TP3

June 9, 2018

1 SD-TSIA214 Machine Learning For Text Mining

2 Text segmentation using Hidden Markov Models

```
In [1]: import numpy as np
```

2.1 Task : automatic segmentation of mails

This Lab aims to build an email segmentation tool, dedicated to separate the email header from its body. It is proposed to perform this task by learning a HMM with two states, one (state 1) for the header, the other (state 2) for the body. In this model, it is assumed that each mail actually contains a header : the decoding necessarily begins in the state 1.

2.2 1. Give the value of the vector of the initial probabilities

It is said above that our HMM has necessarily an initial state of 1. So, the vector of the initial probabilities will have the following form:

$$\pi^T = (1, 0)$$

Since the HMM will be in state 0 with probability 0 and in state 1 with probability 1, i.e. always.

Knowing that each mail contains exactly one header and one body, each mail follows once the transition from 1 to 2. The transition matrix (A(i, j) = P(j | i)) estimated on a labeled small corpus has thus the following form :

```
In [2]: A = np.array([[0.999218078035812, 0.00078192196418797 4], [0, 1]])
        A

Out[2]: array([[9.99218078e-01, 7.81921964e-04],
               [0.00000000e+00, 1.00000000e+00]])
```

2.3 2. What is the probability to move from state 1 to state 2 ? What is the probability to remain in state 2 ? What is the lower/higher probability ? Try to explain why

Probabilities of movements between states can be found in the transition matrix A, which was just given. The i index (row index) determines the starting state, while the j index (column index) determines the arriving state. Thus:

- Probability to move from state 1 to state 2: A(1,2) =

```
In [3]: A[0,1]
```

```
Out[3]: 0.000781921964187974
```

- Probability to remain in state 2: A(2,2) =

```
In [4]: A[1,1]
```

```
Out[4]: 1.0
```

Probability to remain in state 2 is actually the highest obtainable probability (=1), and this makes sense in our use case: once our email is in the state corresponding to the body, we can't go back anymore to the header state with more iterations, that correspond to reading more characters. In fact, an email's body goes from its beginning to end of an email, and in no case is followed by another header. As a consequence, when the number of iterations/observation n goes to infinity, our observed Markov Chain will forcely be in state 1, meaning that - as stated before - we will necessarily be in the email's body.

On the contrary, the probability of moving from state 1 to state 2 is much lower because in average we have to read a significant number of characters - and, therefore, wait for several iterations - before being able to affirm that the email has also a body.

For example, if we had built our model and the transition matrix A using some strange kind of emails that have 3 characters in the header and (in average) more than 2000 in the body, this probability would be much higher. Obviously, the creation of the HMM and thus our probabilty also depend on the observations, which influence directly the hidden process and indirectly the observed one: if I had trained it with headers containing only 'a' characters and bodies that never contain the 'a' character, this would surely impact the transition probability matrix.

A mail is represented by a sequence of characters. Let N be the number of different characters. Each part of the mail is characterized by a discrete probability distribution on the characters P(c | s), with s = 1 or s = 2.

2.4 3. What is the size of the corresponding matrix ?

The size of the corrispondig matrix is

$$N \times |s|$$

Since we are encoding characters as ASCII values, which range from 1 to 256, the matrix will have a size of 256x2, namely 256 rows and 2 columns.

2.5 Material

2.5.1 Coding/decoding mails

Emails are represented as ASCII character vectors. In dat.zip, mail.txt in a vector of numbers (one vector per line). that can be transformed into a vector of numbers (between 0 and 255) in a text; Files of the form dat /*. dat contain the already encoded versions of the corresponding mails. The list is in mail.lst.

2.5.2 Visualizing segmentation

Finally, the utility segment.pl allows to visualize a segmentation produced by your segmenter in the form of the best path found by the Viterbi algorithm (in a vector of 1 and 2). It produces a file where the segmentation is visualized.

To use it :
perl segment.pl mail.txt path.dat

2.5.3 Distribution files

For the first part of the Lab, we work with the distributions that are provided in the P.dat file.

Each of the columns of this file contains the distribution of the probabilities of occurrence of each character of the ASCII codes respectively in the header and in the body. These distributions were learned on a small corpus labeled with 10 emails ; there are obvious differences, especially in areas where ASCII codes correspond to alphabetic characters, as you can see by viewing these distributions.

To implement: All the work is to be done under Python.

