[INF552 - Machine Learning for Data Informatics]

Technical Specification and Study Report

[Hidden Markov Model with Viterbi Algorithm]

Author: [Cheng-Lin Li (9799716705), Jianfa Lin (6950769029)]
Date: [Dec., 3, 2016]
Version: [1.0]

Document Control

Author

Position	Name	USC ID
Student	Cheng-Lin Li	9799716705
Student	Jianfa Lin	6950769029

Revision history

Version	Issue date	Author/editor	Description/Summary of changes
0.9	12/03/2016	Jianfa Lin	First draft release
1.0	12/03/2016	Jianfa Lin	First release

Related documents

Document	Description
HMM.py	The implementation of HMM by Python 3.5
hmm-data.txt	The data file.

Contribution

Name	USC ID	Labor
Cheng-Lin Li	9799716705	Implement the code.
Jianfa Lin	6950769029	Write the document.

Table of Contents

1 INTRODUCTION

1.1 Objectives

1. Implement a Hidden Markov Model to figure out the most likely trajectory of a robot in a 10-by-10 2D grid-world. For a true distance d, the robot records a noisy measurement chosen uniformly at random from the set of numbers in the interval [0.7d, 1.3d] with one decimal place.
2. Output should be the coordinates of the most likely trajectory of the robot for 11 time-steps.

2 Functionality [1]

2.1 Description: Hidden Markov Model

1. Input different sets of data as GRID_WORLD, TOWER_LOCATION and NOISY_DISTANCES from the file.
2. Generate states matrix, initial probability matrix and transition matrix according to GRID_WORLD data. Assign corresponding coordinates to valid cells in GRID_WORLD.
3. Generate emission matrix and observation matrix according to GRID_WORLD data and TOWER_LOCATION data.
4. Print the result of the above matrixes.
5. Instantiate a HMM variable “hmm”.

2.2 Description: Viterbi Algorithm

1. Get the probabilities of states from 4 given observation sequences of evidences.
2. Calculate the states probability from last states in every time step, select the maximum probability of each current state and record the last state that leads to this probability. Form a new initial matrix in each time-step.

3 Design

3.1 Data Structure

1. STATES: States list to store labels of each state.
   - ⇨ [state1, state2, ..., stateN].
2. TOWER_LOCATION: location list for towers
   - ⇨ [tower1[x1,y1], ..., towerN[xN,yN]]
3. NOISY_DISTANCES: The observation sequence of evidence
   - ⇨ [Time1[distance between robot and Tower1, ..., distance between robot and TowerN], ..., TimeN[distance between robot and Tower1, ..., distance between robot and TowerN]]
4. OBSERVATIONS = []: Observation list to store labels of evidence
   - ⇨ [observation1, observation2,..., observationN]
5. INITIAL_PROB: The matrix of probability from system start to first state.
   - ⇨ [[probability from start to state1, ..., probability from start to stateN]]
6. TRANSITION_MATRIX
   - ⇨ [Time T at state1[Probability to Time T+1 to state1,..., Probability to Time T+1 to stateN], ..., Time T at stateN[Probability to Time T+1 to state1,..., Probability to Time T+1 to stateN]]
7. EVIDENCE_MATRIX: Store observation of evidence. It support multiple evidences in the same time step.
   - ⇨ [[evidence1],[evidence2],...,[evidenceN]], [evidenceN]=[state1[Probability of evidence1, ..., Probability of evidenceN], ..., stateN[Probability of evidence1, ..., Probability of evidenceN]]
8. DISTANCE_MATRIX: Store distance(observation) probability of each cell for each tower.
   - ⇨ [[tower1],...,[tower4]], [tower1]=[cell1[Probability of distance0, ..., Probability of distanceN]]
9. VALIDCELL2COORDINATION: Store the map between state to coordination of the grid.
   - ⇨ {state1,[x,y], ...,stateN[x,y]}
10. DISTANCE_MATRIX: Store distance(observation) probability of each cell for each tower
    - ⇨ [[tower1],...,[tower4]]
    - ⇨ [tower1]=[cell1[Probability of distance0, ..., Probability of distanceN]] stateN[Probability of evidence1, ..., Probability of evidenceN]]
11. _state_prob: List of consolidate probability for every state at time t.
    - ⇨ [probability of state1, ..., probability of stateN]

3.2 Procedures

1. Get input data from file.
2. Format the GRID_WORLD, TOWER_LOCATION, NOISY_DISTANCES from data file.
3. From GRID_WORLD, generate matrix STATES, matrix INITIAL_PROB, and matrixes TRANSITION_MATRIX, VALIDCELL2COORDINATION through getCPT(GRID_WORLD).
4. Generate matrixes DISTANCE_MATRIX, OBSERVATIONS through getDistance(TOWER_LOCATION, GRID_WORLD).

