Simple-HOHMM Documentation

Release 0.0.3

Jacob Krantz

Mar 18, 2018

Topics

Simple-HOHMM is an end-to-end sequence classifier using Hidden Markov Models. Let the builder construct a model for you based on chosen model attributes. Now you can solve the classic problems of HMMs: evaluating, decoding, and learning. Play with different orders of history to maximize the accuracy of your model.

This documentation is under development, but the tutorials are a good place to start.

CHAPTER 1

Getting Started

1.1 Installation for Python 2 or 3

`Simple-HOHMM` can be installed directly from Github using `pip`. You must have `git` installed for this process to work.

```
>>> pip install git+https://github.com/jacobkrantz/Simple-HOHMM.git
```

If you want the most recent staging build:

```
>>> pip install git+https://github.com/jacobkrantz/Simple-HOHMM.git@staging
```

Alternative: to view the source code and run the tests before installation:

```
>>> git clone https://github.com/jacobkrantz/Simple-HOHMM.git
>>> cd Simple-HOHMM
>>> python setup.py test
>>> python setup.py install
```

1.2 Installation for Pypy

For usage with `pypy`, you must install with `pip` inside `pypy`:

```
>>> pypy -m pip install git+https://github.com/jacobkrantz/Simple-HOHMM.git
```

If this fails, try installing `pip` for `pypy` first:

```
>>> curl -O https://bootstrap.pypa.io/get-pip.py
>>> pypy get-pip.py
```

If you want the most recent staging build still with `pypy`:

```
>>> pypy -m pip install git+https://github.com/jacobkrantz/Simple-HOHMM.git@staging
```

Alternative staging branch with `pypy`:

```
>>> sudo pypy -m pip install --upgrade https://github.com/jacobkrantz/Simple-HOHMM/
↪archive/staging.zip
```

CHAPTER 2

Tutorials

The following tutorials are meant to give you a jump start in applying the tools of Simple-HOHMM. To see what model attributes are adjustable, view the API Reference.

2.1 Supervised

The following example is adapted from Wikipedia.

Suppose villagers are either healthy or have a fever. Fevers are diagnosed by the doctor asking patients how they feel (normal, dizzy, or cold). Assuming their health can be modeled by a discrete Markov chain, the observations are `(normal, dizzy, cold)` and the hidden states are `(healthy, fever)`. The doctor has seen patients in the past, and kept that data. The observations are in one list and the states are in another such that `states[i]` corresponds to `observations[i]`:

```
observations = [
        ['normal', 'cold', 'dizzy', 'dizzy','normal','normal'],
        ['cold', 'cold', 'dizzy', 'normal','normal','normal'],
        ['dizzy', 'dizzy', 'cold', 'normal', 'dizzy', 'normal'],
        ['normal', 'normal', 'cold', 'dizzy', 'dizzy', 'dizzy']
]
states = [
        ['healthy', 'healthy', 'fever', 'fever', 'healthy', 'healthy'],
        ['healthy', 'fever', 'fever', 'healthy', 'healthy', 'fever'],
        ['fever', 'fever', 'fever', 'healthy', 'healthy', 'healthy'],
        ['healthy', 'healthy', 'healthy', 'fever', 'fever', 'fever']
]
```

We can now build a first order Hidden Markov Model based on the observations and states above:

```
from SimpleHOHMM import HiddenMarkovModelBuilder as Builder
builder = Builder()
builder.add_batch_training_examples(observations, states)
hmm = builder.build()
```

Now suppose a patient has been seeing the doctor for three days and felt `(normal, cold, dizzy)`. What might the doctor guess about this patient's health? This is solved with Viterbi decoding:

```
obs =  ['normal', 'cold', 'dizzy']
states = hmm.decode(obs)
print(states) # prints: ['healthy', 'healthy', 'fever']
```

We can also determine the likelihood of a patient feeling `(normal, cold, dizzy)`:

```
obs = ['normal', 'cold', 'dizzy']
likelihood = hmm.evaluate(obs)
print(likelihood) # prints: 0.0433770021525
```

2.2 Semi-Supervised

For this example, we will use the same `observations` and `states` as the Supervised example. Here we initialize our model just as before:

```
from SimpleHOHMM import HiddenMarkovModelBuilder as Builder
builder = Builder()
builder.add_batch_training_examples(observations, states)
hmm = builder.build()
```