- implement the Viterbi algorithm. Concretely, it comes to coding a function which takes as argument a vector of observations and the parameters of the model, and returns a vector of states representing the most probable sequence.

```
In [5]: """
            Viterbi Algorithm Implementation

            Inputs:
                - obs: a 1-d nparray containing the observations. Its values must be lesser
                    or equal than the number of rows of the conditional probability matrix.
                - states: a 1-d nparray containing the number of possible states. Its size
                    has to be equal to the number of columns of the conditional probability
                    matrix.
                - trans: a 2-d nparray transition probability matrix containing the
                    transition probabilities for the observed process.
                - start_prob: a 1-d nparray containing the starting probabilities for the
                    observed process. Its length has to be equal to the number of possible
                    states.
                - cond_prob: also known as emission probability, it is a 2-d nparray
                    having a number of columns equal to the number of possible states.

            Returns:
                - seq: a 1-d nparray containing, for each observation given in the input,
                    the corresponding state that represent the most probable sequence

        """

        def viterbiAlgorithm(obs, states, trans, start_prob, cond_prob):
            # Rescaling function that avoids state probabilities going to zero
            # Computes the order of magnitude of values in the array's last row
```

```
    # and rescales them back to 10^-1
    def rescale(state_probs, last_row):
        # Compute order of magnitude
        om = int(np.log10(np.max(state_probs[last_row,:])))
        # Rescale
#         print('Before rescaling ' + str(state_probs[last_row, :])
#                     + ' - OM ' + str(om) + ' Resc factor ' + str((om+1)))
        state_probs[last_row, :] = state_probs[last_row, :]*(10**(-1*(om+1)))
#         print('After rescaling ' + str(state_probs[last_row, :]))
        return state_probs

    # State probabilities
    state_probs = np.zeros((len(obs), len(states)))
    # Sequence of states
    path = {}

    for i in range(0,len(observations)):

        # First iteration
        if i == 0:
            # Initialize states probabilites
            for state in range(0,len(states)):
                state_probs[i][state] = start_prob[state]\
                                        * cond_prob[obs[i]][state]
                path[state] = [states[state]]
#             print('States probabilities: ' + str(state_probs[i]))
        else:
            # Update
            tempPath = {}
            probs = np.zeros((len(states), len(states)))
            for arr_state in range(0, len(states)):
                (probs, state) = max([(state_probs[i-1][dep_state] \
                                       * trans[dep_state][arr_state]\
                                       * cond_prob[obs[i]][arr_state],
                                      dep_state) \
                                      for dep_state in range(0, len(states))])
#                 print('Arrival ' + str(arr_state) + ' - Probabilities : '
#                 + str((probs, state)))
                state_probs[i, arr_state] = probs
                tempPath[arr_state] = path[state] + [states[arr_state]]
#             print('Observation ' + str(i) + ' - Probabilities: ')
#             print(str(probs))
#             print('Max prob : ' + str(probs[r][c]) + ' - R, C : ' + str(r) + ', '
#             + str(c) + ' - State: ' + str(states[c]))
#             print('States probabilities: ' + str(state_probs[i]))
            path =  tempPath
            state_probs = rescale(state_probs, i)
    (finProb, state) = max([(state_probs[-1][s], s)\
```

```
                                    for s in range(0, len(states))])
        return path[state]
```

