Implementation of the Linear Sparse Version Algorithms to Hidden Markov Model

Hao Sheng, Xiaozhou Wang

May 1, 2017

Abstract

Hidden Markov Model(HMM) is a statistical model which is used to describe Markov processes in which the states are not observed. It has been widely used in the area of recognition(such as speech, handwriting,etc.) and bioinformatics that usually involve long observed sequences. Considering the run time as well as the memory usage of the corresponding computation, an updated memory sparse version of implementing HMM is proposed by Alexander Churbanov and Stephen Winters-Hilt. Researchers suggested the updated Baum-Welch algorithm implementation to HMM could reduce the memory use significantly, compared with other implementations of the Baum-Welch algorithm. The authors also discussed the linear memory approach of implementing the Viterbi decoding algorithm to HMM. In this project, we would implement the memory sparse version of the above two algorithms to HMM, optimize them, and also compare them with the ones that from 'hmmlearn' library in Python.

Keywords— Hidden Markov Model, Viterbi algorithm, Baum-Welch algorithm, linear memory

1 Background

Our final project is based on the paper "Implementing EM and Viterbi algorithms for Hidden Markov Model in linear memory", published by Alexander Churbanov and Stephen Winters-Hilt (2008). Researchers first argued that conventional implementations of HMM's Baum-Welch and Viterbi decoding

algorithm can be limited due to the restrictions regarding the size of the dynamic programming table, and become problematic in terms of the use of memory. The authors agreed that checkpointing algorithm implements the Baum-Welch algorithm and Viterbi decoding algorithm associate with $O(\sqrt{T}N)$ and $O(\sqrt{T}\ N\ +\ T\ )$ memory, which is more efficient compared with their conventional implementations. However, the authors suggested these can be further reduced to O(N(Q + ED)) and O(T) by adapting the linear memory implementation. Despite providing with the theoretical inference of the algorithms, the authors also gave results of implementing them in practice. According to Churbanov and Winters-Hilt, the linear memory implementations of Viterbi decoding algorithm to HMM involves less use of memory although it requires the same runtime as the previous two algorithms. However, as for the linear memory modification of the Baum-Welch algorithm that could alleviate the use of memory, the authors suggested it requires more running time than either the conventional or the checkpointing approach. Since the checkpointing algorithm tends to provide an optimal trade-off between the run time and the use of memory, researchers was inspired to optimize the efficiency of the memory sparse version of Baum-Welch algorithm by constraining its runtime and thus make it have a similar runtime as the checkpointing algorithm.

Note: In the above Big-Oh notation of algorithms: T represents the length of the observed sequence; Q represents the HMM node out-degree; E represents the number of free emission parameter; and Q represents the number of free transition parameters.

2 Description of algorithm

According to the authors, a HMM model can be used to depict Markov processes involved hidden states, in which each transition between hidden states gives either a discrete(such as a character) or continuous emission. Although the states are hidden here, which results in no explicit state transition probability, it is still possible to figure out the parameters involved in the model(e.g. Baum-Welch algorithm) as well as the most likely sequence of the hidden states(e.g. Viterbi decoding algorithm) since the emissions are depend on the corresponding states.

The conventional HMM procedure can be described as the following:
• A set of states S = $\{S_1, ..., S_N\}$ with q_t being the state visited at time t.
• A set of PDFs B = $\{b_1(o), ..., b_N(o)\}$, which represents the emission probabilities such that $b_j(o_t) = p(o_t|q_t = S_j)$ where 1≤ j ≤N, and o_t describes the observation at time-point t.
• The state-transition probability matrix A = $\{a_{i,j}\}$, 1≤ i, j ≤ N, where $a_{i,j}$ = $p(q_{t+1} = S_j|q_t = S_i)$.
• The initial state distribution which is represented by a vector Π such that Π = $\{\pi_1, ..., \pi_N\}$.

Thus, the authors proposed that the above model can be specified by a set of three parameters such that $\lambda = (A, B, \Pi)$. Baum-Welch algorithm and Viterbi decoding algorithm are two algorithms that are often being implemented to HMM process. While Baum-Welch algorithm is adapted to estimate the parameters of HMM processes, Viterbi decoding algorithm is often served as a method to explore the most possible sequence of the hidden states. The detailed description of these two algorithms as well as their implementations to the HMM process is discussed in the following section.

2.1 Memory sparse version of the Baum-Welch algorithm

As a special case of EM algorithm, Baum-Welch algorithm can be used to estimate the HMM parameters through calculating the forward and backward probabilities.The memory sparse version of the Baum-Welch algorithm implementation to HMM was being modified to be independent of the observed sequence(T)'s length, and linearly dependent on the number of states. Several new terms were introduced by the authors here, and the initial values are updated accordingly.
$E_i(\gamma, t, m)$ - the weighted sum of probabilities of all possible state paths that emit subsequence $o_1, ..., o_t$ and finish in state m, while among all the emissions, state i emits observation γ at least once.
$T_{i,j}(t, m)$ - the weighted sum of probabilities of all possible state paths that emit subsequence $o_1, ..., o_t$ and finish in state m, for which state i goes to state j for at least once. The weight of each path is the number of times that the previous transaction happened under that path.

Instead of calculating the forward and backward probabilities as in the conventional Baum-Welch algorithm, only the backward probability of the state occupation $B_t(i)$ is calculated here, which makes the estimation procedure to be faster. However, if the given sequence is somehow long, it would be not precise enough for just having these original $B_t(i)$s. Instead, the $B_t(i)$s will be scaled, and thus the memory sparse version can be done by using a scaled backward sweep. Our implementation of this algorithm to HMM is based on the pseudo code provided by the authors in their paper.

