Information Extraction Project 3

Tianyi Chen*

May 11, 2017

1 Code compile and usage

A **C++** program has been implemented for handling this project. To compile the **C++** program, in project directory, run the below command in terminal.

```
make
```

Successful compiling can generate an executable program called **test_tree**. **test_tree** can deliver all required experiments with suitable setting. The usage of running **test_tree** is as the below:

```
./test_tree -m mode
```

where option **-m** can control the which part of experiment to run, feasible values are in $\{1, 2, 3\}$

- **1**: Part 1: Construct agglomerative clustering tree.
- **2**: Part 2: Construct Bit-encoding based decision tree.
- **3**: Part 3: Construct Chou's decision tree.

2 Agglomerative clustering tree

To complete the required experiment, run the below command in terminal

```
./test_tree -m 1
```

We get the below output:

```
Initialize agglomerative clustering tree...
Agglomerative clustering tree is growing...
Print out growed agglomerative clustering tree...
level is 9,  letters: o, a, u, i, e,  , z, v, k, m, f, q, j, b, p, w, c, t, h, x, r, l, n, y, s, g, d,
level is 4,  letters: o, a, u, i, e,  ,
level is 8,  letters: z, v, k, m, f, q, j, b, p, w, c, t, h, x, r, l, n, y, s, g, d,
level is 3,  letters: o, a, u, i, e,
level is 0,  letters:  ,
level is 7,  letters: z, v, k, m, f, q, j, b, p, w, c, t, h,
level is 4,  letters: x, r, l, n, y, s, g, d,
level is 2,  letters: o, a, u, i,
level is 0,  letters: e,
level is 6,  letters: z, v, k, m, f, q, j, b, p, w, c, t,
level is 0,  letters: h,
level is 3,  letters: x, r, l, n,
level is 2,  letters: y, s, g, d,
level is 1,  letters: o, a,
level is 1,  letters: u, i,
level is 5,  letters: z, v, k, m, f, q, j, b, p,
level is 2,  letters: w, c, t,
level is 2,  letters: x, r, l,
```

*Email: tchen59@jhu.edu.

```
level is 0,  letters: n,
level is 1,  letters: y, s,
level is 1,  letters: g, d,
level is 0,  letters: o,
level is 0,  letters: a,
level is 0,  letters: u,
level is 0,  letters: i,
level is 4,  letters: z, v, k, m, f,
level is 3,  letters: q, j, b, p,
level is 1,  letters: w, c,
level is 0,  letters: t,
level is 1,  letters: x, r,
level is 0,  letters: l,
level is 0,  letters: y,
level is 0,  letters: s,
level is 0,  letters: g,
level is 0,  letters: d,
level is 3,  letters: z, v, k, m,
level is 0,  letters: f,
level is 2,  letters: q, j, b,
level is 0,  letters: p,
level is 0,  letters: w,
level is 0,  letters: c,
level is 0,  letters: x,
level is 0,  letters: r,
level is 2,  letters: z, v, k,
level is 0,  letters: m,
level is 1,  letters: q, j,
level is 0,  letters: b,
level is 1,  letters: z, v,
level is 0,  letters: k,
level is 0,  letters: q,
level is 0,  letters: j,
level is 0,  letters: z,
level is 0,  letters: v,
```

For the questions shown in the description of this project. The below are the answers.

2.1 Drawing clustering tree

The clustering tree after growing is shown in Figure 1 based on the information of output shown in Section 2.

