Machine Learning Project Report

Pang Zichen 1004525, Xiong Maihe 1004537, Lu Qianxi 1004533

Instruction

How to run the approach codes

- Part 1:
  ```
  python HMM_p1.py
  ```
- Part 2/3:
  ```
  python HMM_p23.py
  ```
- Part 4:
  ```
  python HMM_p4.py
  ```

How to run evaluation script

- Part 1:
  ```
  python evalResult.py ES/dev.out ES/dev.p1.out
  python evalResult.py RU/dev.out RU/dev.p1.out
  ```
- Part 2:
  ```
  python evalResult.py ES/dev.out ES/dev.p2.out
  python evalResult.py RU/dev.out RU/dev.p2.out
  ```
- Part 3:
  ```
  python evalResult.py ES/dev.out ES/dev.p3.out
  python evalResult.py RU/dev.out RU/dev.p3.out
  ```
- Part 4:
  ```
  python evalResult.py ES/dev.out ES/dev.p4.out
  python evalResult.py RU/dev.out RU/dev.p4.out

  For test data
  python evalResult.py <gold standard> ES/test.p4.out
  python evalResult.py <gold standard> RU/test.p4.out
  Replace the <gold standard> with your gold standard for
  evaluation, path should be included.
  ```

Approaches

Part 1

- Estimate emission parameters
  - Before adding the #UNK#

    First, we deal with the case without unknown observations (i.e. #UNK#). We count the number of labels (y) in the training set and the number of patterns that label y generates observation x, then store them in dictionaries respectively:
    - Dictionary storing y: key:y, value: count(y)
    - Dictionary storing y to x: key: y+ Chinese character +x

Here we use Chinese characters to separate labels and observations since there is definitely no Chinese character in the datasets. After counting we store the fractions count(y→x)/count(y) to our output.

  - After adding the #UNK#
    Then we deal with the cases with unknown observations. After counting count(y→x) and count(y), we save count(y→x)/ (count(y)+k) to our output. And we save k/(count(y)+k) for all label ys where the transition pattern is y→#UNK#.
  - Prediction
    We import x value for each observation and find y=argmax$_y$ e(x|y). The output y sequence is the label sequence we want.

Part 2

- Estimate transition parameters
  We initialize a list of labels with START state. In each line of the dataset, if the line is empty, meaning that it is the end of the previous sentence and the start of the next sentence, thus we append STOP state and START state consequently in empty lines. In word lines, we strip and split the line to get the label and update the label list. After going through all lines in the dataset, append one more STOP state at the end of the label list. Then we count the number of times we see the transition of each pair of states. Lastly, we calculate the transition probabilities by dividing the number of each transition pattern by the number of the first state in each transition pattern. This function returns the transition parameters as a dictionary with keys of transition patterns and the corresponding value of transition parameters.
- Viterbi algorithm
  We define a function Forward() to get the pi scores of each node. In the base case, the value is set to 1 only when the state is START. For position 1 to n, where n is the length of the sentence, we calculate the pi scores by maximizing the score over all states. We define a function backtracking() to get the sequence of labels based on the pi scores we get in Forward(). In function viterbi(), firstly we go through each line in the dataset to get the sentences. we use a temporary list to store the word lines, once an empty line appears, the temporary list is added to the observation list which contains all sentences in the end. In each sentence, we get the pi scores by calling Forward () and the label sequence of the sentence by calling backtracking (). After getting all the label sequences of the whole dataset, we return an output list consisting of all the observations and the corresponding label in label sequences. If

the observation is an empty line, the corresponding element in the output list is also an empty line.

Part 3

We have made changes based on the Viterbi algorithm.
Basic idea: For the node at position k-1, suppose we can find the first to fifth possible paths from START to the end point. Then, for each node at position k, we can calculate the first to fifth possible paths from START to the endpoint based on the distance between all START and k-1 nodes and a, b. After that, the process is repeated for the remaining nodes.
In our design, these data are maintained by the dictionary scores in the Forwarding process, and the structure is (position, node, serial) = score.

Base Case 1: START→k=1
This situation is special because START has only one node. Compared with the normal situation, 5×7=35 sets of data are calculated. In this case, we only need to set all the data to 1.

Base Case 2: k=1→k=2
This situation is special because the probability of k=1 is the same for every node. If we do not deal with it, the first five possible paths will definitely be repeated. Therefore, in order to avoid this problem, we treat it specially, treating the first to fifth probabilities of a single node as a single case.

Recursive Case: k-1→k
For the node at position k-1, we know the first to fifth possible paths from START to the endpoint (position, node, serial) = score, where serial = 1...5. For each node at position k, we need to calculate the distance from all the nodes at position k-1 to the node and find the best five records.

Part 4

Instead of using the Unigram, we switched to Bigram, meaning that instead of only generating from the previous state, we are generating from the previous 2 states. Now the model used is the second-order Hidden Markov Model.

- The way to calculation a value (transition parameter) and b value (emission parameter) becomes the following (b remains the same):

  $a_{u,v,w}$=Count(u,v,w)/Count(u,v)

  $b_w(x)$=Count(w->x)/Count(w)

- For the Viterbi algorithm:
  1. Moving forward:

     For n=1:

$\pi(0, \text{Start}, u) = 1$ if $u = \text{Start}$, 0 otherwise

$\pi(1, \text{Start}, w) = a_{START,START,w} \times b_w(x_1)$

Final step:

$\pi(2, w, STOP) = \max \{\pi(1, START, w) \times a_{v,w,STOP} \}$

For $n>1$:

Base case:

$\pi(0, \text{Start}, u) = 1$ if $u = \text{Start}$, 0 otherwise

$\pi(1, \text{Start}, w) = a_{START,START,w} \times b_w(x_1)$

For $j=1$ to $j= n-1$:

$\pi(j+1, v, w) = \max_v \{\pi(j, u, v) \times a_{u,v,w} \times b_w(x_{j+1})\}$

Final step:

$\pi(n+1, w, STOP) = \max_v \{\pi(n, v, w) \times a_{v,w,STOP} \}$

2. Backtracking

For $n=1$:

$y1=\text{argmax}_w \{\pi(1, START, w) \times a_{START,w,STOP} \}$

For $n>1$:

Y_n: $y_n= \text{argmax}_w \{\pi(n, v, w) \times a_{v,w,STOP} \}$

y_2 to y_{n-1}: $yi = \text{argmax}_w \{\pi(i, v, w) \times a_{v,w,yi+1} \}$

y_1: $y_1= \text{argmax}_w \{\pi(1, START, w) \times a_{START,w,yi+1} \}$

Results

Table 1: Results of entity analysis

Language	Entity	Precision	Recall	F
ES	Part 1	0.1182	0.8039	0.2061
	Part 2	0.2377	0.5137	0.3251
	Part 3	0.0833	0.2627	0.1265
	Part 4	0.5395	0.4824	0.5093
RU	Part 1	0.1604	0.7267	0.2627
	Part 2	0.4117	0.4751	0.4411
	Part 3	0.1317	0.2625	0.1754
	Part 4	0.4870	0.4056	0.4426

Table 2: Results of sentiment analysis

Language	Sentiment	Precision	Recall	F
ES	Part 1	0.0646	0.4392	0.1126
	Part 2	0.1887	0.4078	0.2581
	Part 3	0.0510	0.1608	0.0774
	Part 4	0.4430	0.3961	0.4182
RU	Part 1	0.0651	0.2950	0.1067
	Part 2	0.2707	0.3124	0.2900
	Part 3	0.0696	0.1388	0.0928
	Part 4	0.3229	0.2690	0.2935