CS 532c — Probabilistic Graphical Models
N-Best Hypotheses

Zvonimir Rakamaric Chris Dabrowski

December 18 2004

Contents

1 Introduction

Our project concerns finding the N-Best hypothesis (configurations). We implemented a number of algorithms in Matlab to find the N most probable configurations. These algorithms are called Brute Force, N-Viterbi, and Nilsson and Goldberger and they are for HMMs. We also provided a transformation from HMMs to MRFs so that we could use an already existing implementation of the Loopy Best Max-Marginal First algorithm. The first three are exact algorithms, while the last one is an approximation algorithm. We tested the N-Viterbi, Nilsson and Goldberger, and Loopy Best Max-Marginal First against each other, running twelve sets of tests. In our tests, we used three different seeds and varied the number of states, the number of different characters (symbols) that can be found in the data sequence, the length of the data sequence, and the number of best configurations. Then, we compared the running times of the algorithms and based on this made some conclusions.

The project report is structured as follows. Section 2 talks about background info related to N-Best. Section 3 describes the Brute Force algorithm, section 4 describes the N-Viterbi algorithm, section 5 describes the Nilsson and Goldberger algorithm, and section 6 describes the Loopy Best Max-Marginal First algorithm. Section 7 describes our testing methodology and section 8 discusses the results of our tests. Section 9 contains conclusions based on the results of our work, and section 10 suggestions for future work.

The source code of our implementation of the algorithms, the source code of the scripts for the test cases, the results of our tests, and the graphs of these results are all available at http://www.cs.ubc.ca/~zrakamar.

2 Background Info

Finding N-Best hypothesis (not just the best one) is important in many applications, like speech recognition, protein-folding and image analysis. When developing a speech recognition system, it is normally not sufficient to provide only models of the words that are to be recognized. To obtain reasonable performance, it is common to include additional knowledge sources in the recognition process. For instance, additional knowledge sources can be models of the natural language or knowledge about the task the recognition system will be used for. If all knowledge sources are simultaneously included, the search for the most likely word string (hypothesis) for a spoken utterance might become very complex. Techniques that allow to apply the different knowledge sources sequentially and thus reduce computation have been proposed and these techniques are based on the N-Best search paradigm. An N-Best algorithm generating the list of the N most likely hypotheses is required. These N-Best hypotheses then can be re-scored according to additional knowledge sources. In this way, also complex knowledge sources can easily be included in the recognition process [5].

One of the real-life examples of where N-Best has been used is in speech recognition for the United Airlines Flight Information Service [7]. This is a service where callers can speak naturally to an automated system and get information about arrival and departure times for flights. It has also been used in speech recognition of Arabic which is often more difficult than speech recognition of more "mainstream" languages such as Spanish and English because of the extreme dialectical variation [1].

3 Brute Force Algorithm

3.1 Description

This algorithm finds all the possible configurations and then calculates their probabilities. It then orders them from the highest to the lowest probability. Once we have this we can pick the N-Best from the array of sorted configurations. This algorithm is obviously not very powerful as it has an exponential running time. So therefore, it can only be run on small samples and we haven't included it in our testing. It was mainly used for comparing initial results against our other algorithms to make sure that they were implemented correctly.

3.2 Computational Complexity

Let S be the number of states and T the length of the data sequence. Then the running time of the Brute Force algorithm is $O(S^T)$.

4 N-Viterbi Algorithm

This algorithm is just a generalization of the Viterbi algorithm [6]. The Viterbi algorithm is a dynamic programming method for finding the single best state sequence. The N-Viterbi algorithm uses this same principle except it stores the N-Best sequences at each time step. One of the flaws of this algorithm is that you have to specify N before you start running it.

4.1 Description

N-Viterbi works by first specifying data structures to hold the N-Best paths.

There are three objects that are input to the algorithm — prior, obsmat and transmat. Prior is a matrix representing the probability of starting in a certain state. Obsmat is a matrix representing the probability of being in a certain state given what was observed. Transmat is a matrix representing the probability of a transition from a state A to a state B.

We first calculate the probability of starting out in a certain state given the first character (symbol) that was observed.

Then we recursively calculate the probabilities of what happens at each time step. We calculate the probability given that we are in a certain state that we transition to another state. We sort these probabilities and keep the N-Best for each time step. Then we multiply the probability of having transitioned to a certain state with the probability of being at that state given the data character that was observed.

We keep doing this for every time step, and in every time step one character from the data sequence is observed.