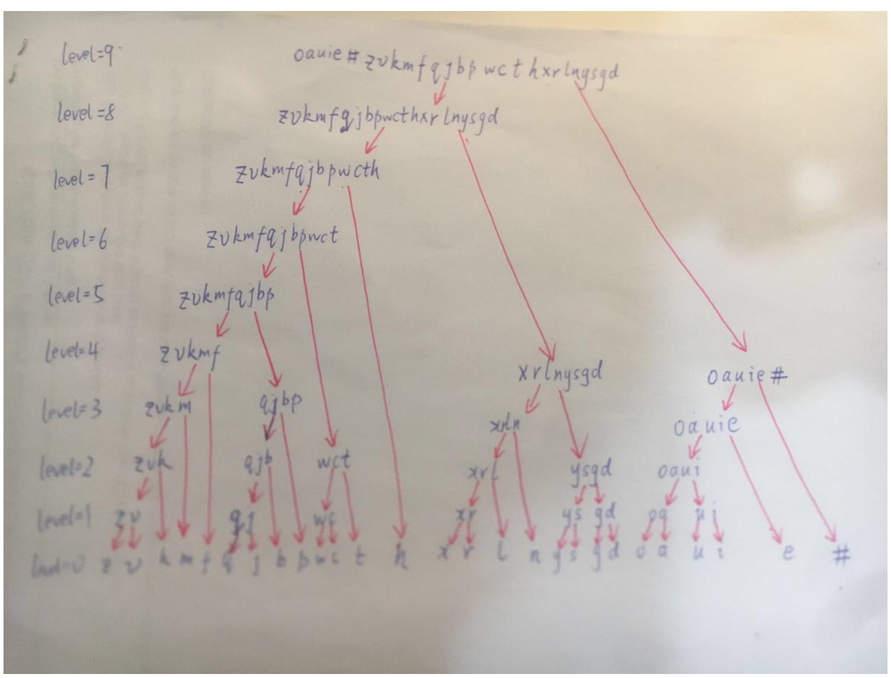

Figure 1: Clustering tree

2.2 The best 2-way clustering of letters

As shown in Figure 1, the best 2-way clustering of letters are $\{z, v, k, m, f, q, j, b, p, w, c, t, h, x, r, l, n, y, s, g, d\}$ and $\{o, a, u, i, e, \#\}$.

In Project 1, we found that in 2-state HMM, vowel and consonant have the same state with large probabilities. The best 2-way clustering is consistent with this observation in Project 1.

In Homework #9, we found that $a, e, i, o, u, \#$ have larger probabilities to belong to the same set by Chou's algorithm. The best 2-way clustering is also consistent with the observation in Homework #9.

2.3 The best 4-way clustering of letters

By the output shown in Section 2, we have the best 4-way clustering of letters is $\{o, a, u, i, e\}$, $\{\#\}$, $\{z, v, k, m, f, q, j, b, p, w, c, t, h\}$, and $\{x, r, l, n, y, s, g, d\}$.

It is also consistent with the observation in 4-state HMM in Project 1 that the letters in the same set have higher probabilities with the same state in 4-state HMM.

