Hidden Markov Model

DMKMK-UPO

Carlos López Roa
me@mr3m.me

February 3, 2017

Abstract

In this work the scope was to implement a Hidden Markov Model module using the object oriented programming language Python in the version 2.7. The implementation exposes methods for generating sequences, finding the most probable sequence of hidden states (Viterbi algorithm) and estimating the parameters of a HMM from observations (Baum - Welch algorithm)

Code is available in the repository https://github.com/mr3m/LGM/

1 Introduction

Hidden Markov Model (HMM) was first describen in 1960 by Ruslan L. Stratonovich [1]. Then it was further described in a series of statistical papers by Leonard E. Baum et al in the second half of the 1960s. One of the first applications of HMMs was speech recognition, starting in the mid-1970s [2]. In the second half of the 1980s, HMMs began to be applied to the analysis of biological sequences, [3] in particular DNA. Since then, they have become ubiquitous in the field of bioinformatics. [4]

In a hidden Markov model the system being modelled is assumed to be a Markov process with unobserved (hidden) states. The state is not directly visible, but the output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens.

2 Hidden Markov Model

2.1 Linear Dynamical Systems

Linear Dynamical Systems (LDS) represent a system of one or more real-valued variables that evolve linearly over time, with some Gaussian noise. Such systems are also often called Kalman Filters.

A lineal dynamical system can be viewed as a dynamic Bayesian network where the variables are all continuous and all of the dependencies are linear Gaussian.

LDS are represented as a state-observation model, where the state and observation are both vector-valued random variables, and the transition and observation models are encoded using matrices. More precisely, the model is generally define via the following set of equations:

$$\begin{aligned} P\left(X^{(t)}|X^{(t-1)}\right) &= \mathcal{N}\left(AX^{(t-1)};Q\right), \\ P\left(O^{(t)}|X^{(t)}\right) &= \mathcal{N}\left(HX^{(t)};R\right), \end{aligned} \tag{1}$$

where: X is an $n-$vector of state variables, O is an $m-$vector of observation variables, A is an $n \times n$ matrix defining the linear transition model, Q is an $n \times n$ matrix defining the Gaussian noise associated with the system dynamics, H is an $n \times m$ matrix defining the linear observation model, and R is an $m \times m$ matrix defining the Gaussian noise associated with the observations.

LDS satisfy the differential equation [6]

$$\begin{aligned} \frac{\partial}{\partial t}X(t) &= A \cdot X(t) + \epsilon_1 \\ O(t) &= H \cdot X(t) + \epsilon_2 \end{aligned} \tag{2}$$

A LDS restricting the vectors X and O from $\mathbb{R}^n$ and $\mathbb{R}^m$ to $\mathbb{Z}^n$ and $\mathbb{Z}^p$ respectively is a Hidden Markov Model [5]

In this case, let us define

$$\lambda = \{A, H, X(0)\} = \{A, B, \pi\}$$

as a HMM.

Let $q_i \in S$ be the set of states (hidden variables) and $o_i \in O$ the set of observations. Then let

$$\begin{aligned}
\alpha_t(i) &= P(\{o_i\}_1^t, q_t = S_i|\lambda), \\
\alpha_1(i) &= \pi\, b_i(O_1) \\
\alpha_{t+1}(i) &= [\sum_{k=1} \alpha_t(k) a_{ki}] b_i(O_{t+1}) \\
P(O|\lambda) &= \sum_{i=1} \alpha_T(i)
\end{aligned} \tag{3}$$

$$\begin{aligned}
\beta_t(i) &= P(\{o_i\}_{t+1}^T, q_t = S_i|\lambda), \\
\beta_T(i) &= 1 \\
\beta t(i) &= \sum_{j=1} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \\
P(O|\lambda) &= \sum_{i=1} \pi_i b_i(o_1) \beta_1(i)
\end{aligned} \tag{4}$$

be the recursive definition of α and β.

