HMMPy

1.0

Generated by Doxygen 1.6.3

Thu Feb 17 18:25:18 2011

Contents

1 Namespace Index **1**

1.1 Package List 1

2 Class Index **3**

2.1 Class List 3

3 Namespace Documentation **5**

3.1 Package hmm 5

3.1.1 Detailed Description 5

3.1.2 Function Documentation 5

3.1.2.1 backward 5

3.1.2.2 baum_welch 6

3.1.2.3 forward 6

3.1.2.4 symbol_index 6

3.1.2.5 viterbi 6

4 Class Documentation **7**

4.1 hmm.HMM Class Reference 7

4.1.1 Detailed Description 7

4.1.2 Member Function Documentation 8

4.1.2.1 __init__ 8

4.2 hmm.HMM_Classifier Class Reference 9

4.2.1 Detailed Description 9

4.2.2 Member Function Documentation 9

4.2.2.1 __init__ 9

4.2.2.2 add_neg_hmm 9

4.2.2.3 add_pos_hmm 9

4.2.2.4 classify 9

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

hmm 5

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

hmm.HMM 7
hmm.HMM_Classifier 9

Chapter 3

Namespace Documentation

3.1 Package hmm

Classes

- class HMM_Classifier
- class HMM

Functions

- def symbol_index
- def forward
- def backward
- def viterbi
- def baum_welch
- def **dishonest_casino_test**

3.1.1 Detailed Description

```
Python module for creating, training and applying hidden
Markov models to discrete or continuous observations.
Author: Michael Hamilton,  hamiltom@cs.colostate.edu
Theoretical concepts obtained from Rabiner, 1989.
```

3.1.2 Function Documentation

3.1.2.1 def hmm.backward (*hmm*, *Obs*, *c* = `None`)

```
Calculate the probability of a partial observation sequence
from t+1 to T, given some state t.
Obs: observation sequence
hmm: model
c: the scaling coefficients from forward algorithm
returns: B_t(i)
```

3.1.2.2 def hmm.baum_welch (*hmm*, *Obs_seqs*, *args*)

```
EM algorithm to update Pi, A, and B for the HMM
:Parameters:
  - 'hmm' - hmm model to train
  - 'Obs_seqs' - list of observation sequences to train over
:Return:
  a trained hmm

:Keywords:
  - 'epochs' - number of iterations to perform EM, default is 20
  - 'val_set' - validation data set, not required but recommended to prevent over-fitting
  - 'updatePi' - flag to update initial state probabilities
  - 'updateA' - flag to update transition probabilities, default is True
  - 'updateB' - flag to update observation emission probabilites for discrete types, default is True
  - 'scaling' - flag to scale probabilities (log scale), default is True
  - 'graph' - flag to plot log-likelihoods of the training epochs, default is False
  - 'normUpdate' - flag to use 1 / -(normed log-likelihood) contribution for each observation
                   sequence when updating model parameters, default if False
  - 'fname' - file name to save plot figure, default is ll.eps
  - 'verbose' - flag to print training times and log likelihoods for each training epoch, default is false
```

3.1.2.3 def hmm.forward (*hmm*, *Obs*, *scaling* = `True`)

```
Calculate the probability of an observation sequence, Obs,
given the model, P(Obs|hmm).
Obs: observation sequence
hmm: model
returns: P(Obs|hmm)
```

3.1.2.4 def hmm.symbol_index (*hmm*, *Obs*)

```
Converts an obeservation symbol sequence into a sequence
of indices for accessing distribution matrices.
```

3.1.2.5 def hmm.viterbi (*hmm*, *Obs*, *scaling* = `True`)

```
Calculate P(Q|Obs, hmm) and yield the state sequence Q* that
maximizes this probability.
Obs: observation sequence
hmm: model
```

