 1 proved in paper [2], only type (2) of entities is sufficient to compute the N best sequences.

At each phase, the N-Best List Algorithm has three main phases:

(1) Partition phase: $X\backslash\{x^1,\ldots, x^{L-1}\}$ is partitioned into subsets. The algorithm partitions the configuration subset from which the last best configuration x^{L-1} is selected, using a strictly mathematical method.
(2) Candidate phase: For each subset in this partition, compute the probability of its most likely state sequence, called a candidate.
(3) Identification phase. Finally the configuration associated with the highest candidate is identified.
MFP and this algorithm need not to fix N ahead of time, because they find the N-best list sequentially.

2.2.2 Implementation

Because the whole algorithm is based on $f_{t,t+1}(s,s')$, the first important step is to calculate the values of all $f_{t,t+1}(s,s')$ for each time t and each pair of states (s, s'). The calculation is realized in a forward-backward algorithm (fwdback.m). It computes the posterior probabilities and conjunctive probabilities, including $f_{t,t+1}(s,s')$, in an HMM.

The N-Best List Algorithm is divided into three parts:
(1) Compute the most likely state sequence x^1;
(2) Compute the 2nd most likely state sequence x^2
(3) Compute x^3 to x^N.

My implement in computing the 2nd most likely state sequence x^2 doesn't use "real" partitioning immediately. I use another way to directly compute max probability of each partition, store in a matrix and get the max probability of partition 1, and then identify the 2nd most likely state sequence.

Actually I encountered an obstacle in calculating $f_{t,t+1}(s,s')$. The paper inventing the N-Best List in HMM Algorithm [3] says that in paper [4] we can use a single iteration of the forward-backward algorithm to yield $f_{t,t+1}(s,s')$. According to the definition $f_{t,t+1}(s,s') = \max_{\{x|(x_t,x_{t+1})=(s,s')\}} P(x,y)$, I inferred in paper [4] that $f_t(s) = \max_{\{x|x_t=s\}} \alpha_t(i)\cdot\beta_t(i)$ (from Solution to Problem 2) and $f_{t,t+1}(s,s') = \max_{\{x|(x_t,x_{t+1})=(s,s')\}} \alpha_t(i)a_{ij}b_j(O_{t+1})\cdot\beta_{t+1}(j)$ (from Solution to Problem 3). So firstly I started to implement a function according to the method in paper [4] to compute α and β. But later I found that the definition of α and β in paper [4] is different from what I need. Here α and β are sum partial path probabilities, not single partial path probabilities. So my inference can't be used to compute $f_{t,t+1}(s,s')$. Anyway this kind of

'fruitless' attempting is kept as the files findF.m and test1.m.

Later solution for computing $f_{t,t+1}(s,s')$ is based on the faculty's FullBNT library file fwdback.m to calculating xi, which stands for $f_{t,t+1}(s,s')$. Here xi shouldn't be normalized if it is used in the N-Best List Algorithm.

My program list for the N-Best List in HMM Algorithm is as follows:

(1) findF.m: this function attemps to find the term values of $f_t(s)$ and $f_{t,t+1}(s,s')$, but fails. It's kept in my source code anyway;
(2) fwdback.m: this function compute the posterior and conjunctive probabilities in an HMM using the forwards backwards algorithm;
(3) Max_mult.m: this function does matrix multiplication, but sum gets replaced by max;
(4) NBestInHmm.m: this function computes N best lists in Hidden Markov Models;
(5) process_options.m: this function processes options passed to a Matlab function. It provides a simple means of parsing attribute-value options. Each option is named by a unique string and is given a default value.
(6) test1.m: test file for findF;
(7) test2.m: test file for NBestInHmm.
(8) Other files: all are about the language segmenting HMM data from Assignment 5, including hmm.mat, multinomial_prob.m, normalise.m, segment.mat, stream2text.m.

2.3 BMMF (Best Max-Marginal First)

2.3.1 Basic Ideas

BMMF is a method by the tools of approximate inference.

As we know, MFP needs to calculate max-marginals for many times. If wish to significantly reduce the computation, we must use traceback. But traceback operations are problematic in loopy graphs. So what BMMP does is to adopt a better way using additional constraints gradually to calculate MMs without traceback and at the same time reduce the computation. BMMF will provably solve the N-best list problem if MMs can be calculated exactly.

