Hidden Markov Model

Phuc Dinh Hoang

*Abstract*—Hidden Markov Models (HMM) are stochastic approaches to model temporal and sequence data. They are particularly known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

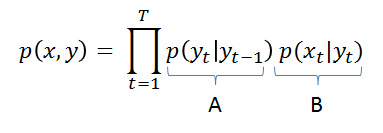
Keywords—Hidden Markov Model, Hidden Markov Algorithm, Baum-Welch learning algorithm, LearningAPI, .NET Core.

# Introduction

http://accord-framework.net/docs/images/hmm/hmm-tuple.pngIn simpler Markov models (for example - Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters, while in the hidden Markov model, the state is not directly visible, but the output (in the form of data or "token" in the following), dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states; this is also known as pattern theory, a topic of grammar induction. This model is implemented as part of LearningAPI Machine Learning Foundation running on top of .NET Core [1]. The rest of the report is organized as follows. Section II describes the algorithm in details and presents the implementation alongside with the unit test. Finally, the conclusion gives a summary of development of the hidden Markov model as.

# Methods

## Hidden Markov Algorithm

This project refers to the discrete-density version of the model. Dynamical systems of discrete nature assumed to be governed by a Markov chain emits a sequence of observable outputs. Under the Markov assumption, it is also assumed that the latest output depends only on the current state of the system. Such states are often not known from the observer when only the output values are observable. Assuming the Markov probability, the probability of any sequence of observations occurring when following a given sequence of states can be stated as:

in which the probabilities can be read as the probability of being currently in state given we just were in the state at the previous instant t-1, and the probability can be understood as the probability of observing at instant t given we are currently in the state . To compute those probabilities, we simple use two matrices A and B. The matrix A is the matrix of state probabilities: it gives the probabilities of jumping from one state to the other, and the matrix B is the matrix of observation probabilities, which gives the distribution density associated a given state . In the discrete case, B is really a matrix. In the continuous case, B is a vector of probability distributions. The overall model definition can then be stated by the tuple   
in which n is an integer representing the total number of states in the system, **A** is a matrix of transition probabilities, **B** is either a matrix of observation probabilities (in the discrete case) or a vector of probability distributions (in the general case) and **pi** is a vector of initial state probabilities determining the probability of starting in each of the possible states in the model.

Hidden Markov Models attempt to model such systems and allow, among other things,

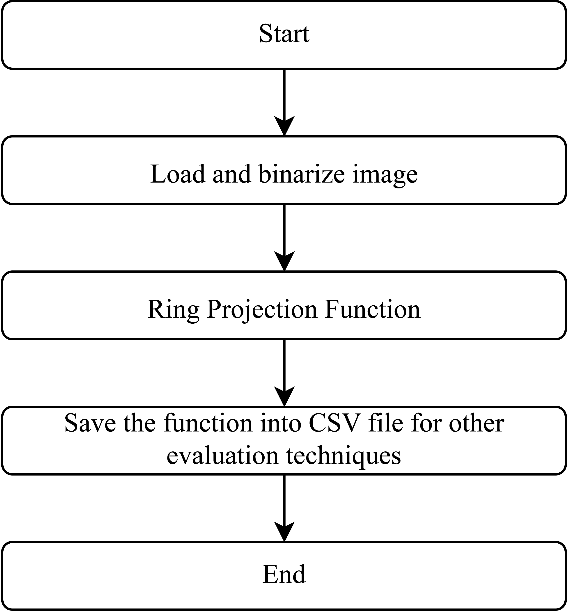
* To infer the most likely sequence of states that produced a given output sequence,
* Infer which will be the most likely next state (and thus predicting the next output),
* Calculate the probability that a given sequence of outputs originated from the system (allowing the use of hidden Markov models for sequence classification).

The “hidden” in Hidden Markov Models comes from the fact that the observer does not know in which state the system may be in, but has only a probabilistic insight on where it should be [2].

## Architecture

The program is divided into two main projects: HiddenMarkovLib and HiddenMarkovLibUnitTest. The implementation of the algorithm is created inside the HiddenMarkovAlgorithm.Cs and the second one is for testing the model by input sequences. The project is created based on .NET Core 2.0 using C# programming language.