Chapter 4

Class Documentation

4.1 hmm.HMM Class Reference

Public Member Functions

- def __init__
- def **__repr__**

Public Attributes

- **N**
- **V**
- **M**
- **symbol_map**
- **A**
- **B**
- **F**
- **Pi**
- **Labels**

4.1.1 Detailed Description

```
Creates and maintains a hidden Markov model.  This version assumes the every state can be
reached DIRECTLY from any other state (ergodic).  This, of course, excludes the start state.
Hence the state transition matrix, A, must be N X N .  The observable symbol probability
distributions are represented by an N X M matrix where M is the number of observation
symbols.

             |a_11 a_12 ... a_1N|                   |b_11 b_12 ... b_1M|
             |a_21 a_22 ... a_2N|                   |b_21 b_22 ... b_2M|
         A = | .    .        .  |               B = | .    .        .  |
             | .         .   .  |                   | .         .   .  |
             |a_N1 a_N2 ... a_NN|                   |b_N1 b_N2 ... b_NM|

       a_ij = P(q_t = S_j|q_t-1 = S_i)       b_ik = P(v_k at t|q_t = S_i)
    where q_t is state at time t and v_k is k_th symbol of observation sequence
```

4.1.2 Member Function Documentation

4.1.2.1 def hmm.HMM.__init__ (*self*, *n_states* = 1, *args*)

```
:Keywords:
  - 'n_states' - number of hidden states
  - 'V' - list of all observable symbols
  - 'A' - transition matrix
  - 'B' - observable symbol probability distribution
  - 'D' - dimensionality of continuous observations
  - 'F' - Fixed emission probabilities for the given state ( dict: i -> numpy.array( [n_states] ),
  where i is the state to hold fixed.
```

The documentation for this class was generated from the following file:

- hmm.py

4.2 hmm.HMM_Classifier Class Reference

Public Member Functions

- def __init__
- def classify
- def add_pos_hmm
- def add_neg_hmm

Public Attributes

- **neg_hmm**
- **pos_hmm**

4.2.1 Detailed Description

```
A binary hmm classifier that utilizes two hmms: one corresponding
to the positive activity and one corresponding to the negative
activity.
```

4.2.2 Member Function Documentation

4.2.2.1 def hmm.HMM_Classifier.__init__ (*self*, *args*)

```
:Keywords:
  - 'neg_hmm' - hmm corresponding to negative activity
  - 'pos_hmm' - hmm corresponding to positive activity
```

4.2.2.2 def hmm.HMM_Classifier.add_neg_hmm (*self*, *neg_hmm*)

```
Add the hmm corresponding to negative
activity.  Replaces current negative hmm, if it exists.
```

4.2.2.3 def hmm.HMM_Classifier.add_pos_hmm (*self*, *pos_hmm*)

```
Add the hmm corresponding to positive
activity.  Replaces current positive hmm, if it exists.
```

4.2.2.4 def hmm.HMM_Classifier.classify (*self*, *sample*)

```
Classification is performed by calculating the
log odds for the positive activity.  Since the hmms
return a log-likelihood (due to scaling)
of the corresponding activity, the difference of
the two log-likelihoods is the log odds.
```

The documentation for this class was generated from the following file:

- hmm.py

Index

__init__
    hmm::HMM, 8
    hmm::HMM_Classifier, 9

add_neg_hmm
    hmm::HMM_Classifier, 9
add_pos_hmm
    hmm::HMM_Classifier, 9

backward
    hmm, 5
baum_welch
    hmm, 5

classify
    hmm::HMM_Classifier, 9

forward
    hmm, 6

hmm, 5
    backward, 5
    baum_welch, 5
    forward, 6
    symbol_index, 6
    viterbi, 6
hmm::HMM, 7
    __init__, 8
hmm::HMM_Classifier, 9
    __init__, 9
    add_neg_hmm, 9
    add_pos_hmm, 9
    classify, 9

symbol_index
    hmm, 6

viterbi
    hmm, 6