Machine Learning Fundamentals

Hidden Markov Models, Practical Session Bonus work

Master DSC/MLDM/3DMT

Note: all the material is available on the claroline connect page! This work is not mandatory for the evaluation of the machine course but it can bring a bonus for the practical mark.

1 Introduction and Installation

For this practical session we will use the `learnhmm` library available here

https://github.com/hmmlearn/hmmlearn

`hmmlearn` is a set of algorithms for unsupervised learning and inference of Hidden Markov Models[1].
This library requires the following dependencies

- Python $\geq$ 2.6
- NumPy (tested to work with $\geq$1.9.3)
- SciPy (tested to work with $\geq$0.16.0)
- scikit-learn $\geq$ 0.16
- You also need Matplotlib $\geq$ 1.1.1 to run the examples and pytest $\geq$ 2.6.0 to run the tests.

If you installed anaconda and scikit-learn, everything should be ok. To install the library it is then sufficient to type

```
pip install -U --user hmmlearn
```

The documentation is available here http://hmmlearn.readthedocs.io/en/stable/
The library allows one to deal 3 types of HMM:

- `GaussianHMM`: HMM with Gaussian emissions (*i.e.* real numbers generated from continuous Gaussian distributions) that can be used to deal with univariate times series for example.
- `GMMHMM`: HMM with Gaussian Mixture emissions.
- `MultinomialHMM`: HMM with multinomial (discrete) emissions - this the type of HMM we saw in class.

A first example:

```
import numpy as np
from hmmlearn import hmm

states = ["Rainy", "Sunny"]
n_states = len(states)
```

[1]For supervised learning learning of HMMs and similar models see seqlearn: https://github.com/larsmans/seqlearn

```
observations = ["walk", "shop", "clean"]
n_observations = len(observations)

model = hmm.MultinomialHMM(n_components=n_states, init_params="",
                           n_iter=10,algorithm='map',tol=0.00001)
model.startprob_ = np.array([0.6, 0.4])
model.transmat_ = np.array([
  [0.7, 0.3],
  [0.4, 0.6]
])
model.emissionprob_ = np.array([
  [0.1, 0.4, 0.5],
  [0.6, 0.3, 0.1]
])

gen1 = model.sample(3)
print(gen1)
seqgen2, stat2= model.sample(5)
print(seqgen2)
print(stat2)
gen3=model.sample(2)
print(gen3)

sequence1 = np.array([[2, 1, 0, 1]]).T
logproba=model.score(sequence1)
print(logproba)

logproba_extend=model.score_samples(sequence1)
print(logproba_extend)

p = model.predict(sequence1)
print(p)
p_extend = model.score_samples(sequence1)
print(p_extend)

model.fit(sequence1)

p_extend = model.score_samples(sequence1)
print(p_extend)

model.startprob_ = np.array([0.5, 0.5])
model.transmat_ = np.array([
  [0.7, 0.3],
  [0.3, 0.7]])
model.emissionprob_ = np.array([
  [0.2, 0.3, 0.5],
  [0.7, 0.2, 0.1]
])

model.fit(seqgen2)

p_extend = model.score_samples(sequence1)
print(p_extend)
```

A note for dealing with multiple sequences - you must concatenate them in a tab of tab and provide the corresponding sizes in anoter tab:

```
sequence3 = np.array([[2, 1, 0, 1]]).T
sequence4 = np.array([[2, 1, 0, 1, 1]]).T
sample = np.concatenate([sequence3, sequence4])
lengths = [len(sequence3), len(sequence4)]
model.fit(sample,lengths)
```

In the example above, the name of the states are not used in the HMM, you can rather use their corresponding index. This is what is done to generate `sequence 1`: 2 stands for `clean`, 1 for `shop` and 0 for `walk` ; similarly the states output by Viterbi algorithm refers to the indexes of the states. To have more semantics in your display, you can use the following trick for the name of the observations/states:

```
sequence = np.array([[2, 1, 0, 1]]).T
logprob, state_seq = model.decode(sequence, algorithm="viterbi")
print("Observations", ", ".join(map(lambda x: observations[x], sequence)))
print("Associated states:", ", ".join(map(lambda x: states[x], state_seq)))
```

Note: in the above code some warnings may appear.

2 Work to do

You must write a report giving your answers to the following questions.