3 Bit-encoding based decision tree

To complete the required experiment, run the below command in terminal

```
./test_tree -m 2
```

```
Constructing agglomerative clustering tree...
Construct bit encoding decision tree...
Encoding l1, l2, l3, l4...
Build development, and held-out data sets...
Bit encoding decision tree is growing...
Process: ..............................................................................................................
No frontier node...
Bit tree growing complete...
Print out growed bit encoding decision tree...
path: , questioned bit 18
path: 0, questioned bit 19
path: 1, questioned bit 19
path: 00, questioned bit 20
path: 01, questioned bit 9
path: 10, questioned bit 9
path: 11, questioned bit 9
path: 000, questioned bit 21
path: 001, questioned bit 9
path: 010, questioned bit 0
path: 011, questioned bit 10
path: 100, questioned bit 10
path: 101, questioned bit 10
path: 110, questioned bit 10
path: 111, questioned bit 20
path: 0000, questioned bit 9
path: 0001, questioned bit 9
path: 0010, questioned bit 10
path: 0011, questioned bit 10
path: 0100, questioned bit 1
path: 0101, questioned bit 1
path: 0110, questioned bit 0
path: 0111, questioned bit 0
path: 1000, questioned bit 0
path: 1001, questioned bit 20
path: 1010, questioned bit 20
path: 1011, questioned bit 20
path: 1100, questioned bit 20
path: 1101, questioned bit 20
path: 1110, questioned bit 10
path: 1111, questioned bit 21
path: 00000, questioned bit 22
path: 00001, questioned bit 22
path: 00010, questioned bit 10
path: 00011, questioned bit 22
```

```
path: 00100, questioned bit 11
path: 00101, questioned bit 0
path: 00110, questioned bit 11
path: 00111, questioned bit 11
path: 01100, questioned bit 1
path: 01101, questioned bit 11
path: 10000, questioned bit 1
path: 10001, questioned bit 11
path: 10010, questioned bit 21
path: 10011, questioned bit 0
path: 10100, questioned bit 0
path: 10101, questioned bit 0
path: 11000, questioned bit 21
path: 11001, questioned bit 21
path: 11110, questioned bit 22
path: 11111, questioned bit 22
path: 000000, questioned bit 10
path: 000001, questioned bit 10
path: 000010, questioned bit 10
path: 000011, questioned bit 0
path: 000100, questioned bit 22
path: 000101, questioned bit 22
path: 000110, questioned bit 10
path: 000111, questioned bit 10
path: 001100, questioned bit 0
path: 001101, questioned bit 0
path: 001110, questioned bit 12
path: 001111, questioned bit 0
path: 100100, questioned bit 22
path: 100101, questioned bit 22
path: 110000, questioned bit 22
path: 110001, questioned bit 0
path: 110010, questioned bit 22
path: 110011, questioned bit 22
path: 0000000, questioned bit 11
path: 0000001, questioned bit 0
path: 0000010, questioned bit 11
path: 0000011, questioned bit 0
path: 0000100, questioned bit 0
path: 0000101, questioned bit 11
path: 0000110, questioned bit 1
path: 0000111, questioned bit 1
path: 0001110, questioned bit 11
path: 0001111, questioned bit 0
path: 0011100, questioned bit 13
path: 0011101, questioned bit 0
path: 1001000, questioned bit 23
path: 1001001, questioned bit 23
path: 1001010, questioned bit 23
path: 1001011, questioned bit 0
path: 1100000, questioned bit 23
path: 1100001, questioned bit 0
path: 1100010, questioned bit 1
path: 1100011, questioned bit 11
path: 1100100, questioned bit 11
path: 1100101, questioned bit 0
path: 1100110, questioned bit 11
path: 1100111, questioned bit 11
path: 00000100, questioned bit 12
path: 00000101, questioned bit 0
path: 00001100, questioned bit 10
path: 00001101, questioned bit 10
path: 00111000, questioned bit 0
path: 00111001, questioned bit 0
path: 10010010, questioned bit 24
path: 10010011, questioned bit 0
path: 11000000, questioned bit 0
path: 11000001, questioned bit 11
path: 11000010, questioned bit 1
path: 11000011, questioned bit 1
path: 11000100, questioned bit 2
path: 11000101, questioned bit 11
path: 11000110, questioned bit 12
path: 11000111, questioned bit 1
path: 11001110, questioned bit 0
path: 11001111, questioned bit 0
path: 000011010, questioned bit 11
path: 000011011, questioned bit 11
path: 001110000, questioned bit 1
path: 001110001, questioned bit 1
path: 100100100, questioned bit 0
```

```
path: 100100101, questioned bit 0
path: 110000010, questioned bit 12
path: 110000011, questioned bit 0
path: 110001100, questioned bit 13
path: 110001101, questioned bit 13
path: 110011100, questioned bit 1
path: 110011101, questioned bit 1
path: 0000110100, questioned bit 12
path: 0000110101, questioned bit 2
path: 1100000100, questioned bit 13
path: 1100000101, questioned bit 0
path: 00001101000, questioned bit 13
path: 00001101001, questioned bit 13
path: 11000001000, questioned bit 0
path: 11000001001, questioned bit 0
path: 000011010010, questioned bit 14
path: 000011010011, questioned bit 2
Calculate perplexity on testing data set...
perplexity is: 8.17174
```

3.1 Draw Bit-encode tree

Based on the output shown in Section 3, we draw the Bit-encode tree in Figure 2. Only the first five levels are drawn. The following levels can be drew similarly by the path information. The number in node circle is the questioned bit at current Tree node.