And let us define the most likely state at time t

$$\begin{aligned}
\gamma_t(i) &= P(q_t = S_i|O, \lambda) \\
&= \frac{\alpha_t(i)\beta_t(i)}{P(O\ |\lambda} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^N \alpha_t(i)\beta_t(i)} \\
q_t &= \arg\max_i \gamma_t(i)
\end{aligned} \tag{5}$$

2.2 Viterbi Algorithm

The Viterbi algorithm for finding the Maximum A-posteriori Probability (MAP) assignment in an HMM was proposed by Viterbi (1967); it is the first incarnation of the variable elimination algorithm for MAP algorithm. [5]

Using the previous definitions let us define the Viterbi algorithm as:

$$\begin{aligned}
\delta_t(i) &= \max P(\{q_i\}_1^{t-1}, q_t = S_i, |\{O_i\}_1^t, \lambda \\
\delta_1(i) &= \pi_i b_i(O_1) \\
\delta_{t+1}(i) &= [\max_k \delta_t(k) a_{ki}] b_i(O_{t+1})
\end{aligned} \tag{6}$$

for finding the most probable sequence of hidden states.

2.3 Baum-Welch Algorithm

Let us define the probability of transition δ_{ij} at time t:

$$\xi_t(i,j) = \frac{\alpha_t(j) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}, \tag{7}$$

and the probability of being in state S_i at time t as:

$$\gamma_t(i) = \frac{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)}{P(O|\lambda)}, \tag{8}$$

the a-posteriori probability of transition δ_{ij} as:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(j) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)}, \tag{9}$$

the a-posteriori probability of observing v_k in state S_j as:

$$\bar{b}_j(k) = \frac{\sum_{t=1,o_t=v_k}^{T-1} \alpha_t(j)\beta_t(j)}{\sum_{t=1}^{T-1} \alpha_t(j)\beta_t(j)}, \tag{10}$$

the probability of starting in S_i given the observation O:

$$\begin{aligned} \bar{\pi} &= \frac{\alpha_1(i)\beta_1(i)}{P(O|\lambda)} \\ &= \frac{\pi_i b_i(O_1)\beta_1(i)}{P(O|\lambda)} \\ &= \gamma_1(i) \end{aligned} \tag{11}$$

Then, the Baum Welch Algorithm consists in updating the values

$$\begin{aligned} \pi &:= \{\bar{\pi}_i\}_1^N \\ A &:= \{\bar{a}_{ij}\}_1^N \\ B &:= \{\bar{b}_j(k)\}_{1,1}^{N,|\Sigma|} \\ \lambda &:= \lambda' = \{\pi, A, B\} \end{aligned} \tag{12}$$

Until $P(O|\lambda') \geq P(O|\lambda)$

3 Implementation

The described structure was implemented in Python 2.7 [7] using the library Numpy [8] as the underlying structure for managing multidimensional arrays and vectors efficiently.

The implemented class exposes three methods for: generating sequences (genseq), generating the most probable sequence of hidden states (viterbi) and estimating the parameters λ from a set of observations (baum-welch) in a single class as shown in the figure 1

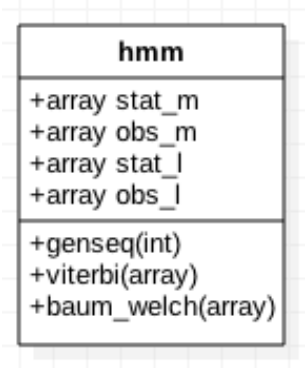


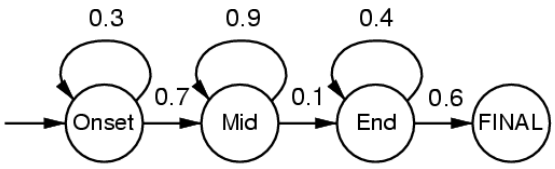
Figure 1: UML diagram of the implemented solution. It consists in a single class that exposes three methods.

