Machine Learning - Lab 2

Mohammad Poul Doust

April 2019

1 Introduction

2 Analyze the given code

we first define the states and the observation for our Hidden Markov Model. we will use the states and observations seen in the lectures

```
states = ["Rainy", "Cloudy", "Sunny"]
observations = ["Museum", "Beach"]
```

After that, we initialize the model (Discrete emission):

```
model = hmm.MultinomialHMM(n_components=n_states, init_params="",
n_iter=10,algorithm='map',tol=0.00001)
```

where:

- n_components: is the number of states
- n_iter: is an upper bound for EM number of iteration
- algorithm: is the decoder algorithm (either "map" for maximum a posterior estimation or "Viterabi" algorithm).
- tol: threshold that EM algorithm stops when the change in gain is smaller than this threshold

Moreover, the model initial configuration are specfified:

```
odel.startprob_ = np.array([1,0,0])
model.transmat_ = np.array([
[0.2, 0.6, 0.3],
[0.3, 0.2, 0.5],
[0.1, 0.1, 0.8]
])
model.emissionprob_ = np.array([
[0.7, 0.3],
[1, 0],
[0.1, 0.9]
])
```

where startprop, transmat and emissionprob is starting probability, transition matrix and probability of emitting a symbol giving a state matrix, respectively. The following code used to generate a sequence of observations given our model.

```
model.sample(2) #this will generate a sequence of length 2
seqgen2, stat2= model.sample(5) # this will generate a sequence of length 5 with its cor
```

Now we want to calculate the probability of going to museum and to the beach after:

```
sequence1 = np.array([[0,1]]).T # Museum :0, Beach: 1
logproba=model.score(sequence1)
print(logproba)    #-1.783791299578878
print(np.exp(logproba)) # 0.168
```

we can see that the probability is 0.168 which is exactly the same answer we got in the lecture.

```
logproba_extend=model.score_samples(sequence1)
print(logproba_extend)
#(-1.783791299578878,
#array([[1.  , 0.  , 0.  ], [0.25, 0.  , 0.75]]))
```

The above code gives the log probability under the model and posteriors. Now we try to predict the most likely states generating the "sequence1"

```
p = model.predict(sequence1)
print(p)     #[0 2]
```

for estimating model parameters (training) using EM algorithm:

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

Finally, we print an observation and the most likely state generating it using "Viterabi" algorithm.

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

3 HMM Learning

3.1 Compute the probability of the string abbaa?

log probability of the string abbaa: -3.5840690897615914
probability of the string abbaa: 0.027762499999999996
Code:

```
model = hmm.MultinomialHMM(n_components=n_states, init_params="",
n_iter=10,algorithm='map',tol=0.00001)
model.startprob_ = np.array([0.5, 0.3, 0.2])
model.transmat_ = np.array([
[0.45, 0.35, 0.20],
[0.10, 0.50, 0.40],
[0.15, 0.25, 0.60]
])
model.emissionprob_ = np.array([
[1, 0],
[0.5, 0.5],
[0, 1]])

sequence1 = np.array([[0,1,1,0,0]]).T
logproba=model.score(sequence1)
print("log probability of the string abbaa: ", logproba)
print("probability of the string abbaa: ", np.exp(logproba))
```

3.2 Apply Baum Welch with only one iteration and check the probability of the string

4 HMM Learning - exercise 2

log One Iteration BaumWelch: -2.938884057874165
One Iteration BaumWelch: 0.05292475676186572

Code:

```
model.fit(sequence1)
p_extend = model.score(sequence1)
print("log One Iteration BaumWelch: ", p_extend)
print("One Iteration BaumWelch: ", np.exp(p_extend))
```

4.1 Do the same thing after 15 iterations

log 15 Iterations BaumWelch: -1.394402415961016
15 Iterations BaumWelch: 0.24798118169432756

Code:

```
model = hmm.MultinomialHMM(n_components=n_states, init_params="",
n_iter=15,algorithm='map',tol=0.00001)
model.fit(sequence1)
p_extend = model.score(sequence1)
print("log 15 Iterations BaumWelch: ", (p_extend))
print("15 Iterations BaumWelch: ", np.exp(p_extend))
```

4.2 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.

log At convergence BaumWelch: -1.386294445203866
At convergence BaumWelch: 0.24999997897900705

Code:

```
model = hmm.MultinomialHMM(n_components=n_states, init_params="",
   n_iter=130,algorithm='map',tol=0.00000001)
```

Yes, the results seem intuitive, but it is not a big progress, relatively.

4.3 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?

log At 5 states BaumWelch: -5.551115123125783e-17
At 5 states BaumWelch: 1.0

Code:

```
model = hmm.MultinomialHMM(n_components=5, init_params="",
n_iter=120,algorithm='map',tol=0.00000001)
```

The probability is 1, which may be explaines as the model fit truly all observation. However, we only have 1 observation of (sequence of 5 characters) so probably the model over-fitted the samples.

5 Handwritten digit recognition

We first start by splitting the data into about 77% for training and the rest for testing. We have three main classes:

- Digit.py : wrapper class that handles initialize hmm model, save, load, train and test
- data_loader.py: helper class to load dataset from test files and convert it into numpy array and split it with 2/3 for training and 1/3 for test.
- main.py: the main class the load datasets for each digit (using data_loader) and train the models, save them for future use. the evaluation is alaso carried out in this class, for each test case, it is passed to all 10 hidden markov models and the classification results is for the model with highest probability.

after trying multiple models for each digit, we achieved results like the following:

```
Correct:  3074  out of:  3519
Total accuracy:  87.35436203466894 %
```

6 Personal Modelling

As a personal modelling, a Hidden Markov Model could be built to predict the direction of penalty kick in football for a given player. for the observation, we could have these options:

- In the middle, looking at :(the middle, upper right, upper left, lower right, lower left)
- Upper Right, looking at :(the middle, upper right, upper left, lower right, lower left)
- Upper Left, looking at :(the middle, upper right, upper left, lower right, lower left)
- Lower Right, looking at :(the middle, upper right, upper left, lower right, lower left)
- Lower Left, looking at :(the middle, upper right, upper left, lower right, lower left)

So basically, we have 25 option. The model based on the idea that football player have tendency to repeat penalty patterns with correlation to eye direction. we could train 5 models, one for each direction (up right, up left, down right, down left, middle). each model take a sequence of observation as described above, for example: (UpRight, LookingUpRight)(UpRight, LookingUpRight)(UpLeft, LookingUpRight)(UpLeft, LookingUpLeft)(LowerRight, LookingLowerRigh)....