BMMF outputs a set of candidates x_t, one at each iteration. In the first iteration, t = 1, we start by calculating the MMs, and use the max-marginal lemma to find m_1 (the best probable configuration). We now search the max-marginal table for the next best max-marginal value. If we find some x(i) = j, we record this constraint x(i) = j and calculate the MMs with this added constraint. Then the algorithm uses the max-marginal lemma to find the most likely configuration with x(i) = j locked and obtains x_2. Continuously repeat adding constraints and re-computing the MMs, the algorithm will find x_3 to x_N, using the max-marginal lemma.

2.3.2 Implementation

The BMMF Algorithm has been implemented by Chen Yanover. This implementation runs the Best Max Marginal First algorithm on an MRF and finds the M most probable configurations.

The problem is the input for this implementation is a MRF representation, and our test example of language recognize uses a Hidden Markov Model. So what I did is to change the test example HMM representation into a corresponding MRF representation. This work is done in HMM2MRF.m.

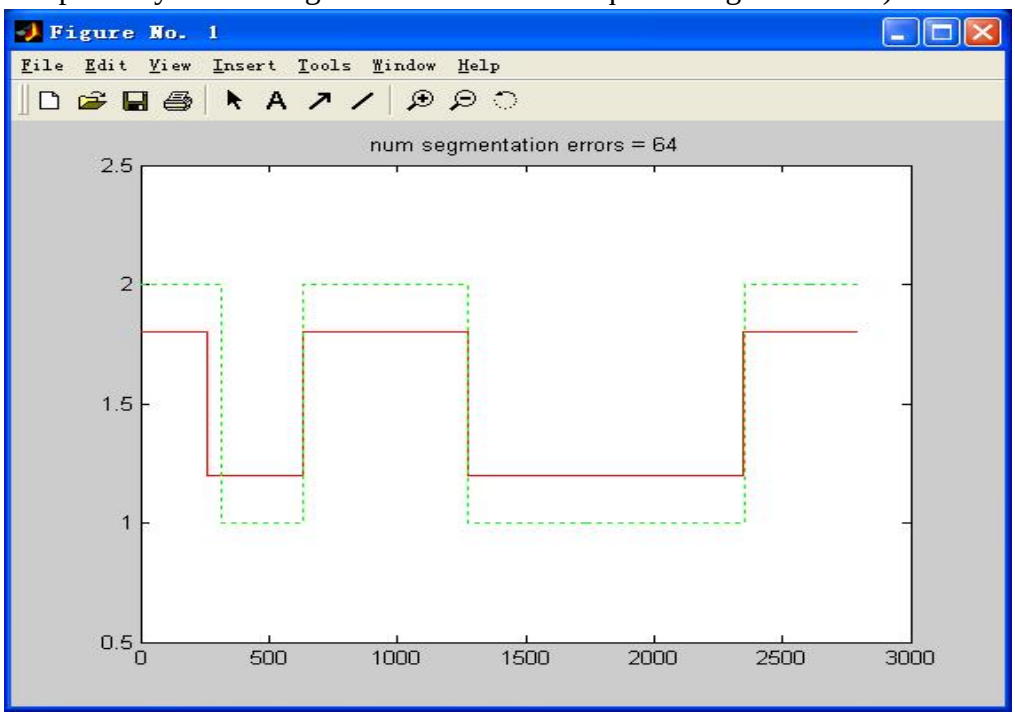
My program list for the BMMF Algorithm is as follows:

(1) HMM2MRF.m: this function converts a HMM representation to corresponding MRF representation;
(2) BMMF_1.0 Algorithm implementation files: they include averageMessages.m, belPropMax.m, belPropMax1.m, BMMF.m, CalculateEnergy.m, colonToij.m, maxMult.m, newCandidates.m, normalize.m, ResolveTies.m;
(3) Other files: all are about the language segmenting HMM data from Assignment 5, including hmm.mat, multinomial_prob.m, normalise.m, segment.mat, stream2text.m.