4 Tests

A model of a phone call was tested with the parameters and structure shown in figure 2

The transition matrix is given by

```
INIT
INIT Onset 1
Onset Onset 0.3
Onset Mid 0.7
Mid Mid 0.9
Mid End 0.1
```



Output probabilities for the phone HMM:

Onset:	Mid:	End:
C1: 0.5	C3: 0.2	C4: 0.1
C2: 0.2	C4: 0.7	C6: 0.5
C3: 0.3	C5: 0.1	C7: 0.4

Figure 2: A model of a phone call with the transition matrix and structure given.

```
End End 0.4
End FINAL 0.6
```

And the emission matrix is given by

```
Onset C1 0.5
Onset C2 0.2
Onset C3 0.3
Mid C3 0.2
Mid C4 0.7
Mid C5 0.1
End C4 0.1
End C6 0.5
End C7 0.4
```

5 Results

5.0.1 Generating sequences

The benchmark program gives the output for 10 sequences

```
C1 C4 C4 C4 C5 C4 C3 C4 C3 C3 C4 C5 C4 C4 C4 C7 C4
C2 C5 C4 C7
C1 C4 C4 C4 C4 C3 C4 C4
C2 C5 C4 C5 C4 C4 C6
C1 C4 C3 C4 C4 C4 C7 C6 C6
C1 C4 C4 C4 C3 C4 C4 C5 C4 C4 C3 C4 C3 C3 C3 C5 C3 C4 C3 C4 C4 C4 C4 C6 C7
C2 C1 C5 C3 C4 C4 C4 C4 C4 C5 C3 C6 C7 C6 C6 C6
C1 C4 C7 C7
C3 C4 C4 C4 C7
C2 C5 C7 C6
```

The implemented program gives the output for 10 sequences

```
C3 C1 C3 C4 C5 C3 C4 C4 C4 C4 C4 C4 C4 C4 C7
C2 C2 C4 C5 C6
C1 C4 C6
C1 C1 C3 C4 C4 C5 C4 C4 C3 C4 C6
C1 C4 C4 C4 C3 C4 C5 C4 C3 C4 C5 C4
C1 C1 C4 C4 C4 C4 C4 C4 C4 C4 C4 C6 C7 C7
C2 C5 C5 C6
C1 C5 C6 C7 C6 C6 C6 C7
```

```
C1 C3 C4 C4 C4 C4 C4 C4 C4 C5 C4 C4 C4 C4 C4 C6
C1 C4 C4 C4 C4 C4 C4 C4 C4 C3 C4 C4 C4 C4 C4 C4 C7 C4 C6 C6
```

They are very similar.

5.0.2 Viterbi

Given the sequences

```
C1 C2 C3 C4 C4 C6 C7
C2 C2 C5 C4 C4 C6 C6
```

The output of the benchmark

```
P(path)=0.625286
path:
C1 Onset
C2 Onset
C3 Mid
C4 Mid
C4 Mid
C6 End
C7 End
P(path)=0.936748
path:
C2 Onset
C2 Onset
C5 Mid
C4 Mid
C4 Mid
C6 End
C6 End
```

And the output of the implementation

```
Onset, Onset, Mid, Mid, Mid, End, End
Onset, Onset, Mid, Mid, Mid, End, End
```

They are identical

5.0.3 Baum-Welch

Training with 100 generated sequences from the original model, the benchmark outputs:

```
INIT
INIT  Onset  1
Onset Onset  0.380199
Onset  Mid  0.619801
Mid  Mid  0.888656
Mid  End  0.107036
End  End  0.452363
```

And

```
Onset  C1  0.539226
Onset  C2  0.18594
Onset  C3  0.274833
Mid  C3  0.198924
Mid  C4  0.726476
Mid  C5  0.0746003
End  C4  0.0828177
End  C6  0.507014
End  C7  0.410168
```