2.2 Memory sparse version of the Viterbi decoding algorithm

Given an observed sequence, there can be multiple paths lead to it. Viterbi decoding algorithm is used to find the path that with the highest probability to happen, and thus could identify the most likely sequence of the hidden states. The checkpointing implementation of Viterbi algorithm splits the sequence into $\sqrt{T}$ blocks of $\sqrt{T}$ symbols each and then reconstruct them based on the last check point. This reduces the memory space while increases the computational time, compared with the conventional implementation of the algorithm. However, the memory sparse version of the Viterbi decoding algorithm implementation to HMM provided by the authors tends to give the same computational time as the conventional implementation while further reduce the use of memory during computation to O(T), conceptually.
The initialization section for the memory sparse version of the Viterbi decoding algorithm is exactly the same as its conventional version. Nevertheless, when computing the maximum probability for a state to run at a certain time t, a linked list is proposed to be used, instead of using an array. Since Python does not have a linked list structure, the linear sparse version of Viterbi decoding algorithm is implemented based on the usage of the deque structure, imported from the 'collections' library, which is similar to a doubly linked list.

3 Optimization

As for the optimization, we employed vectorization to avoid the use of triple for-loops under the update section of the Baum-Welch algorithm. We used broadcasting with numpy.newaxis to implement Baum-Welch algorithm much

faster. As we can see from Benchmark part in the report, under class HMM we have 2 functions for Baum-Welch algorithm called Baum_Welch and Baum_Welch_fast. In Baum_Welch_fast, vectorization is applied when calculating ξ while in Baum_Welch, we use a for loop. Notice in Baum_Welch, all other parts are implemented with vectorization. This is just an example how vectorization greatly improve the speed. Notice that the run time for vectorized Baum-Welch algorithm is 2.43 s per loop (with scaling) and 1 s per loop (without scaling) compared to 4.01 s per loop (with scaling) and 261 s per loop (without scaling). Other functions are implemented with vectorization as well. Vectorization greatly improves our time performance.

4 Simulations

Under the package HMM, the function sim_HMM is implemented to simulate HMMs. The simulations are done within simulations part in the report. In the simulations part, we evaluated the performance of Baum-Welch algorithm and Viterbi decoding by checking the accuracy of Hidden states. Overall, Baum-Welch algorithm and Viterbi decoding give a good estimate of hidden states given a good initial values. In our experiments, we simulated HMMs with 3 hidden states and 4 outputs. We varied different parameters: the length of simulated chain, the initial values for Baum-Welch algorithm and the number of iteration of Baum-Welch algorithm. The results illustrate some limitations of Baum-Welch. Firstly, Baum-Welch algorithm tends to overfit the data: $P(Y|\theta_{final}) > P(Y|\theta_{true})$. Second disadvantage of Baum-Welch algorithm is that the algorithm does not guarantee a global maximum: poorly chosen initial values would make Baum-Welch algorithm to be trapped into a local minimum.

5 Applications

We applied our implementation of HMM onto a (synthesized) weather dataset. The dataset can be retrieved from here. The dataset has two columns: the first column indicates the weather (hidden states) and the second columns indicate if we can see an umbrella on the street (see the HMM tutorial link within the website for more description). From our knowledge, we proposed the initial transition matrix , emission matrix and initial state distribution

as in the report. After estimating these two matrices by Baum-Welch algorithm, we used Viterbi decoding to find the most likely path and obtained an accuracy of 63.4%.

6 Comparative analysis

We compared our implementations with the hmmlearn package. From the Comparative analysis section in the report, we can see that Viterbi decoding is coded correctly. However, the time complexity of our implementation is not good enough. As we check the implementation of hmmlearn package, we found that they implemented Viterbi decoding and BaumWelch algorithm in C to improve time performance. However, there is limitation of hmmlearn package. Because the hmmlearn package does not take in any initial values for BaumWelch algorithm (they randomly simulate initial values), the outcome of the hmmlearn package would fluctuate (see plot in the Code Report). However, as we implemented BaumWelch algorithm that takes in initial values, the choices of initial directly decide the final performance of the algorithm. This again shows the drawback of BaumWelch algorithm: the choice of initial values greatly influence the outcomes because there might be a few local maximum (of likelihood). EM algorithm may be trapped in this place.

7 Discussion

We see vectorization helps to improve the run time of both baum-welch algorithm and viterbi decoding algorithm. Although we noticed there is a limitation with the Baum-Welch algorithm such that it may not give the global optimal result, it is possible for the implemented algorithm to avoid this by adjusting its initial values. In the contrast, the BaumWelch algorithm from hmmlearn package generates initial values randomly, which would lead to another problem: inconsistency. However, since hmmlearn package implements the algorithms in C, they do have a better time performance. As the author discussed at the end of their paper, when processing long sequences, the problem is not about memory but the run time in this case. Thus, they are working on the optimization of the linear memory implements of the algorithms in terms of the time performance.

One thing need to be noticed here is that in the original paper, there is neither clear definition given for the free emission parameters ε, nor the free emanating transitions τ. Since these two ambiguous terms also occur in the pseudo code of Baum-Welch algorithm's linear memory implementation, the results of implementing such algorithm is little departure from what we expected. Similarly, for the linear memory implementation of viterbi decoding algorithm, the pseudo code does not give much explanation regarding its notation of the linked nodes. Thus, our implementation was considerably limited by the lack of clear definitions. Further improvements can be achieved if these terms can be clarified.

8 REFERENCES

Churbanov, A., & Winters-Hilt, S. (2008). Implementing EM and Viterbi algorithms for Hidden Markov Model in linear memory. *BMC bioinformatics*, 9(1), 224.

Bilmes, J. A. (1998). A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute*, 4(510), 126.

BaumWelch algorithm. (n.d.). In Wikipedia. Retrieved May 1, 2017, from https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm