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.

I. Introduction

In 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.

II. Methods

A. 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:

$$p(x,y) = \prod_{t=1}^{T} \underbrace{p(y_t|y_{t-1})}_{A} \underbrace{p(x_t|y_t)}_{B}$$

in which the probabilities $p(y_t|y_{t-1})$ can be read as the probability of being currently in state y_t given we just were in the state y_{t-1} at the previous instant t-1, and the probability $p(x_t|y_t)$ can be understood as the probability of observing x_t at instant t given we are currently in the state y_t. 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 $p(y_t|y_{t-1})$ of jumping from one state to the other, and the matrix B is the matrix of observation probabilities, which gives the distribution density $p(x_t|y_t)$ associated a given state y_t. 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,

$$\lambda = (n, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$$

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].

B. 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 .

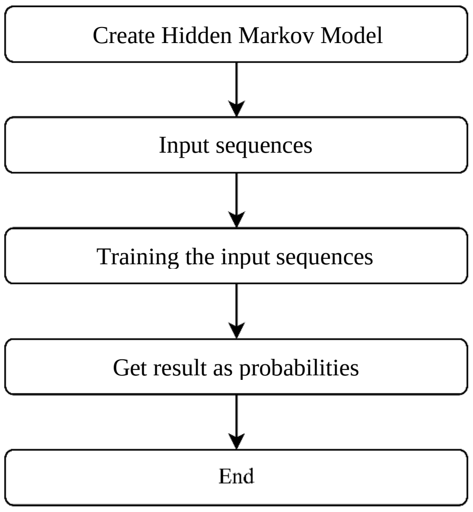
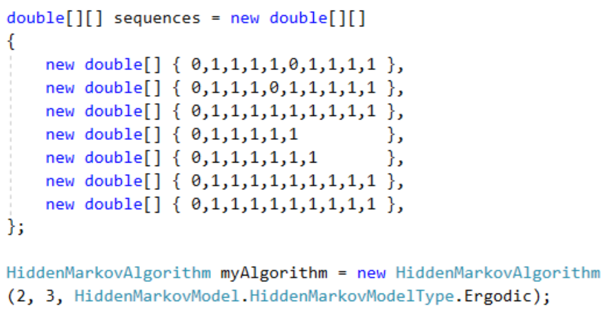
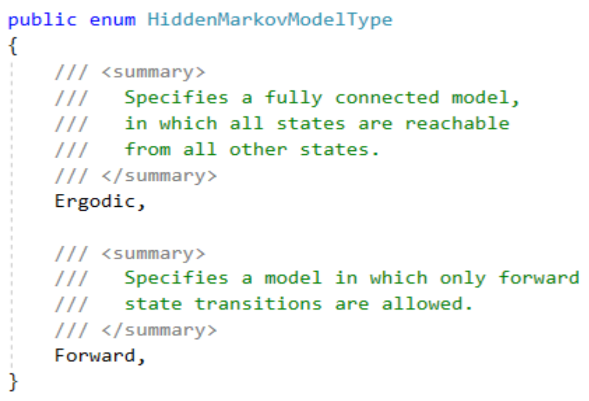

Create Hidden Markov Model
Input sequences
Training the input sequences
Get result as probabilities
End

Fig. 1. Architecture of the project



```
double[][] sequences = new double[][]
{
    new double[] { 0,1,1,1,1,0,1,1,1,1 },
    new double[] { 0,1,1,1,0,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1         },
    new double[] { 0,1,1,1,1,1,1       },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
};

HiddenMarkovAlgorithm myAlgorithm = new HiddenMarkovAlgorithm
(2, 3, HiddenMarkovModel.HiddenMarkovModelType.Ergodic);
```

Fig. 2. Code snippet of input sequences



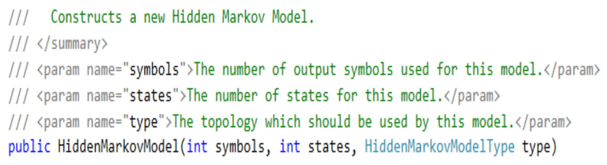
```
public enum HiddenMarkovModelType
{
    /// <summary>
    ///   Specifies a fully connected model,
    ///   in which all states are reachable
    ///   from all other states.
    /// </summary>
    Ergodic,

    /// <summary>
    ///   Specifies a model in which only forward
    ///   state transitions are allowed.
    /// </summary>
    Forward,
}
```

Fig. 3. Hidden Markov Model type

C. Implemetation

1) 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.



```
///   Constructs a new Hidden Markov Model.
/// </summary>
/// <param name="symbols">The number of output symbols used for this model.</param>
/// <param name="states">The number of states for this model.</param>
/// <param name="type">The topology which should be used by this model.</param>
public HiddenMarkovModel(int symbols, int states, HiddenMarkovModelType type)
```

Fig. 1. Code Snippet of constructor of `HiddenMarkovModel`

2) 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 X_t be a discrete hidden random variable with N possible values (i.e. We assume there are N states in total). We assume the $P(X_t|X_{t-1})$ is independent of time t, which leads to the definition of the time-independent stochastic transition matrix:

$$A = \{a_{ij}\} = P(X_t = j|X_{t-1} = i)$$

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

$$\pi_i = P(X_1 = i).$$

The observation variables Y_t 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 y_i at time t for state $X_t = j$ is given by:

$$b_j(y_i) = P(Y_t = y_i|X_t = j)$$

Taking into account all the possible values of Y_t and X_t we obtain the N x K matrix $B = \{b_j(y_i)\}$ where b_j belongs to all the possible states and y_i belongs to all the observations. An observation sequence is given by

$$Y = (Y_1 = y_1, Y_2 = y_2, \ldots, Y_T = y_T)$$

Thus we can describe a hidden Markov chain by

$$\theta = (A, B, \pi)$$

The Baum–Welch algorithm finds a local maximum for

$$\theta^* = \arg max_\theta \, P(Y|\theta)$$

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

2.a) Forward Procedure:

Let $\propto_i (t) = P(Y_1 = y_1, \ldots, Y_t = y_t, X_t = i|\theta)$, the probability of seeing the $y_1, y_2, \ldots, y_t$ and being in state i at time t. This is found recursively:

1. $\propto_i (1) = \pi_i b_i(y_1)$,
2. $\propto_i (t+1) = b_i(y_{t+1}) \sum_{j=1}^{N} a_j(t) a_{ij}$

2.b) Backward Procedure:

Let $\beta_i(t) = P(Y_{t+1} = y_{t+1}, \ldots, Y_t = y_t, X_t = i|\theta)$, that is the probability of ending partial sequence $y_{t+1}, \ldots, y_T$ given starting state i at time t. We calculate $\beta_i(t)$ as,

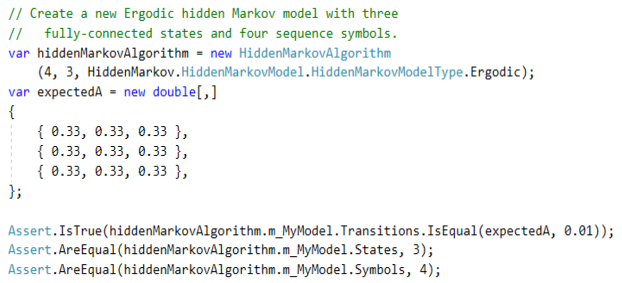
1. $\beta_i(T) = 1$,
2. $\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_{t+1})$

D. *UnitTest*

1) *Topology*

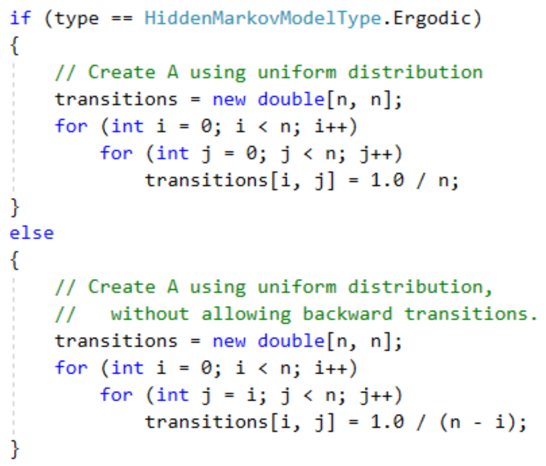
a) ErgodicTest

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



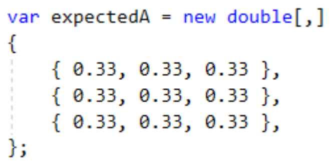
```
// Create a new Ergodic hidden Markov model with three
//   fully-connected states and four sequence symbols.
var hiddenMarkovAlgorithm = new HiddenMarkovAlgorithm
    (4, 3, HiddenMarkov.HiddenMarkovModel.HiddenMarkovModelType.Ergodic);
var expectedA = new double[,]
{
    { 0.33, 0.33, 0.33 },
    { 0.33, 0.33, 0.33 },
    { 0.33, 0.33, 0.33 },
};

Assert.IsTrue(hiddenMarkovAlgorithm.m_MyModel.Transitions.IsEqual(expectedA, 0.01));
Assert.AreEqual(hiddenMarkovAlgorithm.m_MyModel.States, 3);
Assert.AreEqual(hiddenMarkovAlgorithm.m_MyModel.Symbols, 4);
```

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



```
if (type == HiddenMarkovModelType.Ergodic)
{
    // Create A using uniform distribution
    transitions = new double[n, n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            transitions[i, j] = 1.0 / n;
}
else
{
    // Create A using uniform distribution,
    //   without allowing backward transitions.
    transitions = new double[n, n];
    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++)
            transitions[i, j] = 1.0 / (n - i);
}
```

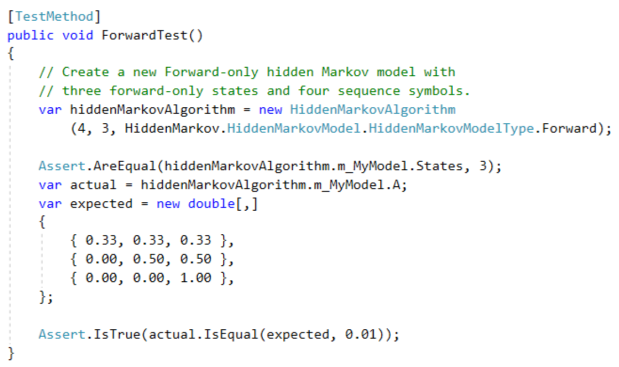
n is the number of state. *Transitions matrix* should be equal to a *new double [3, 3]* with a value of each element equal to *1.0 / 3 = 0.33*. So that, the expected transition matrix A after uniform distribution is:



```
var expectedA = new double[,]
{
    { 0.33, 0.33, 0.33 },
    { 0.33, 0.33, 0.33 },
    { 0.33, 0.33, 0.33 },
};
```

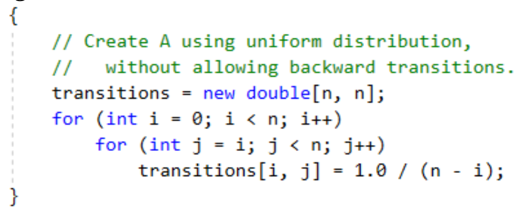
b) ForwardTest

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



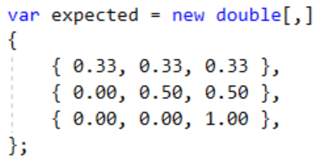
```
[TestMethod]
public void ForwardTest()
{
    // Create a new Forward-only hidden Markov model with
    // three forward-only states and four sequence symbols.
    var hiddenMarkovAlgorithm = new HiddenMarkovAlgorithm
        (4, 3, HiddenMarkov.HiddenMarkovModel.HiddenMarkovModelType.Forward);

    Assert.AreEqual(hiddenMarkovAlgorithm.m_MyModel.States, 3);
    var actual = hiddenMarkovAlgorithm.m_MyModel.A;
    var expected = new double[,]
    {
        { 0.33, 0.33, 0.33 },
        { 0.00, 0.50, 0.50 },
        { 0.00, 0.00, 1.00 },
    };

    Assert.IsTrue(actual.IsEqual(expected, 0.01));
}
```

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



```
{
    // Create A using uniform distribution,
    //   without allowing backward transitions.
    transitions = new double[n, n];
    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++)
            transitions[i, j] = 1.0 / (n - i);
}
```

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

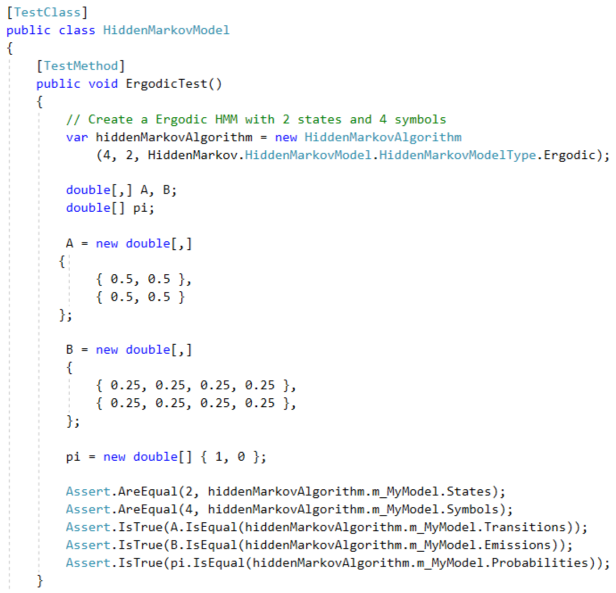


```
var expected = new double[,]
{
    { 0.33, 0.33, 0.33 },
    { 0.00, 0.50, 0.50 },
    { 0.00, 0.00, 1.00 },
};
```