The overview of the project is shown in Fig. . First, an input of sequences is created. Then a HMM will be created with Ergodic type or Forward type. An example of setting up and using the algorithm is illustrated in Fig. 2 and the briefly explain of HMM types is illustrated in Fig. 2 .



Input sequences

Get result as probabilities

Training the input sequences

Create Hidden Markov Model

Fig. 1. Architecture of the project

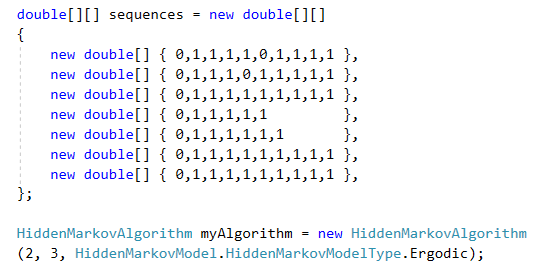


Fig. 2. Code snippet of input sequences

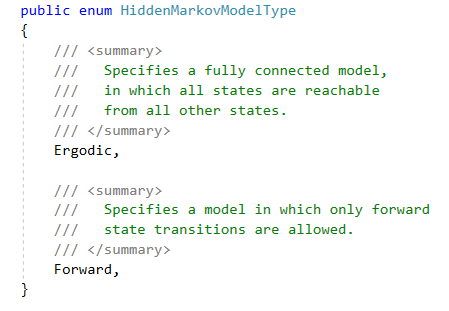


Fig. 3. Hidden Markov Model type

## Implemetation

### Hidden Markov Model Constructor

The algorithm is required to be implemented as a module of LearningAPI. The constructor of the HMM is shown below in Fig. 24. *Symbols* is the number of output symbols used for this model. *States* is the number of states for this model and *Type* is the topology which should be used by this model.

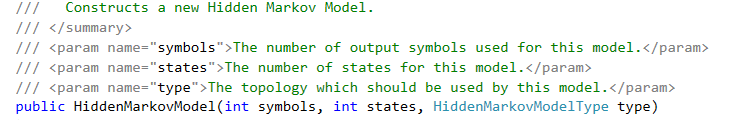


Fig. 1. Code Snippet of constructor of HiddenMarkovModel

### Baum-Welch learning algorithm

The Baum–Welch algorithm is used to find the unknown parameters of a hidden Markov model (HMM). It makes use of a forward-backward algorithm. A hidden Markov model describes the joint probability of a collection of "hidden" and observed discrete random variables. It relies on the assumption that the i-th hidden variable given the (i − 1)-th hidden variable is independent of previous hidden variables, and the current observation variables depend only on the current hidden state. The Baum–Welch algorithm uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden Markov model given a set of observed feature vectors. Let be a discrete hidden random variable with possible values (i.e. We assume there are states in total). We assume the is independent of time t, which leads to the definition of the time-independent stochastic transition matrix:

The initial state distribution (i.e. when t=1) is given by

.

The observation variables can take one of K possible values. We also assume the observation given the "hidden" state is time independent. The probability of a certain observation at time t for state is given by:

Taking into account all the possible values of and we obtain the N x K matrix where belongs to all the possible states and belongs to all the observations. An observation sequence is given by

Thus we can describe a hidden Markov chain by

The Baum–Welch algorithm finds a local maximum for

the HMM parameters θ that maximize the probability of the observation [3].

*2.a) Forward Procedure:*

Let , the probability of seeing the and being in state i at time t. This is found recursively:

1. ,

*2.b) Backward Procedure:*

Let , that is the probability of ending partial sequence given starting state i at time t. We calculate as,