5. Instantiate a HMM variable through HMM(STATES, OBSERVATIONS, INITIAL_PROB, TRANSITION_MATRIX, DISTANCE_MATRIX, 'viterbi').
6. Use the function viterbi (self, observation_seq) which implementing the Viterbi algorighm to calculate the maximum probability of state in the last time-step, then backtrack the state provides the condition for getting this probability, until the starting state is determined.
7. Print out the possible route of robots.

3.3 Optimizations and Challenges

3.3.1 Optimization

- Set up the necessary matrixes showing the state-and-state relation, state-and-observation relation, matrix recording coordinates of point, and other lists to make calculation easier and the result traceable.
- We use array data structure to save multiple loops in probability computing, making the code easy to maintain.
- We save the previous state list when we compute and select the maximum probability of current states. It save another loop to speed up the code.

3.3.2 Challenge:

- At this time, there are 87 states and 166 observations, which lead to a large data set and difficult to set up the matrix clearly.
- It’s confusing to implement the Viterbi Algorithm under the given conditions, especially when distinguish the matrixes needed for initial state and the following states.
- Calculation is quite complex because of too many relations.
- It’s not easy to debug and hard to check the correctness of every calculation results.

4 Usage Instruction

Since our implementation leverage Python 3.5 with only numpy package, as long as installing Python version 3.5 and numpy package, it's easy to run the program.

4.1 Script Files and Dataset File in same folder

- HMM.py (Implementation code)
- Hmm-data.txt (data file)

4.2 Execution from command line

1. Get into the folder of code.
2. Open command line under the current path.
3. Input the command: python HMM.py

4.3 Execution Results

This is the execution result for HMM we get from our implementation.

The route of robot is:

[[5, 3], [6, 3], [7, 3], [7, 2], [7, 1], [7, 2], [7, 1], [6, 1], [5, 1], [4, 1], [3, 1]]

```
NOISY_DISTANCES [[6.3, 5.9, 5.5, 6.7], [5.6, 7.2, 4.3, 6.8], [7.6, 9.4, 4.3, 5.4], [9.5,
10.0, 3.7, 6.6], [6.0, 10.7, 2.8, 5.8], [9.3, 10.2, 2.6, 5.4], [8.0, 13.1, 1.9, 9.4],
[6.4, 8.2, 3.9, 8.8], [5.0, 10.3, 3.6, 7.2], [3.8, 9.8, 4.4, 8.8], [3.3, 7.6, 4.3, 8.5]]
total cells=100, valid cells=87, initial probability=0.011494
STATES, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87]
VALIDCELL2COORDINATION, {1: [0, 0], 2: [0, 1], 3: [0, 2], 4: [0, 3], 5: [0, 4], 6: [0, 5],
7: [0, 6], 8: [0, 7], 9: [0, 8], 10: [0, 9], 11: [1, 0], 12: [1, 1], 13: [1, 2], 14: [1,
3], 15: [1, 4], 16: [1, 5], 17: [1, 6], 18: [1, 7], 19: [1, 8], 20: [1, 9], 21: [2, 0],
22: [2, 1], 23: [2, 7], 24: [2, 8], 25: [2, 9], 26: [3, 0], 27: [3, 1], 28: [3, 3], 29:
[3, 4], 30: [3, 5], 31: [3, 7], 32: [3, 8], 33: [3, 9], 34: [4, 0], 35: [4, 1], 36: [4,
3], 37: [4, 4], 38: [4, 5], 39: [4, 7], 40: [4, 8], 41: [4, 9], 42: [5, 0], 43: [5, 1],
44: [5, 3], 45: [5, 4], 46: [5, 5], 47: [5, 7], 48: [5, 8], 49: [5, 9], 50: [6, 0], 51:
[6, 1], 52: [6, 3], 53: [6, 4], 54: [6, 5], 55: [6, 7], 56: [6, 8], 57: [6, 9], 58: [7,
0], 59: [7, 1], 60: [7, 2], 61: [7, 3], 62: [7, 4], 63: [7, 5], 64: [7, 6], 65: [7, 7],
66: [7, 8], 67: [7, 9], 68: [8, 0], 69: [8, 1], 70: [8, 2], 71: [8, 3], 72: [8, 4], 73:
[8, 5], 74: [8, 6], 75: [8, 7], 76: [8, 8], 77: [8, 9], 78: [9, 0], 79: [9, 1], 80: [9,
2], 81: [9, 3], 82: [9, 4], 83: [9, 5], 84: [9, 6], 85: [9, 7], 86: [9, 8], 87: [9, 9]}
OBSERVATIONS= [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4,
1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2,
3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0,
5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8,
6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.0, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6,
8.7, 8.8, 8.9, 9.0, 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9, 10.0, 10.1, 10.2, 10.3,
10.4, 10.5, 10.6, 10.7, 10.8, 10.9, 11.0, 11.1, 11.2, 11.3, 11.4, 11.5, 11.6, 11.7, 11.8,
11.9, 12.0, 12.1, 12.2, 12.3, 12.4, 12.5, 12.6, 12.7, 12.8, 12.9, 13.0, 13.1, 13.2, 13.3,
13.4, 13.5, 13.6, 13.7, 13.8, 13.9, 14.0, 14.1, 14.2, 14.3, 14.4, 14.5, 14.6, 14.7, 14.8,
14.9, 15.0, 15.1, 15.2, 15.3, 15.4, 15.5, 15.6, 15.7, 15.8, 15.9, 16.0, 16.1, 16.2, 16.3,
16.4, 16.5]
State sequence= [44, 52, 61, 60, 59, 60, 59, 51, 43, 35, 27]
Coordination sequence= [[5, 3], [6, 3], [7, 3], [7, 2], [7, 1], [7, 2], [7, 1], [6, 1],
[5, 1], [4, 1], [3, 1]]
```

