PROJECT 1 - DESIGN DOCUMENTATION

Revision 1.0
Updated: Keyang Li(z5167157)
Lingxu Meng(z5147810)

Table Of Contents

Revision History

Date	Version	Writer	Comments
21st April 2019	1.0	Keyang Li; Lingxu Meng	Question 1
23rd April 2019	2.0	Keyang Li; Lingxu Meng	Question 2

Goal of the document

A report to illustrate:

- Implementation details of Q1
- Details on how you extended the Viterbi algorithm (Q1) to return the top-k state sequences (for Q2).

Question 1 - Viterbi algorithm for Address Parsing

File Processing

Three files in each data set need to be processed, which are:

1. State_File
2. Symbol_File
3. Query_File

We need to extract related information for further analysis. For both State_File and Symbol_File, we know that the first line is the number ***M*** of states and symbols respectively. Hence, we can easily grab a list of state and symbol 'ID' from the number from next ***M*** lines. And the last lines are the transaction between each dimension.
So, we can easily use a simple function ***split_file()*** to process these two files. The sample code is here.
From this function, we can get: ***the number of states(symbols), a list of state(symbol) ID, a list of transactions.***

As for the Query_File, we need to split each query sentence into a token list, an important principle that need to be noticed is the specific symbols. Each token should be and only should be split by space and these symbols, which are: ***, / - ( ) &*** , a set length of six.
To implement this part, we used Python regular expression to split the query in an 'elegant' way, the sample code is here.
From this function ***get_query_tokens()***, we can get: ***a list of query tokens.***

Additionally, tokens cannot attend the calculation, plus, some tokens could be unknown tokens for the symbol list. For further analysis, we made some transmission to the list, for known symbols, we replace the token with the index in the symbol list; for unknown symbols, we just set it to ***-1***. The sample code is here.

Smoothing

Because of the existence of unknown symbols and unconnected states/symbols. We need to smooth the possibilities.
To implement this process, we used the algorithm on the project specification and there is no need to repeat it again here, we put some code as an example during smoothing the transition probabilities. As for emission probabilities, the principle is almost the same.

After processing two lists of transactions, we get two two-dimensional arrays. The possibilities of transaction and emission are A and B respectively in Viterbi algorithm, and π is the sublist in A which represents the possibilities from $BEGIN$ to each states.

With these three parameters, we are ready to implement the algorithm and get the result.

Viterbi Algorithm

For explain, we use S represents the number of states, S_t represents the number of states, S_m represents the number of symbols(including unknown symbols).
After above steps, we acquire:

1. Transition probability which is a S_tS_t matrix, named A
2. Emission probability which is a $S_t(S_m+1)$, named B

For each query q_i, the length of it is l_i .
We name the probability of transition between $BEGIN$ to ***first symbol*** as π_i.

For a model $\lambda=(A,B,\pi)$, we observe $O=(o_1,o_2,\cdots,o_T)$ and get the best path $I^*=(i_1^*,i_2^*,\cdots,i_T^*)$.
For formula, $Pr((i_1,i_2,...i_c)|(o_1,o_2,...,o_c))=Pr((o_1,o_2,...,o_{c-1}),(i_1,i_2,...i_{c-1}))Pr(o_c|o_{c-1})Pr(i_c|o_c)$
δ(named as ***delta*** in our code) is the $(S_t-2)(l_i+1)$ matrix to record each possibility (sum of log) at each time at each state.

ψ(named as ***phsis*** in our code) is the $(S_t-2)l_i$ matrix to record the state we are passing through.
For each symbol in query, we calculate the value for different states as below:

On the initial stage of DP, parsing o_1, this process is:
$\delta_1(i)=\pi_i b_i\left(o_1\right)=\pi_i b_i$

As an example, when we parse o_2, we need to calculate the largest possibilities for each state and record in δ, which means, we need to find the largest possibility in precedent states and add possibilities at this stage:
$\delta_2(i)=\max\limits_{1\in j<len(A)}\left[\delta_1(j)a_{ji}\right]b_i\left(o_2\right)$
And we need to record the state in ψ to for future traceback.
$\psi_2(i)=\arg\max\limits_{1\leqslant j\leqslant len(A)}\left[\delta_1(j)a_{ji}\right]$
For next tokens, we repeat this process and try to get the largest possibilities, so on and so forth.

It is obvious that we get the best precedence in ψ, and the best probability in δ.

To get the entire path, we need to trace back by ψ, which means the previous best precedence is stored in the current observation's best state in ψ.