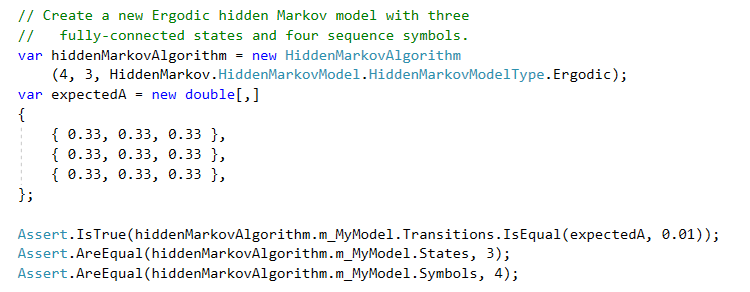
1. ,

## UnitTest

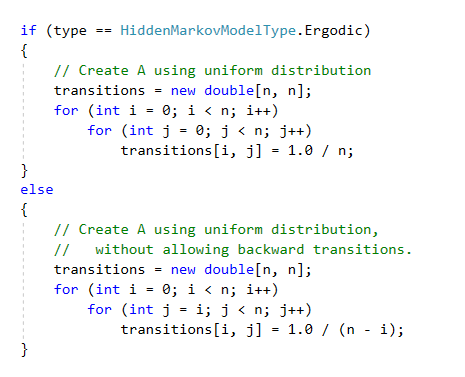
### Topology

#### ErgodicTest

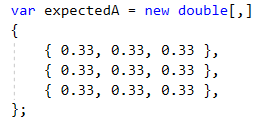
We create a new Ergodic hidden Markov model with three fully-connected states and four sequence symbols to test transition matrix A using uniform distribution as shown below



Within Ergodic hidden Markov model, the matrix A is created base on the number of state (three states) following algorithm which is shown below

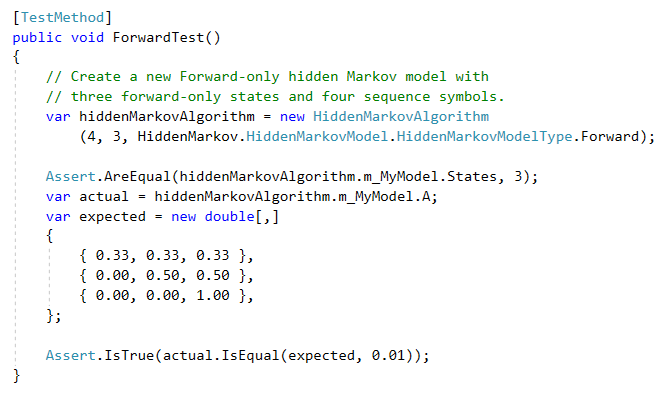


*n* is the number of state. *Transitions matrix* should be equal to a *new double [3, 3]* with a value of each element equal to *1.0 / 3 = 0.33.* So that, the expected transition matrix A after uniform distribution is:

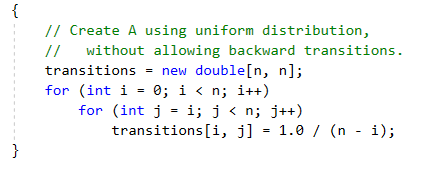


#### ForwardTest

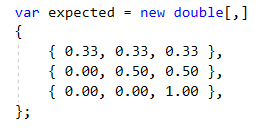
We create a new Forward-only hidden Markov model with three forward-only states and four sequence symbols to test transition matrix A using uniform distribution without allowing backward transitions as shown below



Within Forward hidden Markov model, the matrix A is created base on the number of state (three states) following algorithm which is shown below



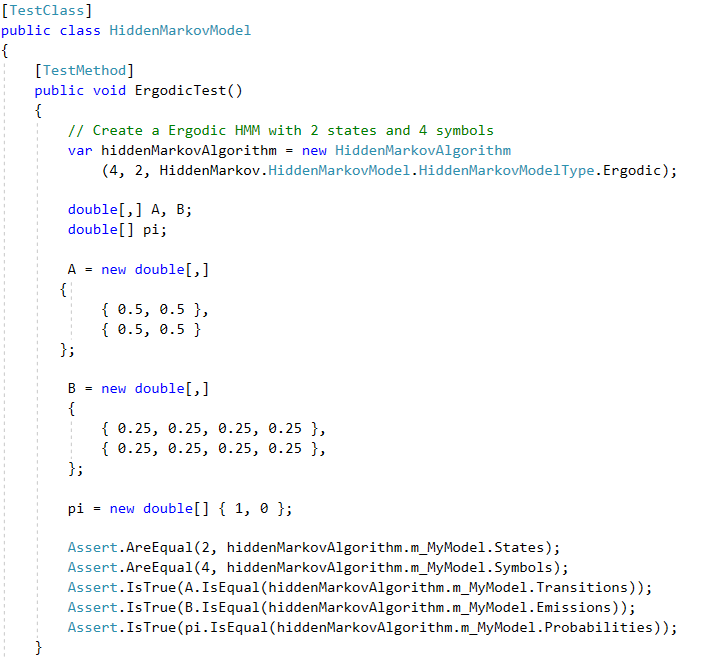
*n* is the number of state. So that, the expected transition matrix A after uniform distribution, without allowing backward transitions is:



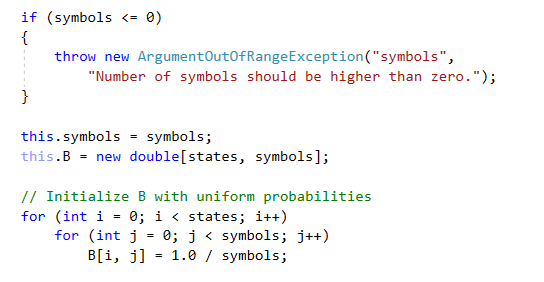
### HiddenMarkovModel

#### ErgodicTest

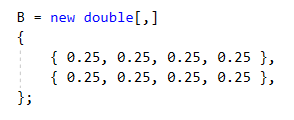
In this test, we create a new Ergodic hidden Markov model with four symbols and two states to check the matrix A, matrix B and pi.



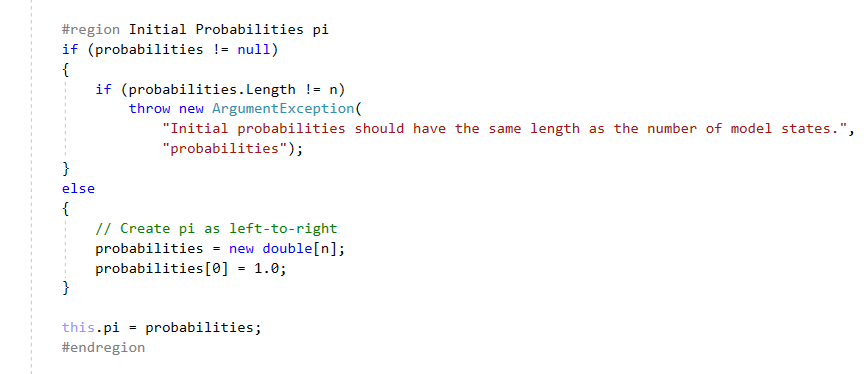
The *Emissions matrix* (or matrix B) is created based on number of states and symbols as below:



Number of states is 2 and number of symbols is 4. *Emissions matrix* should be equal to a *new double [2, 4]* with a value of each element equal to *1.0 / 4 = 0.25.* So that, the expected transition matrix B is:



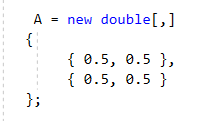
The *Initial Probabilities pi* is compute as below:



We have two states, so that *Initial Probabilities* equal to a *new double [2]* and its first element equal to *1.0*. The expected *Initial Probabilities pi* is:

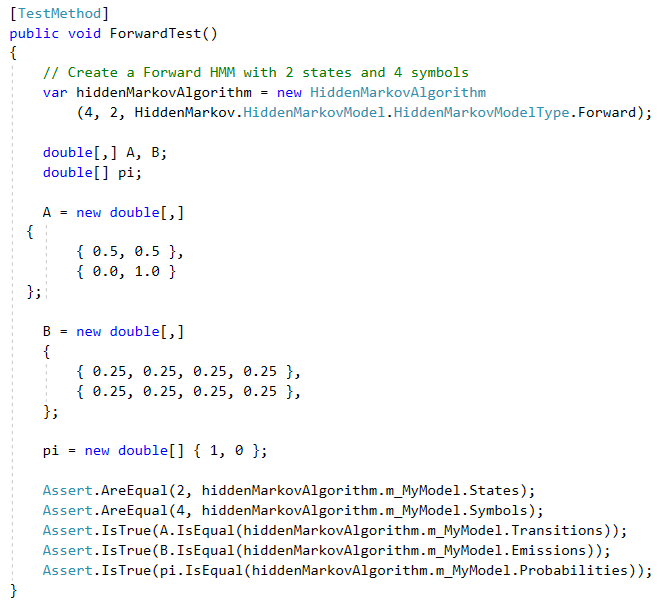


At last but not least, the expected *Transitions matrix* for Ergodic HMM as explained in above is:

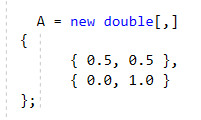


#### ForwardTest

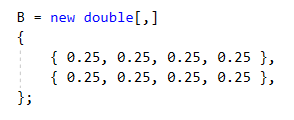
In this test, we create a new Forward hidden Markov model with four symbols and two states to check the matrix A, matrix B and pi.



