sheet07

June 20, 2019

1 Programming Hidden Markov Models (60 P)

In this exercise, you will experiment with hidden Markov models, in particular, applying them to modeling character sequences, and analyzing the learned solution. As a starting point, you are provided in the file `hmm.py` with a basic implementation of an HMM and of the Baum-Welch training algorithm. The names of variables used in the code and the references to equations are taken from the HMM paper by Rabiner et al. downloable from ISIS. In addition to the variables described in this paper, we use two additional variables: Z for the emission probabilities of observations O, and ψ (i.e. psi) for collecting the statistics of Equation (40c).

1.1 Question 1: Analysis of a small HMM (30 P)

We first look at a toy example of an HMM trained on a binary sequence. The training procedure below consists of 100 iterations of the Baum-Welch procedure. It runs the HMM learning algorithm for some toy binary data and prints the parameters learned by the HMM (i.e. matrices A and B).

1.1.1 Question 1a: Qualitative Analysis (15 P)

- *Run* the code several times to check that the behavior is consistent.
- *Describe* qualitatively the solution A, B learned by the model.
- *Explain* how the solution $\lambda = (A, B)$ relates to the sequence of observations O that has been modeled.

```
In [2]: import numpy,hmm

        O = numpy.array([1,0,1,0,1,1,0,0,1,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,1,0,0,1,1,
                         0,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,
                         0,0,1,0,1,0,1,0,0,0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,
                         0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,
                         1,0,0,0,1,1,0,0,1,0,1,1,1,0,0,1,1,0,0,0,1,1,0,0,1,1,0,0,1,
                         0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,
                         0,0,1,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,0,0,0,1,1,0,0])

        hmmtoy = hmm.HMM(4,2)

        for k in range(100):
            hmmtoy.loaddata(O)
```

```
            hmmtoy.forward()
            hmmtoy.backward()
            hmmtoy.learn()

        print('A')
        print("\n".join([" ".join(['%.3f'%a for a in aa]) for aa in hmmtoy.A]))
        print('Supposing A as it is stated in the uploaded programming PDF in ISIS: A forms a c
             'since the transition probabilites are all equal to 0 or 1 and thus yield s1->s2->
        print('B')
        print("\n".join([" ".join(['%.3f'%b for b in bb]) for bb in hmmtoy.B]))
        print('The shape of B indicates that there are two possible outcomes of the HMM-process
             'outputs can be generated following the probilities stated in the rows of B, that
             'every hidden state produces two possible visible outcomes with varying probabili
        print('Pi')
        print("\n".join(['%.3f'%b for b in hmmtoy.Pi]))
        print('lambda is the most likely model or rather model parameters that reproduce the th
             'the best. That means the learning algorithm maximizes P(O|lambda) and the given l
             'maximization given O')
```

```
A
0.278 0.000 0.721 0.000
1.000 0.000 0.000 0.000
0.000 0.409 0.257 0.334
0.000 0.668 0.332 0.000
Supposing A as it is stated in the uploaded programming PDF in ISIS: A forms a circle of hidde
since the transition probabilites are all equal to 0 or 1 and thus yield s1->s2->s3->s4->s1.
B
0.867 0.133
0.000 1.000
0.953 0.047
0.002 0.998
The shape of B indicates that there are two possible outcomes of the HMM-process. According to
 outputs can be generated following the probilities stated in the rows of B, that is why the r
Pi
0.000
0.000
0.000
1.000
lambda is the most likely model or rather model parameters that reproduce the the sequence of
 the best. That means the learning algorithm maximizes P(O|lambda) and the given lamda if the
 maximization given O
```