- test it on some mails that are given in the dat directory (especially mail11.txt to mail30.txt).

```
In [6]: def printPath(path):
            summary = []
            summary.append({ 'start': 0, 'end': 0, 'value': path[0]})
            for i in range(1, len(path)):
                if path[i] != path[i-1]:
                    summary.append({ 'start': i, 'end': i, 'value': path[i] })
                else:
                    summary[-1]['end'] = i

            for s in summary:
                print('State ' + str(s['value']) + ' observations '
                      + str(s['start']) + ' -> ' + str(s['end']))

        distributions = np.loadtxt('p.text', dtype=float)

        for i in range(11,31):
            observations = np.loadtxt('./dat/mail'+str(i)+'.dat', dtype=int)
            p = viterbiAlgorithm(obs=observations, states=[1,2], start_prob=[1,0], trans=A, co
            print('--- MAIL ' + str(i) + ' STATES : ')
            printPath(p)
```

```
--- MAIL 11 STATES :
State 1 observations 0 -> 2850
State 2 observations 2851 -> 3474
--- MAIL 12 STATES :
State 1 observations 0 -> 2937
State 2 observations 2938 -> 3992
--- MAIL 13 STATES :
State 1 observations 0 -> 2303
State 2 observations 2304 -> 3327
--- MAIL 14 STATES :
State 1 observations 0 -> 4812
State 2 observations 4813 -> 6575
--- MAIL 15 STATES :
State 1 observations 0 -> 2182
State 2 observations 2183 -> 6807
--- MAIL 16 STATES :
State 1 observations 0 -> 1970
State 2 observations 1971 -> 2626
--- MAIL 17 STATES :
State 1 observations 0 -> 2281
State 2 observations 2282 -> 3424
--- MAIL 18 STATES :
```

```
State 1 observations 0 -> 2366
State 2 observations 2367 -> 3076
--- MAIL 19 STATES :
State 1 observations 0 -> 2100
State 2 observations 2101 -> 2619
--- MAIL 20 STATES :
State 1 observations 0 -> 1839
State 2 observations 1840 -> 2433
--- MAIL 21 STATES :
State 1 observations 0 -> 2102
State 2 observations 2103 -> 2663
--- MAIL 22 STATES :
State 1 observations 0 -> 2233
State 2 observations 2234 -> 3642
--- MAIL 23 STATES :
State 1 observations 0 -> 2167
State 2 observations 2168 -> 3749
--- MAIL 24 STATES :
State 1 observations 0 -> 2559
State 2 observations 2560 -> 3700
--- MAIL 25 STATES :
State 1 observations 0 -> 2318
State 2 observations 2319 -> 3237
--- MAIL 26 STATES :
State 1 observations 0 -> 2026
State 2 observations 2027 -> 4466
--- MAIL 27 STATES :
State 1 observations 0 -> 1770
State 2 observations 1771 -> 3147
--- MAIL 28 STATES :
State 1 observations 0 -> 2224
State 2 observations 2225 -> 2540
--- MAIL 29 STATES :
State 1 observations 0 -> 2343
State 2 observations 2344 -> 2889
--- MAIL 30 STATES :
State 1 observations 0 -> 2172
State 2 observations 2173 -> 5159
```

2.6 4. Print the track and present and discuss the results obtained on mail11.txt to mail30.txt

The results seem consistent, first of all because of their values: our HMM starts in State 1, goes to State 2 after a considerable number of characters and never comes back to State 1.

Furthermore, I checked some emails and results are encouraging. For example, in mail11 our algorithm states that body starts at character 2850 while I found it beginning at character 2851: it could be my fault (I was using Atom editor to select characters and I'm not 100% sure of where it

starts/ends selecting them), but the results is anyhow very precise!

Obviously, there are some more significant errors, as in email14, where the model struggles a little to recognize the beginning of the body because it begins with a quote from a previous email that actually looks like a header. I think, however, that this is a limitation of the whole model: if I sent an email containing the header of another email in the body, the model will fail to distinguish between the real model and the copied header in the body.

2.7 5. How would you model the problem if you had to segment the mails in more than two parts (for example : header, body, signature) ?

In this case, our the hidden model wouldn't change since the possible values of our observations (characters) remain the same: instead, it would change the observed model which will now have three states (namely header, body and signature). The transition matrix will hence become a 3x3 matrix of this form:

$$\begin{bmatrix} p_{11} & p_{12} & 0 \\ 0 & p_{22} & p_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

and the initial vector :

$$^T = (1, 0, 0)$$

2.8 6. How would you model the problem of separating the portions of mail included, knowing that they always start with the character ">".

The model would now have four states, namely header, body_text, body_included, and signature, with a transition matrix 4x4 of this form:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & 0 \\ 0 & p_{22} & p_{23} & p_{24} \\ 0 & p_{32} & p_{33} & p_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the initial vector :

$$\pi^T = (1, 0, 0, 0)$$

where body_text and body_include are now respectively state 2 and 3, while signature becomes state 4.

The fact of knowing that included mail always start with a ">" character is an information that regards only the outcome of the observation, and would therefore influence not the observation model but the hidden one: the conditional probability vector will have a higher probability at the place corresponding to character ">" and state 3.

More precisely, if the vector of conditional probabilities is built only by looking if a character is present or not in the desired portion of speech, this probability would be equal to 1 for the class corresponding to included mails: since included mails always start with the character ">", the probability of having a ">" conditioned to the fact of being in class body_included is 1.