The only difference between Ergodic and Forward HMM is the calculation of *Transition matrix*. As explain above, the expected matrix A for Forward HMM is:



The expected matrix B is:



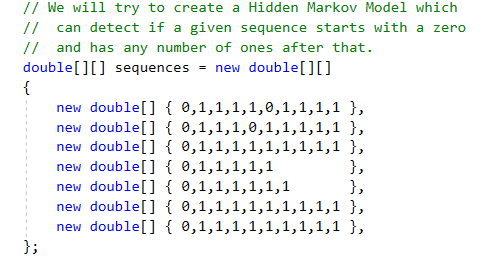
The expected pi is:



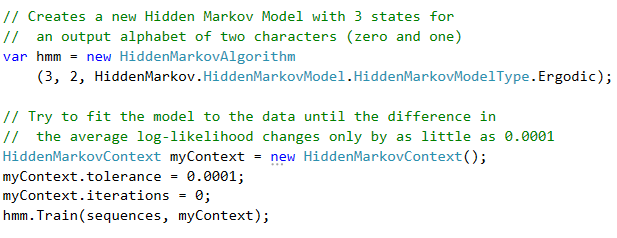
#### LearnTest

In this test, we try to predict to probabilities of a given sequence starts with a zero and has any number of ones after that.

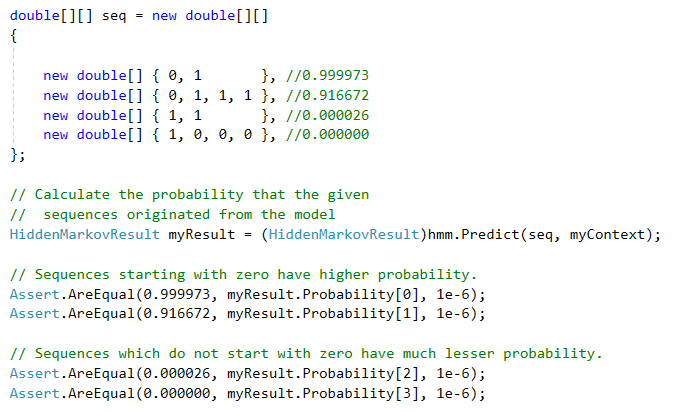
First, we create an input sequences to train the HMM as below. All of this sequences begin with a zero and many ones after that.



Then we create an Ergodic hidden Markov model with three fully-connected states and two sequence symbols. We try to fit the model to the data until the difference in the average log-likelihood changes only by as little as 0.0001 and train the model with the input data.



After that, we create an observation input to check the probabilities of each sequence. The expectation output should be higher probabilities for the sequences begin with a zero and lesser probabilities for sequences which do not start with a zero.



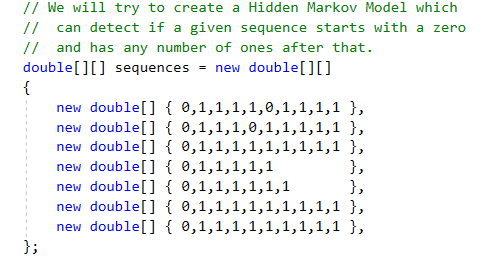
Sequences begin with a zero and has one afterward will have higher probability. The first sequence: *{ 0, 1}* has the highest probability of *0.999973*. The second sequence: *{ 0, 1, 1, 1 }* has the high probability of *0.916672.*

Sequences which do not start with a zero have much lesser probability. The sequence: *{ 1, 1}* has the probability of *0.000026*. The second sequence: *{ 1, 0, 0, 0 }* has the lowest probability of *0.00000.*

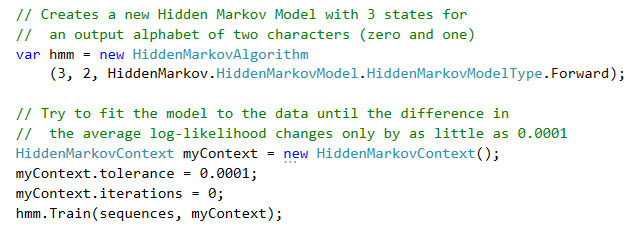
#### LearnTest2

In this test, we also try to predict to probabilities of a given sequence starts with a zero and has any number of ones after that but with Forward hidden Markov model.