5 Software Familiarization

5.1 Overview

We find a very easy-to-use package from Python Package Index[6], which is hidden_markov package. The current version is 0.3.1. This package Implements the Hidden Markov model in discrete domain.

5.2 HMM in hidden_markov library

5.2.1 API introduction[7]

In the this implementation, it provides a HMM class as implementation: class hidden_markov.hmm(states, observations, start_prob, trans_prob, em_prob). There are several parameters can be assigned into this implementation.

1. states (A list or tuple) – The set of hidden states
2. observations (A list or tuple) – The set unique of possible observations
3. start_prob (Numpy matrix, dimension = [length(states) X 1]) – The start probabilities of various states, given in same order as ‘states’ variable. start_prob[i] = probability(start at states[i]).
4. trans_prob (Numpy matrix, dimension = [len(states) X len(states)]) – The transition probabilities, with ordering same as ‘states’ variable . trans_prob[i,j] = probability(states[i] -> states[j]).
5. em_prob (Numpy matrix, dimension = [len(states) X len(observations)]) – The emission probabilities, with ordering same as ‘states’ variable and ‘observations’ variable. em_prob[i,j] = probability(states[i],observations[j]).

5.3 Viterbi Algorithm

In the function Viterbi(self, observation), there are several arguments.

1. param observations: The observation sequence, where each element belongs to 'observations' variable declared with __init__ object.
2. type observations: A list or tuple
3. return: Returns a list of hidden states.
4. rtype: list of states

The function initializes data at first and stores the state sequence giving maximum probability. The following initialization is similar to ours.

During the procedure that finding delta[t][x] for each state 'x' at the iteration 't', it map observation to an index, then update delta and scale it. After this step, it can find state which is most probable using argax. Finally, update the path. With all these effort, it’s easy to find the state in last stage, giving maximum probability

5.4 How to improve our implementation

5.4.1 Improvement opportunities on Viterbi Algorithms

1. Use mapping to make acquire a certain observation easier.
2. According to our latest update referencing the implementation of Viterbi Algorithm from Wikipedia[5], we calculate the initial matrix and transition matrix at first, then select the maximum probability of each state and form a new 1*87 matrix. Multiply this matrix by

emission matrix in the current time-step. This modification improve the reliability of result.

6 Application

6.1 The applications on Speech Synthesis

The HMM not only do well in speech recognition but also be good at speech synthesis.

Hidden Markov model (HMM)-based speech synthesis has recently been demonstrated to be very effective in synthesizing speech. It can change speaker identities, emotions, and speaking styles.[8] Another idea is that an input utterance is recognized utilizing the hidden Markov model (HMM) for speech recognition, and the recognized phoneme sequences are used as labels for speech synthesis. The aim of this work is to extend functionality from many-to-one to many-to-many VC.[9]

7 Reference

[1] Alpaydin E. Introduction to machine learning[M]. MIT press, 2014.

[2] Danial Ramage, Stanford CS229 Section Note, Dec. 1, 2017, URL http://cs229.stanford.edu/section/cs229-hmm.pdf

[3] A Python implementation of the Hidden Markov Model. URL http://jason2506.github.io/PythonHMM/

[4] Hidden Markov Models in Python, with scikit-learn like API http://hmmlearn.readthedocs.org. URL https://github.com/hmmlearn/hmmlearn

[5] Viterbi algorithm on Wikipedia. URL https://en.wikipedia.org/wiki/Viterbi_algorithm

[6] hidden_marcov 0.3.1. URL https://pypi.python.org/pypi/hidden_markov/0.3.1

[7] Document of hidden_markov 0.3.1. URL http://hidden-markov.readthedocs.io/en/latest/

[8] Tokuda, Keiichi, et al. "Speech synthesis based on hidden Markov models." Proceedings of the IEEE 101.5 (2013): 1234-1252.

[9] Aizawa, Yoshitaka, Masaharu Kato, and Tetsuo Kosaka. "Many-to-many voice conversion using hidden Markov model-based speech recognition and synthesis." The Journal of the Acoustical Society of America 140.4 (2016): 2964-2965.