Whereas the implemented model outputs for the transition matrix

```
Init
Init Onset 1.,
Onset Onset 0.19343991,
Onset Mid 0.80656009,
Mid   Mid 0.86083614,
Mid   End 0.13916386,
End   End 1.
```

And the emision matrix

```
Onset C1 0.45434628,
Onset C2 0.10411344,
Onset C3 0.44154029,
Mid   C3 0.20574315,
Mid   C4 0.71503849,
Mid   C5 0.07921837,
End   C4 0.0470633,
End   C6 0.51041397,
End   C7 0.44252273
```

5.1 Performance

A test was done to show the performance of genseq, Viterbi and Baum-Welch algorithms when the input size grows. The results can be seen in figure 3

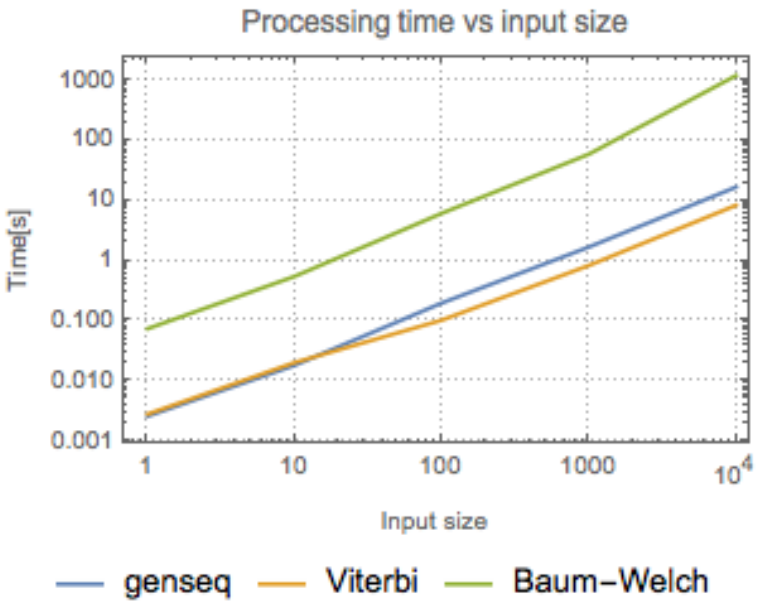


Figure 3: Processing time as a function of the input size. In all cases the processing time grow linearly with the input size.

6 Conclusions

- We were able to implement a HMM implementation in object oriented language Python.
- The proposed architecture is fairly simple, however functional
- We were able to replicate the results of the benchmark program
- The processing time grows linearly with the input size for all three algorithms

References

[1] Stratonovich, R.L. (1960). "Conditional Markov Processes". Theory of Probability and its Applications. 5 (2): 156–178. doi:10.1137/1105015.

[2] Baker, J. (1975). "The DRAGON system—An overview". IEEE Transactions on Acoustics, Speech, and Signal Processing. 23: 24–29. doi:10.1109/TASSP.1975.1162650.

[3] M. Bishop and E. Thompson (1986). "Maximum Likelihood Alignment of DNA Sequences". Journal of Molecular Biology. 190 (2): 159–165. doi:10.1016/0022-2836(86)90289-5. PMID 3641921.

[4] Richard Durbin; Sean R. Eddy; Anders Krogh; Graeme Mitchison (1999). Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press. ISBN 0-521-62971-3.

[5] Koller, D., & Friedman, N. (2013). Probabilistic Graphical Models. Journal of Chemical Information and Modeling (Vol. 53). http://doi.org/10.1017/CBO9781107415324.004

[6] Hendricks, E., Jannerup, O., & Sørensen, P. H. (2008). Linear Systems Control: Deterministic and Stochastic Methods. Springer.

[7] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org

[8] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[9] A C++ Implementation of Hidden Markov Model, Dekang Lin, 2003, http://stp.lingfil.uu.se/ starback-/hmm/