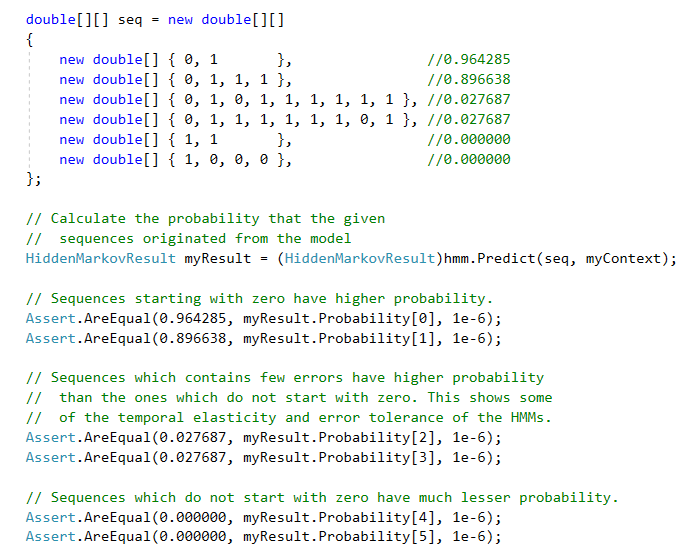
First, we create an input sequences to train the HMM as below. All of this sequences begin with a zero and many ones after that.



Then we create an Forward hidden Markov model with three fully-connected states and two sequence symbols. We try to fit the model to the data until the difference in the average log-likelihood changes only by as little as 0.0001 and train the model with the input data.



After that, we create an observation input to check the probabilities of each sequence. The expectation output should be higher probabilities for the sequences begin with a zero, lesser probabilities for sequences which do not start with a zero and higher probability of sequences contains few errors than sequences which do not start with a zero.



Sequences begin with a zero and has one afterward will have higher probability. The first sequence: *{ 0, 1}* has the highest probability of *0.964285*. The second sequence: *{ 0, 1, 1, 1 }* has the high probability of *0.896638.*

Sequences which contains few errors have higher probability than the ones which do not start with a zero. This shows some of the temporal elasticity and error tolerance of the HMMs. The sequence: *{ 0, 1, 0, 1, 1, 1, 1, 1, 1 }* has the probability of *0.027687.* The sequence: *{ 0, 1, 1, 1, 1, 1, 1, 0, 1 }* has the probability of *0.027687.*

Sequences which do not start with a zero have much lesser probability. The sequence: *{ 1, 1 }* has the probability of *0.000000*. The sequence: *{ 1, 0, 0, 0 }* has the lowest probability of *0.00000.*

# Discussion and Conclusion

The goal of the project is the implementation of hidden Markov Model algorithm for hidden Markov model as a module for LearningAPI. The module takes the input sequences as training data and train them by using Ergodic hidden Markov model or Forward-only hidden Markov model and return the possibilities of the observation sequences. The algorithm used in the hidden Markov model is Baum-Welch learning algorithm. It includes the Baum-Welch forward algorithm and Baum-Welch backward algorithm.

# References

|  |  |
| --- | --- |
| [1] | Open Source, "UniversityOfAppliedSciencesFrankfurt/LearningApi: Machine Learning foundation on top of .NET Core," 2019. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi. [Accessed 01 03 2019]. |
| [2] | Open Source, Accord farmwork, 2019. [Online]. Available: http://accord-framework.net/docs/html/T\_Accord\_Statistics\_Models\_Markov\_HiddenMarkovModel.htm. [Accessed 01 03 2019]. |
| [3] | Bilmes, Jeff A. (1998). A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Berkeley, CA: International Computer Science Institute. pp. 7–13. |