Machine Discovery Homework 1-2

Student Name and ID

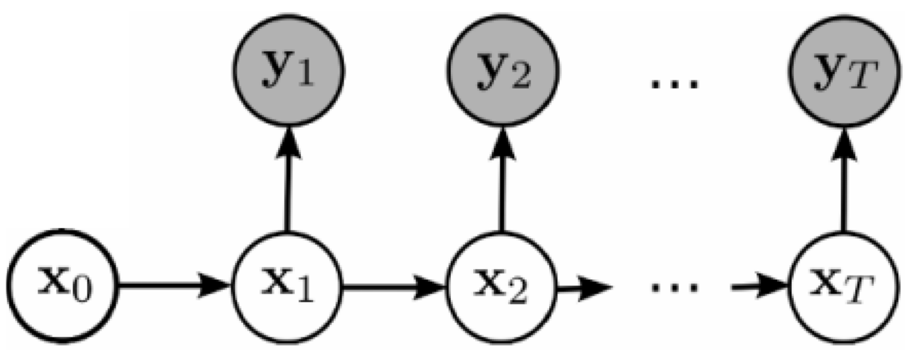
- 簡瑋德
- B03902015

Description

- Given some constraint about encoding probabilities, design a model to decode a text file.

Framework

- Architecture
  - Select a random seed and initialize the parameters of Bigram Language Model and Probabilistic Encoding Function
  - Use the test data (observations) to optimize the parameters, using Forward-backward Algorithm
  - Repeat the optimization until convergence or for specific rounds
  - Constuct the best prediction with the final parameters, using Viterbi Algorithm
  - It's reasonable to try another random seed and restart
- Assumption
  - Bigram Language Model: $P(w_1, w_2, ..., w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2)...P(w_n|w_{n-1})$
  - Probabilistic Encoding Function
- Probalilistic Graphical Model



  - $\forall x_t, y_t \in X$, where X consists of some symbols representing the characters (numbers 0 to $N-1$) and the space (number N)
  - x_0 is the random variable denoting the symbol in front of the word (assumed to be space), and $x_1, x_2, ..., x_T$ are the random variables of predicted symbols within a word
  - $y_1, y_2, ..., y_T$ are the random variables of observed symbols within a word
- Forward-backward Algorithm
  - Alpha Table
    - $\alpha(t, i)$ denotes $P(y_1, y_2, .., y_t, x_t = i|B, E)$, where B, E is the parameters of Bigram Language Model and Probabilistic Encoding Function
    - α table can be implemented by
      - $\alpha(1, i) = B(N, i) \times E(i, y_1)$, for $0 \leq i \leq N$
      - $\alpha(t+1, j) = [\sum_{i=0}^{N} \alpha(t, i) \times B(i, j)] \times E(j, y_{t+1})$, for every t and $0 \leq j \leq N$
  - Beta Table
    - $\beta(t, i)$ denotes $P(y_{t+1}, y_{t+2}, .., y_T|x_t = i, B, E)$, where B, E is the parameters of Bigram Language Model and Probabilistic Encoding Function
    - β table can be implemented by
      - $\beta(T, i) = 1$, for $0 \leq i \leq N$
      - $\beta(t, i) = \sum_{j=1}^{N} B(i, j) \times E(j, y_{t+1}) \times \beta(t+1, j)$, for every t and $0 \leq i \leq N$
  - Updating the Count Table
    - For Encoding Function
      $E \Rightarrow P(x_t = i|y_1, y_2, ..., y_T, B, E) = \frac{P(y_1, y_2, ..., y_T, x_t = i|B, E)}{P(y_1, y_2, ..., y_T|B, E)} \propto P(y_1, y_2, ..., y_T, x_t = i|B, E) = \alpha(t, i) \times \beta(t, i)$
    - For Bigram
      $B \Rightarrow P(x_t = i, x_{t+1} = j|y_1, y_2, ..., y_T, B, E) \propto P(y_1, y_2, ..., y_T, x_t = i, x_{t+1} = j|B, E) = \alpha(t, i) \times B(i, j) \times E(j, y_{t+1}) \times \beta(t, i)$
    - Add the occruance with probability to the count table and do the normolization
- Computational Issues
  - Underflow
    - Sice $p_1 p_2 ... p_n$ tends to underflow, we compute the probabilities under log space

