Jahmm

An implementation of Hidden Markov Models in Java

Information Management Lab.
Dept. of Industrial Engineering
Seoul National Univ.

Introduction

- Jahmm (pronounced "jam"), is
  - a Java implementation of Hidden Markov Model (HMM) related algorithms.
  - been designed to be easy to use (e.g. simple things are simple to program) and general purpose.
  - available under the new BSD license.
- This library is reasonably efficient
  - meaning that the complexity of the implementation of the algorithms involved is that given by the theory.
- However, when a choice must be made between code readability and efficiency, readability has been chosen. It is thus ideal in research (because algorithms can easily be modified) and as an academic tool (students can quickly get interesting results).
- It gives an implementation of the Viterbi, Forward-Backward, Baum-Welch and K-Means algorithms, among others.

download

- http://code.google.com/p/jahmm/

Algorithm

- **Learning algorithm**
  - **k-Means clustering algorithm**
  - **Baum-Welch algorithm**
- **Measuring algorithm**
  - **Kullback-Leibler distance measure**

Input/Output

- Observation sequences

```
Obs11 ; Obs12 ; Obs13 ; Obs14 ;

Obs21 ; Obs22 ; Obs23 ; Obs24 ; Obs25 ;
```

  - Supported observation types are integer, reals and real vectors.

```
2 ; 3 ; 1 ; 3 ;
1 ; 2 ; 3 ; 1 ; 1 ;
```

```
[ 1.1 2.2 ] ; [ 4.4 5.5 ] ; [ 4.3 6. ] ; [ 7.7 8.8 ] ;
[ 0.5 1.5 ] ; [ 1.5 2.5 ] ; [ 4.5 5.5 ] ; [ 8. 8. ] ; [ 7. 8. ] ;
```

- HMM
  - HMMs can also be encoded as a textual file. Its syntax is quite straightforward, so it will be explained by means of an example.

HMM output example

```
1 Hmm v1.0
2
3 NbStates 5
4
5 State
6 Pi 0.1
7 A 0.1 0.2 0.3 0.4 0.0
8 IntegerOPDF [0.4 0.6 ]
9
10 State
11 Pi 0.3
12 A 0.2 0.2 0.2 0.2 0.2
13 IntegerOPDF [0.5 0.5 ]
14
15 State
16 Pi 0.2
17 A 0.2 0.2 0.2 0.2 0.2
18 IntegerOPDF [0.5 0.5 ]
19
20 State
21 Pi 0.2
22 A 0.2 0.2 0.2 0.2 0.2
23 IntegerOPDF [0.5 0.5 ]
24
25 State
26 Pi 0.2
27 A 0.2 0.2 0.2 0.2 0.2
28 IntegerOPDF [0.5 0.5 ]
```

- the number of states of the HMM described.
- state descriptions
- The probability that this state is an initial state.
- The state-to-state transition probabilities.
- A description of the observation distribution function.

Building a HMM

```
Hmm <ObservationInteger > hmm =
    new Hmm <ObservationInteger >(2, new OpdfIntegerFactory (2) );
```

```
hmm.setPi (0, 0.95);
hmm.setPi (1, 0.05);
hmm.setOpdf (0, new OpdfInteger (new double [] {0.95 , 0.05}));
hmm.setOpdf (1, new OpdfInteger (new double [] {0.2 , 0.8}));
hmm.setAij (0, 1, 0.05);
hmm.setAij (0, 0, 0.95);
hmm.setAij (1, 0, 0.1);
hmm.setAij (1, 1, 0.9);
```

Learning

```
KMeansLearner <ObservationInteger > kml =
new KMeansLearner <ObservationInteger >(3, new OpdfIntegerFactory (4) ,
sequences);

Hmm <ObservationInteger > initHmm = kml.iterate ();
```

```
OpdfIntegerFactory factory = new OpdfIntegerFactory (4);
BaumWelchLearner <ObservationInteger > bwl =
new BaumWelchLearner <ObservationInteger >(3, factory);

Hmm <ObservationInteger > learntHmm = bwl.learn(initHmm , sequences);
```

```
State 0
  Pi: 0.9710675169906993
  Aij: 0.949 0.051
  Opdf: Discrete distribution --- OK 0.951, LOSS 0.049

State 1
  Pi: 0.02893248300929998
  Aij: 0.097 0.903
  Opdf: Discrete distribution --- OK 0.2, LOSS 0.8
```

Data file reading

```
Reader reader = new FileReader (" test.seq ");
List <List <ObservationVector >> seqs =
ObservationSequencesReader.readSequences (new ObservationVectorReader (),
reader);
reader.close ();
```

```
KMeansLearner <ObservationVector > kml =
new KMeansLearner <ObservationVector >(3, new OpdfMultiGaussianFactory
(2) , seqs);
Hmm <ObservationVector > fittedHmm = kml.learn ();
```

Measuring distance between two HMMs

```
KullbackLeiblerDistanceCalculator klc =
                                        new KullbackLeiblerDistanceCalculator();
```

```
klc.distance(hmm1, hmm2);
```

Computing the probability of a sequence

```
hmm.probability(testSequence)
```