HMMdotEM User Guide

Nikola Karamanov

HMMdotEM is a fast, feature-full, extendable package for training general discrete-state hidden Markov models using the EM algorithm. All you need to have is a function that computes the conditional log-probability for a set of data points given each discrete state's parameters. Any other information that you have, such as an analytic solution in the M-Step can be easily added to the existing code. This guide provides help for using these features. Currently the code has only a Matlab® interface.

License

Contents

1 Installation

1.1 Quick Start

This section assumes you have extracted a folder called "HMMdotEM" from the "HMMdotEM.zip" file you downloaded.

1.1.1 Using startup.m

1. Inside Matlab® , switch to your HMMdotEM directory and run

   ```
   >> startup
   ```

   Now you can switch to any other directory (and the code will still be accessible).
   Note: If you follow this installation procedure you must follow this step every time.

2. To setup the package, mex files, other optimizations and optional components run

   ```
   >> HMMdotEM.configure();
   ```

   Note: You only need to configure the project once for each architecture you are using.

1.1.2 Configuring from another directory.

1. Open the file HMMdotEM.m in the "config" folder and change the line:
   $PATH$ = HMMdotEM.deducehmmdotempath;
   (If you are wondering what you should change it to, read the instructions above that line).

2. Now add the "config" folder to your Matlab® path. (In the lower left corner click "Start", then "Desktop Tools", then "Path", then click on the "Add folder" button and find the folder)

3. Run the following (This might compile some files):

   ```
   >> HMMdotEM.install();
   ```

4. Now any time you startup Matlab® , from any directory, you can run

   ```
   >> HMMdotEM.startup();
   ```

   to use the package.

5. If you have any problems, try redoing these steps.

1.2 Expert: How to personalize the code to your workflow

There is a file called "HMMdotEM.m" in the "config" folder. It defines an object class and contains code for setting up paths and providing information about the package. There are also methods there that allow you to recompile files and administer the package in general.

2 Features worth acknowledging

2.1 Any Conditional Likelihood

You have the ability to reuse the inference in the E-step for any conditional likelihood that you may think of. There is even a generic and customizable M-step solver that you can use if no analytic solution exists for your M-step.

2.2 Multiple Segments

There is the ability to train simultaneously on multiple segments/chunks of data. This is done by using a herein coined notion of "Param-Group" of HMM's. Param-Groups can be constructed from an instance of a singular HMM and then be passed to the relevant EM functions without any difference in code.

2.3 Efficiency

Speed and memory have been optimized for. The code is very conservative in memory, it clears unneeded quantities the moment they are not needed anymore. C mex files are used in the E-step for speed. If you are having problems with the compilation or otherwise you can avoid using them and fall back on slower but trustworthy Matlab® code.

2.4 Help and Documentation

This documentation is contained in the "doc" folder. Various other help can be found in the source's comments.

3 Developing your model

3.1 Object-Orientation

Recent versions of Matlab® have support for object-orientation.[1] This package uses object-orientation to facilitate code-reuse and abstraction. You are not require you to understand the details of object orientation, but a few things will help you get started faster. Here they are.

3.1.1 Matlab Classes

Matlab® classes get defined in ".m" files. Here is an example:

```
% Example.m
classdef Example
  properties % These can be made private/public/static etc.
    property1     % Default value will be the empty array
    property2 = 3 % Default values can be set
  end
  methods % These can be made private/public/static etc.
    function obj = Example(input1,input2,varargin)
      % The constructor
      if nargin>0 % For inheritance must allow constructing with no inputs
       obj.property1 = input1*input2;
       if nargin>2
         obj.property2 = varargin{1};
       end
      end
    end
    function obj = method1(obj,input1)
      obj.property2 = input1;
    end
  end
  methods(Static)
    function main
      fprintf('Constructing Example with no inputs\n');
      E = Example
      fprintf('Constructing Example with inputs 10,2,13\n');
      E = Example(10,2,13)
    end
  end
end

% To run this example, put this "classdef" a file named Example.m and run
% >> Example.main
```

For more information see the Matlab® documentation of object orientation.