1.1.2 Question 1b: Finding the best number N of hidden states (15 P)

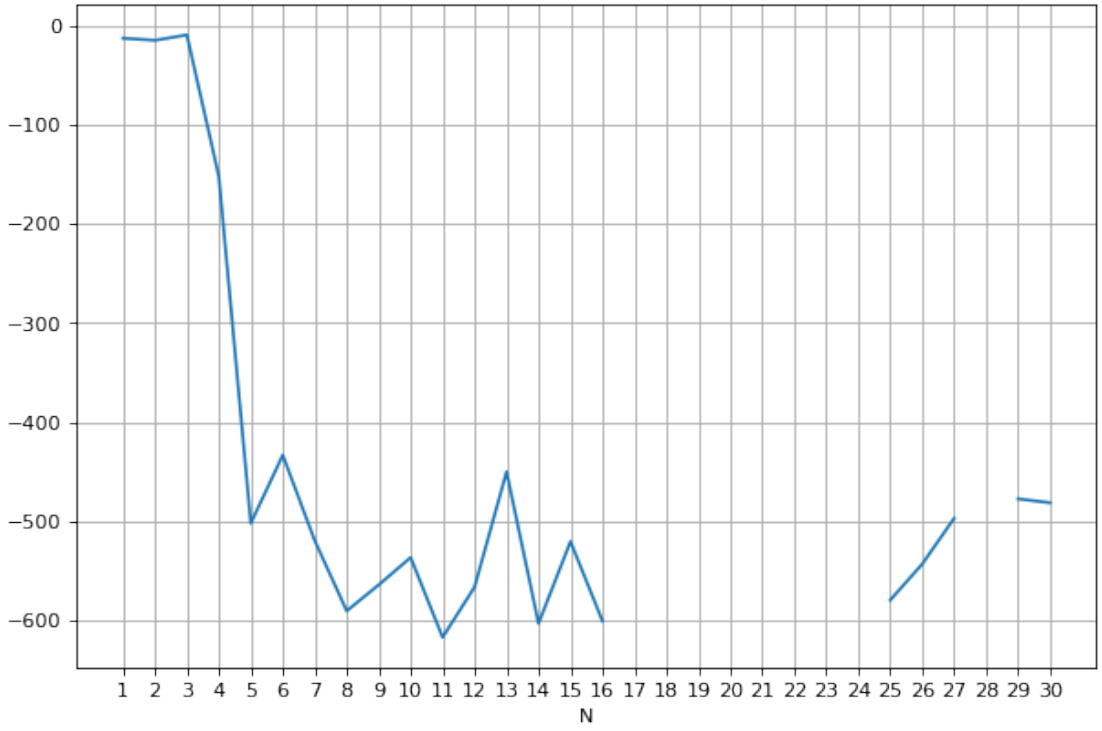
For the same sequence of observations as in Question 1a, we would like to determine automatically what is a good number of hidden states $N = \text{card}(S)$ for the model.

- *Split* the sequence of observations into a training and test set (you can assume stationarity).

- *Train* the model on the training set for several iteration (e.g. 100 iterations) and for multiple parameter N.
- *Show* for each choice of parameter N the log-probability $\log p(O|\lambda)$ for the test set. (If the results are unstable, perform several trials of the same experiment for each parameter N.)
- *Explain* in the light of this experiment what is the best parameter N.

```
In [11]: import matplotlib
         from matplotlib import pyplot as plt
         %matplotlib inline

         from IPython.display import clear_output, display
         import time

         # split into test and training set
         split_index = 180
         O_train = O[0:split_index]
         O_test = O[split_index+1:-1]   3.1

         N_max = 30
         log_p = numpy.zeros(N_max)

         for N in range(1, N_max + 1):
             start_time = time.time()

             hmmtoy = hmm.HMM(N,2)
             for k in range(100):
                 hmmtoy.loaddata(O_train)
                 hmmtoy.forward()
                 hmmtoy.backward()
                 hmmtoy.learn()
             A,B = hmmtoy.A, hmmtoy.B

             # initialization
             alpha_t = hmmtoy.Pi*B[:,O_test[0]]   3.2
             # induction
             for t in range(1,len(O_test)):
                 alpha_t_p_1 = alpha_t@A*B[:,O_test[t]]
                 alpha_t = alpha_t_p_1
             # termination
             log_p[N-1] = numpy.log(numpy.sum(alpha_t))

             clear_output(wait=True)
             print(f"Finished N = {N} in {time.time() - start_time:.2f} seconds.")

         print(log_p)                                   3.3

         fig = plt.figure(figsize=(9, 6), dpi=80)
         N_vec = numpy.arange(1,N_max+1)
```

```
        _ = plt.plot(N_vec, log_p)
        _ = plt.xlabel('N')
        _ = plt.xticks(N_vec)
        _ = plt.grid()

Finished N = 30 in 0.55 seconds.
[ -12.45830548  -14.57982066   -9.11962997 -151.94845182 -502.10188454
 -433.32773559 -519.60449349 -590.57804142 -564.32197344 -536.64437271
 -617.25530139 -566.46557861 -450.16541492 -603.2238342  -520.51626558
 -600.75616162          -inf          -inf -499.54449142          -inf
 -500.23664208          -inf -596.33066406          -inf -579.69979923
 -543.38162327 -497.05467703          -inf -477.45975757 -481.38973237]
```