        - $\log(p_1 p_2) = \log(p_1) + \log(p_2)$
        - $\log(p_1 + p_2) = \log(p_2)$ if $\log(p_1) == -\infty$, otherwise, $\log(1 + e^{\log(p_2) - \log(p_1)})$
    - Zero Probabilities
        - In Java, `Double.NEGATIVE_INFINITY` is usefull
        - For example, `Math.log(0.0)` is equal to `Double.NEGATIVE_INFINITY` , and `Math.exp(Double.NEGATIVE_INFINITY)` is equal to `0.0`
        - Moreover, `Double.NEGATIVE_INFINITY` + `Double.NEGATIVE_INFINITY` is still `Double.NEGATIVE_INFINITY` , so we do not need to deal with overflow.
        - However, it's still need to deal with `Division by Zero` if necessary
    - Parallel
        - Since Alpha table and Beta table can be calculated simultaneously, multi-threads is a solution to speed up the process
        - In Java, I simply use the instance `ExecutorService` and `Runnable` interface to calculate the tables

Settings and Configuration

- `used-tools.txt` : A list of third-part tools
- `report.pdf` : The report of the homework
- `README.txt` : Instructions to execute the program
- `src/` : Source codes
- `bin/` : Java compiled class files
- `valid/` and `valid2/` : Test data with answer
    - `encode.bin` : Text file of the encoding table
    - `test.num` : File of the test data
    - `ans.num` : The answer of the text data
    - `pred.txt` : The prediction of the test data, generated by the model
- `test1/` and `test2/` : Test data without answer
    - `encode.bin` : Text file of the encoding table
    - `test.num` : File of the test data
    - `pred.txt` : The prediction of the test data, generated by the model
- `Makefile` : Makefile for Linux
- Compile and Run:
    - Prerequisite
        - JDK/JRE-1.8
    - Makefile is available
        - `B03902015$ make`
            - Compile the source code in `src/` to `bin/`
        - `B03902015$ make run_valid1`
            - Input: `./valid/encode.bin` and `./valid/test.num`
            - Ouptut: `./pred.num` and the accuracy dumped by standard-out
        - `B03902015$ make run_valid2`
            - Input: `./valid2/encode.bin` and `./valid2/test.num`
            - Ouptut: `./pred.num` and the accuracy dumped by standard-out
        - `B03902015$ make run_test1`
            - Input: `./test1/encode.bin` and `./test1/test.num`
            - Ouptut: `./pred.num`
        - `B03902015$ make run_test2`
            - Input: `./test2/encode.bin` and `./test2/test.num`
            - Ouptut: `./pred.num`
    - Commands
        - `B03902015$ javac -d bin -sourcepath src src/launch/Main.java`
        - `B03902015$ java -Xmx4096M -cp bin launch.Main train ./valid/encode.bin ./valid/test.num`
            - Input: `./valid/encode.bin` and `./valid/test.num`
            - Ouptut: `./pred.num`
            - It's avaliable to modify the input arguments when testing different data sets

        - `B03902015$ java -Xmx4096M -cp bin launch.Main valid ./pred.num ./valid/ans.num`
            - Input: `./pred.num` and `./valid/ans.num`
            - Output: The accuracy dumped by standard-out
            - Optional, since there mihgt not be an answer file
    - The process will generate `./pred.txt` according to the given test data and it takes about 30 minutes and at most 2G RAM
- Screenshot

References

- Forward-backward Algorithm
- Viterbi Algorithm