The method is pretty similar to our second lab. We use these simple but straightforward code to tell the story.

Question 2 - Extending Viterbi for top-k Parsing

Extended Viterbi Algorithm

As we have explained above.

For a model $\lambda = (A, B, \pi)$, we observe $O = (o_1, o_2, \cdots, o_T)$ and get the k-best paths $I^* = (I_1^*, I_2^*, \cdots, I_k^*)$.
For formula, $Pr((i_1, i_2, ...i_c)|(o_1, o_2, ..., o_c)) = Pr((o_1, o_2, ..., o_{c-1}), (i_1, i_2, ...i_{c-1}))Pr(o_c|o_{c-1})Pr(i_c|o_c)$
However, in this stage, it is necessary to calculate k best instead of the best.
It is straightforward that we can follow the algorithm we used in trivial viterbi algorithm.
δ(named as ***delta*** in our code) is the 3-Dimensions matrix to record each possibility (sum of log) at each time at each state.

ψ(named as ***phsis*** in our code) is the 4-Dimensions matrix to record the state we are passing through.
For each symbol in query, we calculate the value for different states as below:

On the initial stage of DP, parsing o_1, this process is:
$\delta_1(i) = \pi_i b_i(o_1) = \pi_i b_i$

As an example, when we parse o_2, we need to calculate the largest possibilities for each state and record in δ, which means, we need to find the largest possibility in precedent states and add possibilities at this stage:
$\delta_2(i) = \max_{1 \in j < len(A)} [\delta_1(j) a_{ji}] b_i(o_2)$
And we need to record the state in ψ to for future traceback.
$\psi_2(i) = \arg \max_{1 \leqslant j \leqslant len(A)} [\delta_1(j) a_{ji}]$
For next tokens, we repeat this process and try to get the largest possibilities, so on and so forth.

It is apparently that we get the best precedence in ψ, and k- best probability in δ.
To get the entire path, we need to trace back by ψ, which means the previous best precedence is stored in the current observation's best state in ψ

Question 3 - Advanced Decoding

TBD

Reference & Appendix

Relative Code

1.1

```
def split_file(file):
    f = open(file, 'r')
    lines = f.readlines()
    lines = ([l.strip('\n') for l in lines])
    file_num = int(lines[0])              # how many ids in the file
    id_list = lines[1:file_num+1]         # id list
    frequency = lines[file_num+1:]        # frequency list
    return file_num, id_list, frequency
```

1.2

```
lines = ([l.strip('\n') for l in lines])
    symbol_set = set(symbol_ids)
    for i, x in enumerate(lines):
        tmp_line = lines[i].split()
        for index, item in enumerate(tmp_line):
            # special_char: , / - ( ) & split each query by them
            tmp_line[index] = re.split(r'([,()/&-])', item)
        lines[i] = []
        for index_outer in range(len(tmp_line)):
            for index_inner in range(len(tmp_line[index_outer])):
                lines[i].append(tmp_line[index_outer][index_inner])
        # lines[i] = re.split('([^a-zA-Z0-9.?])', x)
```

```
        lines[i] = list(filter(lambda x: x != '', lines[i]))
        lines[i] = list(filter(lambda x: x != ' ', lines[i]))
```

1.3

```
for j, word in enumerate(lines[i]):
    if word not in symbol_set:
        lines[i][j] = -1
    else:
        lines[i][j] = symbol_ids.index(lines[i][j])
```

1.4

```
transition_frequency = np.zeros((state_num, state_num))
    state_probability = np.zeros((state_num, state_num))
    for f in state_fre:
        s_1, s_2, times = list(map(int, f.split(" ")))
        transition_frequency[s_1][s_2] = times
    for i in range(len(transition_frequency)):
        sum_i = sum(transition_frequency[i])
        if i == state_num-1:
        # j is end state, keep A[END, j] be 0
            continue
        else:
            for j in range(len(transition_frequency[i])):
                if j == state_num-2:
                    # i is begin state, keep A[i,BEGIN] be 0
                    continue
                else:
                    state_probability[i][j] = (transition_frequency[i][j] + 1) / (sum_i + state_num - 1)

    return state_probability
```

Requirement

numpy	1.8.0rc1
math	built-in
re	built-in

Human Cost

We spent around 9 days on this project, including:

1. Read specification and relative knowledge: 2 days
2. Make project design: 1day
3. Implement Question 1: 1 day
4. Implement Question 2: 3 days
5. Debug by referring autotest and Piazza: 1day
6. Write documentation: 1 day