2) *HiddenMarkovModel*

a) 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.



```
[TestClass]
public class HiddenMarkovModel
{
    [TestMethod]
    public void ErgodicTest()
    {
        // Create a Ergodic HMM with 2 states and 4 symbols
        var hiddenMarkovAlgorithm = new HiddenMarkovAlgorithm
            (4, 2, HiddenMarkov.HiddenMarkovModel.HiddenMarkovModelType.Ergodic);

        double[,] A, B;
        double[] pi;

        A = new double[,]
       {
            { 0.5, 0.5 },
            { 0.5, 0.5 }
       };

        B = new double[,]
        {
            { 0.25, 0.25, 0.25, 0.25 },
            { 0.25, 0.25, 0.25, 0.25 },
        };

        pi = new double[] { 1, 0 };

        Assert.AreEqual(2, hiddenMarkovAlgorithm.m_MyModel.States);
        Assert.AreEqual(4, hiddenMarkovAlgorithm.m_MyModel.Symbols);
        Assert.IsTrue(A.IsEqual(hiddenMarkovAlgorithm.m_MyModel.Transitions));
        Assert.IsTrue(B.IsEqual(hiddenMarkovAlgorithm.m_MyModel.Emissions));
        Assert.IsTrue(pi.IsEqual(hiddenMarkovAlgorithm.m_MyModel.Probabilities));
    }
```

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

```
if (symbols <= 0)
{
    throw new ArgumentOutOfRangeException("symbols",
        "Number of symbols should be higher than zero.");
}

this.symbols = symbols;
this.B = new double[states, symbols];

// Initialize B with uniform probabilities
for (int i = 0; i < states; i++)
    for (int j = 0; j < symbols; j++)
        B[i, j] = 1.0 / symbols;
```

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:

```
B = new double[,]
{
    { 0.25, 0.25, 0.25, 0.25 },
    { 0.25, 0.25, 0.25, 0.25 },
};
```

The *Initial Probabilities pi* is compute as below:

```
#region Initial Probabilities pi
if (probabilities != null)
{
    if (probabilities.Length != n)
        throw new ArgumentException(
            "Initial probabilities should have the same length as the number of model states.",
            "probabilities");
}
else
{
    // Create pi as left-to-right
    probabilities = new double[n];
    probabilities[0] = 1.0;
}

this.pi = probabilities;
#endregion
```

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:

```
pi = new double[] { 1, 0 };
```

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

```
  A = new double[,]
{
    { 0.5, 0.5 },
    { 0.5, 0.5 }
};
```

b) 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.

```
[TestMethod]
public void ForwardTest()
{
    // Create a Forward HMM with 2 states and 4 symbols
    var hiddenMarkovAlgorithm = new HiddenMarkovAlgorithm
        (4, 2, HiddenMarkov.HiddenMarkovModel.HiddenMarkovModelType.Forward);

    double[,] A, B;
    double[] pi;

    A = new double[,]
  {
        { 0.5, 0.5 },
        { 0.0, 1.0 }
  };

    B = new double[,]
    {
        { 0.25, 0.25, 0.25, 0.25 },
        { 0.25, 0.25, 0.25, 0.25 },
    };

    pi = new double[] { 1, 0 };

    Assert.AreEqual(2, hiddenMarkovAlgorithm.m_MyModel.States);
    Assert.AreEqual(4, hiddenMarkovAlgorithm.m_MyModel.Symbols);
    Assert.IsTrue(A.IsEqual(hiddenMarkovAlgorithm.m_MyModel.Transitions));
    Assert.IsTrue(B.IsEqual(hiddenMarkovAlgorithm.m_MyModel.Emissions));
    Assert.IsTrue(pi.IsEqual(hiddenMarkovAlgorithm.m_MyModel.Probabilities));
}
```

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:

```
  A = new double[,]
{
    { 0.5, 0.5 },
    { 0.0, 1.0 }
};
```