After we have gone through all the time steps we find the N-Best configurations by back-tracking through our data structures.

4.2 Computational Complexity

Let S be the number of states, T the length of the data sequence, and N the number of configurations that are requested. Then the running time of the N-Viterbi algorithm is $O(NTS^2)$.

5 Nilsson and Goldberger Algorithm

This is a simplification of Nilsson's Max Flow Propagation Algorithm [3, 4]. It is specifically for HMMs. It is a divide and conquer algorithm. An advantage of this algorithm over the N-Viterbi algorithm is that this is an anytime algorithm. It means that the number of wanted N-Best configurations doesn't have to be specified before running the algorithm. For instance, we have calculated the 10 best configurations and based on the result we have concluded that we need 10 more. We can just continue running the algorithm, and the data that was generated while searching for the first 10 best configurations will be used to find the next 10 configurations (from the 11th to the 20th).

5.1 Description

Let $CL = \{\}$ be the candidate list.

The algorithm first runs a single iteration of the forwards-backwards algorithm to determine the following terms for each time index t:

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

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

Using these terms the algorithm can find the most likely state sequence.

Then using the most likely state sequence the algorithm comes up with a number of candidates of length ranging from 1 to the length of the sequence. The algorithm then computes a probability for each of these candidates and adds them to CL. It uses the candidate with the highest probability to calculate the next best state sequence. The candidate that was used is removed from CL and the state sequence that was just calculated is used to find the next set of candidates to add to CL. The algorithm continues this for N steps until the N-Best sequences are found.

5.2 Computational Complexity

Let S be the number of states, T the length of the data sequence, and N the number of configurations that are requested. Then the theoretical running time is $O(S^2T)+O(TSN)+O(TNlog(TN))$. The part of the running time $O(TNlog(TN))$ is dependent on a certain implementation of the algorithm. This implementation stores all of the candidates in a 2-3 search tree. We did not use this data structure in our implementation so searching for the best candidate takes $O(TN^2)$ time. Therefore, the running time for our implementation is actually $O(S^2T) + O(TSN) + O(TN^2)$.

5.3 Implementation

All operations in our implementation are performed in the log domain to avoid problems with underflow which would be encountered with bigger examples. We use the repmat function quite a bit, therefore, instead of using the original Matlab repmat function we are using the implementation of this function in C provided by Kevin Murphy to improve our running time [2].

6 Loopy Best Max-Marginal First Algorithm

This algorithm [8] does not use the junction tree formalism, but rather uses max-marginals. Therefore, this algorithm is used in many real world problems when building a junction tree is infeasible. It does approximate inference as opposed to exact inference.

6.1 Description

The algorithm starts by finding the max-marginals (MM). It uses the max-marginal lemma to find the best state sequence. It then searches the max-marginal table to find the next best max-marginal value. This value corresponds to some state in the sequence being set to for instance n. Call this state x_i. The algorithm then locks $x_i = n$ and calculates the MMs with this added constraint. It then uses the max-marginal lemma to find the best sequence with x_i locked. This is the second best state sequence. It then adds the constraint that $x_i \neq n$ to the original constraint set and calculates the MMs again. In the next iteration it searches all the previous tables and finds the best remaining max-marginal. It locks the state $x_j = m$ corresponding to that max-marginal and calculates the MMs again, and uses them to find the third best state sequence. It then calculates the MMs with the added constraint that $x_j \neq m$. This is repeated N times so that the N-Best state sequences are found.

In the algorithm the MMs are based on the beliefs computed by loopy max-product belief propagation.

6.2 Implementation

For testing this algorithm we used the implementation which can be found at:
http://www.cs.huji.ac.il/∼cheny/.

7 Testing

We have run the tests on a SUN Fire 880 server with 8 processors and 32GB of RAM. The version of Matlab used is 6.5.1.

Let S be the number of states, T the length of the data sequence, N the number of configurations that are requested, and O the number of different characters that can be found in the data sequence. We did the testing by starting with three different seeds. Each seed had a preset value for S, T, N and O and then we increased each of these by a predefined amount and recorded the running times. We did this for each seed. The first seed had the following values:

- $S = 2$
- $T = 15$
- $N = 5$
- $O = 2$

We ran all three algorithms five times for these values, and then calculated the mean running time for each algorithm. We then increased S to 5 and again ran all three algorithms five times,

and calculated the mean for each algorithm. We continued doing this, increasing S by 5 until we reached $S = 100$. We then set the value of S back down to 2 and did the same thing for T, N and O. T was increased by 100 each time up to 4000, N was increased each time by 50 up to 1000 and O increased by 5 each time up to 50. The second seed was:

- $S = 20$
- $T = 50$
- $N = 5$
- $O = 10$

And the third seed was:

- $S = 30$
- $T = 1000$
- $N = 5$
- $O = 20$

For the second and third seeds we did the same thing as for the first seed. We incremented S, T, N and O by the same amounts up to the same maximums.