[1] This has existed before, but was based on a non-elegant use of struct, while the most recent versions handle classes and inheritance in a clean way.

3.1.2 Folders

Optionally ".m" files containing classes can be placed in separate folders with corresponding names. For "Example.m" above this folder would be called "@Example", and it can contain other files (see the Matlab® documentation for details).

3.1.3 Inheritance

Inheritance lets you use alter/add methods/properties of one class to your liking, while still using all of the other non-altered methods/properties. Here is an example:

```
% AnotherExample.m
classdef AnotherExample < Example % AnotherExample inherits from Example
  properties
    property3 = 5
  end
  methods
    function obj = AnotherExample(input1)
      obj.property3 = input1;
    end
    function obj = method2(obj,input1)
      % Do something
      obj.property1 = input1;
    end

  end
  methods(Static)
    function main
      Example.main
      fprintf('Constructing AnotherExample\n')
      AE = AnotherExample(.001)
      fprintf('Using method1\n')
      AE = AE.method1(1.1)
      fprintf('Using method2\n')
      AE = AE.method2(2.2)
    end
  end
end

% Put this "classdef" a file called AnotherExample.m
% Make sure to have Example.m in the same directory.
% Then run
% >> Example.main
```

3.2 The Matlab Class for your model

3.2.1 Gaussian Mixture Conditional Likelihood

This package has built-in Gaussian-Mixture HMM in the "src" folder. See the file "TESTselection.m" in the "test" folder for examples on how you use it.

3.2.2 Other HMM's

This package is designed so you don't have to worry about the details of EM or its implementation. However you will probably want to define something other than the Gaussian Mixture HMM. To do

this you will need to implement (read to the end of this section to understand everything before you begin):

- (Required Constructor) A constructor for a Matlab® class instance representing your model.
- (Required Conditional Likelihood) A conditional likelihood function of your data given the hidden states.
- (Optional M-step solver) The function that solves the M step of the EM algorithm for your type of conditional likelihood. This is optional because there is a built-in general optimizer for the M-step, but that should not be expected to give you good results, unless you understand how to tweak it.

Refer to "@GaussMixHMM" and "HMMTemplate.m" in the "src" folder for example code. Suppose you want to define a new model with a Exponential(λ_h) conditional distribution, where h is the index of the hidden state. Then you would copy the contents "HMMTemplate.m" to "Exponential.m" and alter them in accordance with the instructions in the comments and relying on the example in "GaussMixHMM.m" for guidance. A very basic understand of HMM's will be required.

3.3 Details of Implementation

The methods and properties, as well as the logic of the package are described in corresponding files in the "src" folder.

3.4 WARNINGS TO DEVELOPER

3.4.1 Breaking the code

IT IS STRONGLY ADVISED THAT YOU **D**O NOT ALTER ANY FILES IN THE ORIGINAL PACKAGE. Doing otherwise may break some of the logic that makes the code run efficiently or correctly. You should not have to do anything other than define methods explicitly presented in "HMMTemplate.m" whilst following the instructions therein very carefully.[2]

[2]This warning is mostly a scare tactic, you can obviously find things in the main files that are safe to alter. The maintained view is that you should never need to do this.

4 Usage

4.1 Workflow

It is advised to put all new model files in the "src" folder and any scripts in the "test" folder. A typical workflow is the following[1]:

```
(Start Matlab and move to HMMdotEM directory)
>> cd test % Best place to run anything
>> TESTselection(); % Run something (in this case the built-in Gaussian test)
>> restart; % useful, sometimes Matlab has stale object instances hanging around
(Now in "test" folder again, ready to run next script)
>> somescriptforyourmodel; % Some script you wrote for your model
```

[1] Make sure you have installed the package (see chapter 1)