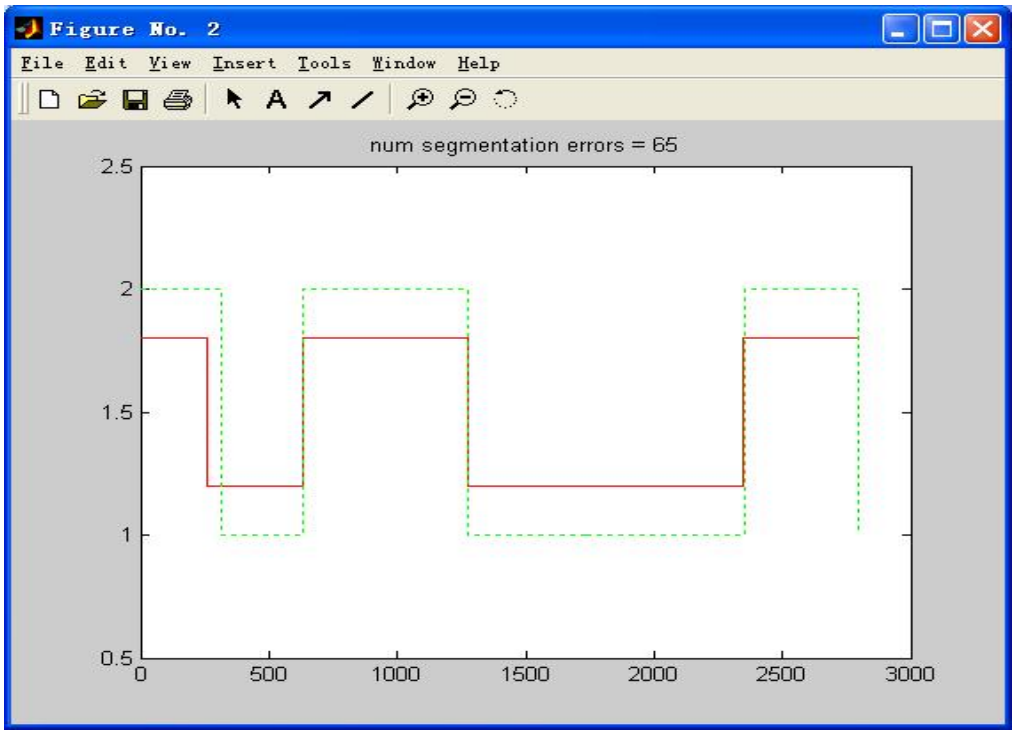
3. Some Experiment Results

The implemented algorithms are tested based on the context of segmenting a sentence which is a mixture of German and Spanish (Assignment 5).

This following is the plot window of the 1st and 2nd best sequence computed by the General Viterbi Algorithm and the N-Best List in HMM Algorithm comparing with the true language segment: (Remember: the difference of the first best sequence computed by Viterbi Algorithm and the true sequence segment is 64.)



Compare the 1st best sequence with the true sequence



Compare the 2nd best sequence with the true sequence

The results of the General Viterbi Algorithm and the N-Best List in HMM Algorithm are just the same, but BMMF may be little different because it is based on the approximate inference.

My test on these algorithms is not enough for time limitation. A better and formal way to test them is to create random Hidden Markov Models (and Markov Random Fields) with different sizes and run these algorithms on the test cases. Both the efficiency and result difference should be included for these test cases.

4. Future Work

As I have shown, I implemented the General Viterbi Algorithm instead of MFP (Max-Flow Propagation) and only test the result on a specific case. So my future work will be the following contents:

(1) Understand and implement the MFP Algorithm;

(2) Create more random test cases of HMM or MRF and test the algorithms' efficiency and results on them.

5. References

1. *An efficient algorithm for finding the M most probable configurations in probabilistic expert systems.* D. Nilsson. Statistics and Computing. 1998.
2. *Sequentially finding the N-Best List in Hidden Markov Models*. Dennis Nilsson, Jacob Goldberger. IJCAI 2001.
3. *Finding the M Most Probable Configurations Using Loopy Belief Propagation.* Chen Yanover, Yair Weiss. NIPS 2003.
4. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Lawrence R. Rabiner. IEEE 1989.