The expected matrix B is:

```
B = new double[,]
{
    { 0.25, 0.25, 0.25, 0.25 },
    { 0.25, 0.25, 0.25, 0.25 },
};
```

The expected pi is:

```
pi = new double[] { 1, 0 };
```

c) 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.

```
// We will try to create a Hidden Markov Model which
//  can detect if a given sequence starts with a zero
//  and has any number of ones after that.
double[][] sequences = new double[][]
{
    new double[] { 0,1,1,1,1,0,1,1,1,1 },
    new double[] { 0,1,1,1,0,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1         },
    new double[] { 0,1,1,1,1,1,1       },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
};
```

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.

```
// Creates a new Hidden Markov Model with 3 states for
//  an output alphabet of two characters (zero and one)
var hmm = new HiddenMarkovAlgorithm
    (3, 2, HiddenMarkov.HiddenMarkovModel.HiddenMarkovModelType.Ergodic);

// Try to fit the model to the data until the difference in
//  the average log-likelihood changes only by as little as 0.0001
HiddenMarkovContext myContext = new HiddenMarkovContext();
myContext.tolerance = 0.0001;
myContext.iterations = 0;
hmm.Train(sequences, myContext);
```

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.

```
double[][] seq = new double[][]
{
    new double[] { 0, 1       }, //0.999973
    new double[] { 0, 1, 1, 1 }, //0.916672
    new double[] { 1, 1       }, //0.000026
    new double[] { 1, 0, 0, 0 }, //0.000000
};

// Calculate the probability that the given
//  sequences originated from the model
HiddenMarkovResult myResult = (HiddenMarkovResult)hmm.Predict(seq, myContext);

// Sequences starting with zero have higher probability.
Assert.AreEqual(0.999973, myResult.Probability[0], 1e-6);
Assert.AreEqual(0.916672, myResult.Probability[1], 1e-6);

// Sequences which do not start with zero have much lesser probability.
Assert.AreEqual(0.000026, myResult.Probability[2], 1e-6);
Assert.AreEqual(0.000000, myResult.Probability[3], 1e-6);
```

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.*

d) 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.

```
// We will try to create a Hidden Markov Model which
//  can detect if a given sequence starts with a zero
//  and has any number of ones after that.
double[][] sequences = new double[][]
{
    new double[] { 0,1,1,1,1,0,1,1,1,1 },
    new double[] { 0,1,1,1,0,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1         },
    new double[] { 0,1,1,1,1,1,1       },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
    new double[] { 0,1,1,1,1,1,1,1,1,1 },
};
```

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.

```
// Creates a new Hidden Markov Model with 3 states for
//  an output alphabet of two characters (zero and one)
var hmm = new HiddenMarkovAlgorithm
    (3, 2, HiddenMarkov.HiddenMarkovModel.HiddenMarkovModelType.Forward);

// Try to fit the model to the data until the difference in
//  the average log-likelihood changes only by as little as 0.0001
HiddenMarkovContext myContext = new HiddenMarkovContext();
myContext.tolerance = 0.0001;
myContext.iterations = 0;
hmm.Train(sequences, myContext);
```

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.

```
double[][] seq = new double[][]
{
    new double[] { 0, 1       },                    //0.964285
    new double[] { 0, 1, 1, 1 },                    //0.896638
    new double[] { 0, 1, 0, 1, 1, 1, 1, 1, 1 },     //0.027687
    new double[] { 0, 1, 1, 1, 1, 1, 1, 0, 1 },     //0.027687
    new double[] { 1, 1       },                    //0.000000
    new double[] { 1, 0, 0, 0 },                    //0.000000
};

// Calculate the probability that the given
//  sequences originated from the model
HiddenMarkovResult myResult = (HiddenMarkovResult)hmm.Predict(seq, myContext);

// Sequences starting with zero have higher probability.
Assert.AreEqual(0.964285, myResult.Probability[0], 1e-6);
Assert.AreEqual(0.896638, myResult.Probability[1], 1e-6);

// Sequences which contains few errors have higher probability
//  than the ones which do not start with zero. This shows some
//  of the temporal elasticity and error tolerance of the HMMs.
Assert.AreEqual(0.027687, myResult.Probability[2], 1e-6);
Assert.AreEqual(0.027687, myResult.Probability[3], 1e-6);

// Sequences which do not start with zero have much lesser probability.
Assert.AreEqual(0.000000, myResult.Probability[4], 1e-6);
Assert.AreEqual(0.000000, myResult.Probability[5], 1e-6);
```

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.*

III. 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.

IV. 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.