It seems like the best parameter choice is $N = 3$ since the likely hood for this point is the highest. With more hidden states the prediction quality is decreasing extremely quickly and doesn't yield useful result.

1.2 Question 2: Text modeling and generation (30 P)

We would like to train an HMM on character sequences taken from English text. We use the 20 newsgroups dataset that is accessible via scikits-learn http://scikit-learn.org/stable/datasets/twenty_newsgroups.html. (For this, you need to install scikits-learn if not done already.) Documentation is available on the website. The code below allows you to

(1) read the dataset, (2) sample HMM-readable sequences from it, and (3) convert them back into string of characters.

```
In [92]: from sklearn.datasets import fetch_20newsgroups

         # Download a subset of the newsgroup dataset
         newsgroups_train = fetch_20newsgroups(subset='train',categories=['sci.med'])
         newsgroups_test  = fetch_20newsgroups(subset='test' ,categories=['sci.med'])

         # Sample a sequence of T characters from the dataset
         # that the HMM can read (0=whitespace 1-26=A-Z).
         #
         # Example of execution:
         # O = sample(newsgroups_train.data)
         # O = sample(newsgroups_test.data)
         #
         def sample(data,T=50):
             i = numpy.random.randint(len(data))
             O = data[i].upper().replace('\n',' ')
             O = numpy.array([ord(s) for s in O])
             O = numpy.maximum(O[(O>=65)*(O<90)+(O==32)]-64,0)
             j = numpy.random.randint(len(O)-T)
             return O[j:j+T]

         # Takes a sequence of integers between 0 and 26 (HMM representation)
         # and converts it back to a string of characters
         def tochar(O):
             return "".join(["%s"%chr(o) for o in (O+32*(O==0)+64*(O>0.5))])
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
```

1.2.1 Question 2a (15 P)

In order to train the HMM, we use a stochastic optimization algorithm where the Baum-Welch procedure is applied to randomly drawn sequences of $T = 50$ characters at each iteration. The HMM has 27 visible states (A-Z + whitespace) and 200 hidden states. Because the Baum-Welch procedure optimizes for the sequence taken as input, and no necessarily the full text, it can take fairly large steps in the parameter space, which is inadequate for stochastic optimization. We consider instead for the parameters $\lambda = (A, B, \Pi)$ the update rule $\lambda^{new} = (1 - \gamma)\lambda + \gamma\bar{\lambda}$, where $\bar{\lambda}$ contains the candidate parameters obtained from Equations 40a-c. A reasonable value for γ is 0.1.

- *Create* a new class `HMMChar` that extends the class `HMM` provided in `hmm.py`.
- *Implement* for this class a new method `HMMchar.learn(self)` that overrides the original methods, and implements the proposed update rule instead.
- *Implement* the stochastic training procedure and run it.

- *Monitor* $\log p(O|\lambda)$ on the test set at multiple iterations for sequences of same length as the one used for training. (Hint: for less noisy log-probability estimates, use several sequences or a moving average.)

```
In [97]: # Hidden Markov Model class
         class HMMChar(hmm.HMM):
             def learn(self):
                 pass

         hmmchar = HMMChar(200,27)
         trainsample = lambda: sample(newsgroups_train.data)
         testsample  = lambda: sample(newsgroups_test.data)

         #solutions.question2a(hmmchar,trainsample,testsample)
```

1.2.2 Question 2b (15 P)

In order to visualize what the HMM has learned, we would like to generate random text from it. A well-trained HMM should generate character sequences that have some similarity with the text it has been trained on.