From here we can improve the model's training even further by exposing it to observations it has not seen before. Since we are using a small set, we will limit the learning process to one iteration instead of delta convergence by utilizing the `iterations=1` parameter. Also, we use `k_smoothing=0.05` to avoid cases of zero probability:

```
sequences = [
                ['normal', 'cold', 'dizzy','normal','normal'],
                ['normal', 'cold', 'normal','dizzy','normal'],
                ['dizzy', 'dizzy', 'dizzy','cold','normal'],
                ['dizzy', 'dizzy', 'normal','normal','normal'],
                ['cold', 'cold', 'dizzy','normal','normal'],
                ['normal', 'dizzy', 'dizzy','normal','cold'],
                ['normal', 'cold', 'dizzy', 'cold'],
                ['normal', 'cold', 'dizzy']
]
hmm.learn(sequences, k_smoothing=0.05, iterations=1)
```

We now determine the updated likelihood and hidden state sequence. Notice that running hmm.learn() has increased the likelihood of our observation:

```
obs = ['normal', 'cold', 'dizzy']
print(hmm.evaluate(obs)) # prints 0.052111435936
print(hmm.decode(obs)) # prints ['healthy', 'fever', 'fever']
```

2.3 Unsupervised

In fully unsupervised scenarios, we build and train a model with no prior training examples to draw from. The only data we supply to our model is the set of possible observations, the set of possible hidden states, and a collection of observation sequences to optimize for.

We first gather the data to supply to our model:

```
possible_observations = ['normal', 'healthy', 'dizzy']
possible_states = ['healthy', 'fever']
sequences = [
        ['normal', 'cold', 'dizzy','normal','normal'],
        ['normal', 'cold', 'normal','dizzy','normal'],
        ['dizzy', 'dizzy', 'dizzy','cold','normal'],
        ['dizzy', 'dizzy', 'normal','normal','normal'],
        ['cold', 'cold', 'dizzy','normal','normal'],
        ['normal', 'dizzy', 'dizzy','normal','cold'], #start new here
        ['normal', 'cold', 'dizzy', 'dizzy','normal','normal'],
        ['dizzy', 'cold', 'dizzy', 'normal','normal','normal'],
        ['dizzy', 'cold', 'dizzy', 'normal','normal','normal'],
        ['normal', 'cold', 'dizzy', 'dizzy','cold','normal'],
        ['dizzy', 'dizzy', 'dizzy', 'dizzy', 'cold', 'cold'],
        ['cold', 'cold', 'cold', 'normal', 'dizzy', 'normal'],
        ['dizzy', 'normal', 'cold', 'cold', 'dizzy', 'dizzy']
]
```

There are two initial distributions to choose from, either `uniform` or `random`. This selection applies to model parameters A, B, pi. In our case we will initialize with a random distribution:

```
from SimpleHOHMM import HiddenMarkovModelBuilder as Builder
builder = Builder()
hmm = builder.build_unsupervised(
        single_states=possible_states,
        all_obs=possible_observations,
        distribution="random",
        highest_order=2
)
```

We can view the initial model parameters, train our model using Baum-Welch EM, then again view our parameters to see how they have been modified:

```
hmm.display_parameters()
hmm.learn(sequences, k_smoothing=0.001)
hmm.display_parameters()
```

Results may be inconsistent due to the random initial distributions. You can play with different k_smoothing values, delta values, and sequence selection. Of course, train on prior examples where possible.

CHAPTER 3

API Reference

TODO: detailed reference guide to using the API

CHAPTER 4

Implementation References

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," in Proceedings of the IEEE, vol. 77, no. 2, pp. 257-286, Feb 1989. doi: 10.1109/5.18626 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=18626&isnumber=698

[2] Daniel Jurafsky & James H. Martin. (2016). *Speech and Language Processing*. Draft of August 7, 2017. URL: https://web.stanford.edu/~jurafsky/slp3/

[3] Du Preez, J.A., *Efficient high-order hidden Markov modelling.* PhD Dissertation, University of Stellenbosch, South Africa, 1998. URL: http://www.ussigbase.org/downloads/jadp_phd.pdf

4.1 Web articles

- https://en.wikipedia.org/wiki/Forward_algorithm
- https://en.wikipedia.org/wiki/Forward%E2%80%93backward_algorithm
- https://en.wikipedia.org/wiki/Viterbi_algorithm
- https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm

CHAPTER 5

License