8 Results

The results of our tests are available at http://www.cs.ubc.ca/~zrakamar. The results show that the Loopy Best Max Marginal First algorithm was slower in all test instances than the N-Viterbi algorithm and the Nilsson and Goldberger algorithm. The N-Viterbi and the Nilsson and Goldberger algorithms took turns being the fastest, depending on the settings. The N-Viterbi algorithm outperforms the Nilsson and Goldberger algorithm for cases where there is a small number of states that can be assigned at each time step of the sequence. Once the number of states is increased the Nilsson and Goldberger algorithm does much better than the N-Viterbi algorithm. This is consistent with the claims made by Nilsson and Goldberger in their article [4].

9 Conclusions

The main contribution of this project is the implementation of two algorithms — the N-Viterbi algorithm and the Nilsson and Goldberger algorithm (available at http://www.cs.ubc.ca/~zrakamar) and their comparison against the Loopy Best Max-Marginal First algorithm.

In all our tests the Loopy Best Max-Marginal First algorithm was slower that the other two algorithms. This doesn't correspond to the results from the article written about the Loopy Best Max-Marginal First algorithm [8]. We conclude that the test data (HMMs) that we generated doesn't show the real strength of this algorithm. HMMs always have cliques of at most size two. The Loopy Best Max-Marginal First algorithm is designed especially to not have to construct a junction

tree, but since we are using HMMs constructing the junction tree is not that computationally expensive. Therefore, we can conclude the results we have for the Loopy Best Max-Marginal First algorithm are not indicative of its usefulness.

10 Future Work

While working on the project and learning more and more about this area we found many things that would be an interesting extension to our project. Future work would include implementing the Max Flow Propagation algorithm that works for junction trees. This is a more general version of the Nilsson and Goldberger algorithm that we have implemented.

Based on our tests, it seems like the implementation of the Loopy Best Max-Marginal First algorithm that we used for testing is quite slow. This algorithm should work well for graphical models with large junction trees. Therefore, most likely the graphs used for our tests (HMMs) did not have cliques that were big enough to show the real strength of this algorithm. A good thing to do would be to test this algorithm thoroughly and optimize it if necessary, or make a completely new implementation.

Our implementation of the Nilsson and Goldberger algorithm doesn't use a 2-3 search tree data structure to store generated elements. Incorporating this data structure into the implementation would probably make this algorithm even faster.

To make the algorithms we implemented more compatible with other N-Best algorithms (Frank Hutter and Sohrab Shah, "A general framework for comparing (approximate) inference algorithms"), the implementation of the interface that uses factor graphs would be a good way to go.

We have tested our algorithms on randomly generated HMMs. It would be very interesting to test them on real data and then compare the results.

References

[1] Katrin Kirchhoff and Dimitra Vergyri. Cross-dialectal acoustic data sharing for arabic speech recognition. Website. http://ssli.ee.washington.edu/people/katrin/Papers/kirchhoff-icassp04.pdf Accessed December 10, 2004.

[2] Kevin Murphy. Function repmatc from kmptools. Website. http://www.cs.ubc.ca/ murphyk/Software/index.html Accessed December 14, 2004.

[3] D. Nilsson. An efficient algorithm for finding the m most probable configurations in probabilistic exper systems. *Statistics and Computing*, 8:159–73, 1998.

[4] Dennis Nilsson and Jacob Goldberger. Sequentially finding the n-best list in hidden markov models. In *Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[5] Heiko Purnhagen. N-best search methods applied to speech recognition. Website. http://www.tnt.uni-hannover.de/js/org/whois/wissmit/purnhage/thesis.html Accessed November 15, 2004.

[6] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286, 1989.

[7] Carmen Wai, Roberto Pieraccini, and Helen M. Meng. A dynamic semantic model for re-scoring recognition hypotheses. Website. http://www.se.cuhk.edu.hk/people/hmmeng/Meng_DynSemModel_ICASSP2001.pdf Accessed December 10, 2004.

[8] Chen Yanover and Yair Weiss. Finding the m most probable configurations in arbitrary graphical models. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.