- *Implement* a method `generate(self,T)` of the class `HMMChar` that takes as argument the length of the character sequence that has to be generated.
- *Test* your method by generating a sequence of 250 characters and comparing it with original text and a purely random sequence.
- *Discuss* how the generated sequences compare with written English and what are the advantages and limitations of the HMM for this problem.

```
In [ ]: print("original:\n"+tochar(sample(newsgroups_test.data,T=250)))
        print("\nlearned:\n"+tochar(hmmchar.generate(250)))
        print("\nrandom:\n" +tochar(solutions.HMMChar(200,27).generate(250)))
```

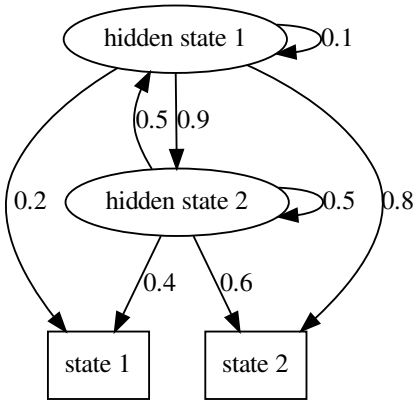
Machine Learning 2 Homework 7

Group: DLBSP

June 20, 2019

Exercise 1

a)



b)

To describe an experiment means to describe, what happens when we run the system drawn above. Each visible coin corresponds to one hidden coin (that means we can only toss the former when we have the latter in hand). At the beginning (in the first round) we start with the first hidden coin in hand. When we hold first hidden coin we must first toss a visible coin whose outcome probabilities are defined by the first row of B. The outcome of the toss is added to observation sequence (heads or tails). Then, independent of the outcome, we toss our hidden coin whose outcome probabilities are described in the first row of A and, depending on the outcome, change the hidden coin to the second one or stick with it. At the next round the same steps are executed: visible coin corresponding to the hidden coin is tossed that results in an observation and consequently the hidden coin is also tossed. The outcome of the latter determines whether we change the hidden coin or not.

The goal of an experiment is to obtain the sequence of observations, corresponding to the outcomes of tosses of visible coins at each round.

Coins are pure abstractions, they relate to transition probabilities and conditional distributions in the real world.

c)

The task is defined in very ambiguous way. Our interpretation of it is: compute probabilities for all possible sequences of states that produce sequence of observations (tails, tails).

Since initial state is always S_1 according to π the set of possible sequences is reduced to $(S_1,\ S_1)$, $(S_1,\ S_2)$. Bayes formular yields

$$P(Q|O) = \frac{P(O|Q)P(Q)}{P(O)}$$

For $O = (\text{tails}, \text{tails})$, $Q = (S_1, S_1)$

$$
\begin{aligned}
P(O|Q) &= p(\text{tails}|S_1)p(\text{tails}|S_2) = 0.8 \cdot 0.6 = 0.48 \\
P(Q) &= P(S_1, S_2) = 1 \cdot 0.9 = 0.9 \\
P(O = (\text{tails}, \text{tails})) &= \sum_{(S_1,S_1),(S_1,S_2)} P(O|Q)P(Q) = 0.48 \cdot 0.9 + 0.64 + 0.64 \cdot 0.1 = 0.496 \\
\Rightarrow P(Q|O) &= \frac{0.48 \cdot 0.9}{0.496} = 0.87
\end{aligned}
$$

For $O = (\text{tails}, \text{tails})$, $Q = (S_1, S_2)$

$$
\begin{aligned}
P(O) &= 0.496 \\
P(O|Q) &= 0.8 \cdot 0.8 = 0.64 \\
P(Q) &= 1 \cdot 0.1 = 0.1 \\
P(O = (\text{tails}, \text{tails})) &= \sum_{(S_1,S_1),(S_1,S_2)} P(O|Q)P(Q) = 0.48 \cdot 0.9 + 0.64 + 0.64 \cdot 0.1 = 0.496 \\
\Rightarrow P(Q|O) &= \frac{0.64 \cdot 0.1}{0.496} = 0.129
\end{aligned}
$$

Index of comments