2.1 Analyse the previous code

In the example above, several functions for HMM have been used: forward, Viterbi, BaumWelch - note that the probabilities are output in log form. Try this code and reports the objective of each function - you can use the documentation to help you.

To help you, you can test the HMM seen in class - the 3-states HMM about weather forecast and Museum/Beach activities - and check that you are able to find back the results of the exercises done in class.

To facilitate your tests, it could be good to create python script(s).

2.2 HMM Learning

Initialize now an HMM with following parameters:

$$A = \begin{pmatrix} 0.45 & 0.35 & 0.20 \\ 0.10 & 0.50 & 0.40 \\ 0.15 & 0.25 & 0.60 \end{pmatrix}, \; B = \begin{pmatrix} 1.0 & 0.0 \\ 0.5 & 0.5 \\ 0.0 & 1.0 \end{pmatrix}, \; \pi = \begin{pmatrix} 0.5 \\ 0.3 \\ 0.2 \end{pmatrix}$$

1. Compute the probability of the string *abbaa*
2. Apply BaumWelch with only one iteration and check the probability of the string
3. Do the same thing after 15 iterations
4. Try to obtain the result at convergence (probably around 150 iterations, but the precision parameter should be very small). What are the new parameters of the HMM and the new probability of the string. Does the result seem intuitively correct.
5. Now create an HMM with 5 states with parameters initialized at any non zero correct values. Learn using only the string *abbaa*, what is the final result?

2.3 HMM Learning - exercise 2

Consider the following HMM initialized randomly

$$A = \begin{pmatrix} 0.40 & 0.60 \\ 0.52 & 0.48 \end{pmatrix},\ B = \begin{pmatrix} 0.49 & 0.51 \\ 0.40 & 0.60 \end{pmatrix},\ \pi = \begin{pmatrix} 0.31 \\ 0.69 \end{pmatrix}$$

1. Create this HMM with the library.
2. Learn the HMM with the following sample $L_1 = \{aaabb, abaabbb, aaababb, aabab, ab\}$. Give the model obtained
3. Now learn the HMM created in 1) with this new sample $L_2 = \{bbbaa, babbaa, bbbabaa, bbabba, bbaa\}$
4. Compute the probabilities of the strings *aababbb* and *bbabaaa*. Are the results intuitive?

2.4 Handwritten digit recognition

On claroline connect, you can download a database of handwritten digit represented with Freeman codes - `digit_strings.zip`. The database contains around 1000 numbers per class (from 0 to 9), and each number is represented by a sequence of Freeman directions. The Freeman encoding consists in representing the contour of a shape according to 8 primitive directions (i.e. an alphabet of 8 symbols). The algorithms starts from the upper left pixel and follows the shape until coming back to the starting point. The sequence of directions used during the traversal defines the string representation of the considered number. Check Figure 1 for an illustration.

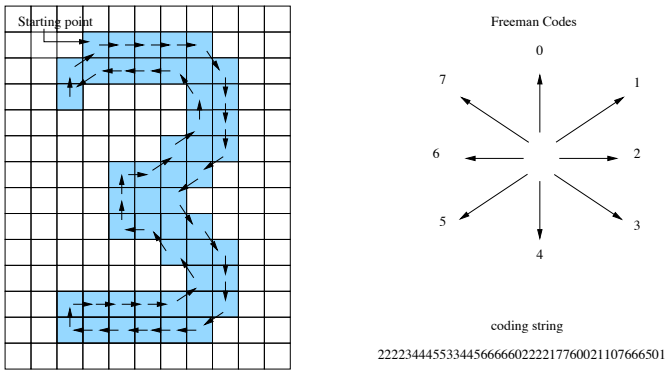

Figure 1: Representation of the digit 3 with the Freeman codes.

The objective is to define a classification model for classifying new digits. The idea is to learn on HMM per class, and to classify any new digit to the class associated to the HMM giving the highest probability to the corresponding sequence.

For this purpose you must define an experimental setup (train, test, ...) and consider different HMM sizes and give the best accuracy obtained. To start you can begin with small samples and increase them when you have found a good setting. Do not hesitate to create python scripts to automate your procedures.

Note that in the digit database, the strings are just sequences of integers. To use them with the library, you can process the file to separate each caracter by a comma (,) and then you can use the python function `fromstring` from the `numpy` library.

2.5 Personal modeling

Propose a problem that you can be tackled by learning HMM models. You do not have to provide any code or any experiments but you must precise in depth how you model the problem with HMM.