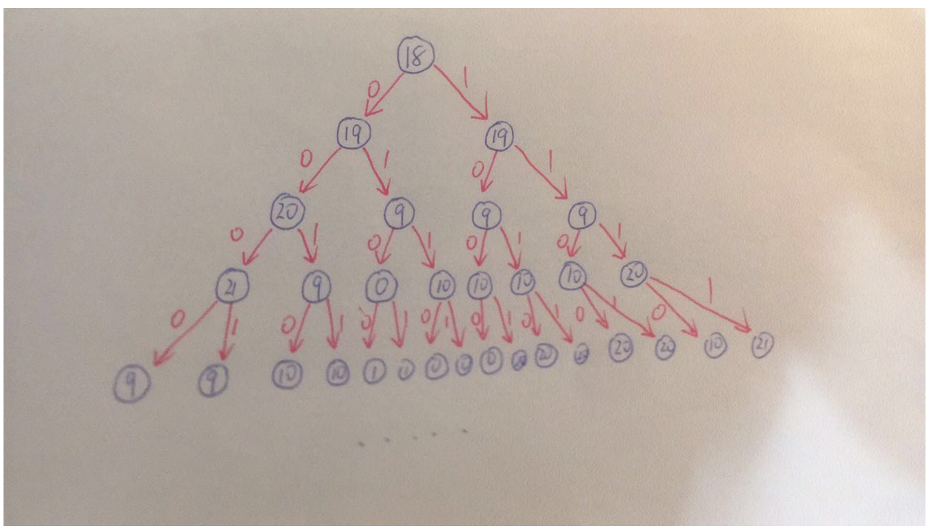

Figure 2: The first five levels Bit-encoded Tree

Based on the tree shown in 2, we can observe that b_{11} is often asked after some $b_{3,j}$ has been asked. Therefore, the immediate preceding letter l_3 is more informative than the coarsest information about the letter l_1.

3.2 Compute Perplexity

The perplexity on test dataset is computed as well, which is shown in the output in Section 3. The perplexity is 8.17174 with default threshold 0.005.

4 Chou's decision tree

To complete the required experiment, run the below command in terminal

```
./test_tree -m 3
```

The implementation is a little slow, typically requires several minutes to one hour to complete. The threshold for accepting candidate left and right nodes is still 0.005. Repeat the experiments four times, we obtain the below outputs.

```
Constructing agglomerative clustering tree...
Initialize Chou's decision tree...
Build development, and held-out data sets...
Chou's decision tree is growing...(A little slow...)
Process: ................
No frontier node...
Bit tree growing complete...
Calculate perplexity on testing data set...
perplexity is: 4.93075
```

```
Constructing agglomerative clustering tree...
Initialize Chou's decision tree...
Build development, and held-out data sets...
Chou's decision tree is growing...(A little slow...)
Process: .
No frontier node...
Bit tree growing complete...
Calculate perplexity on testing data set...
perplexity: 17.0487
perplexity is: 17.0487
```

```
Constructing agglomerative clustering tree...
Initialize Chou's decision tree...
Build development, and held-out data sets...
Chou's decision tree is growing...(A little slow, allow several minutes to one hour to complete...)
Process: .......
No frontier node...
Bit tree growing complete...
Calculate perplexity on testing data set...
perplexity is: 9.01359
```

```
Constructing agglomerative clustering tree...
Initialize Chou's decision tree...
Build development, and held-out data sets...
Chou's decision tree is growing...(A little slow, allow several minutes to one hour to complete...)
Process: ..........
No frontier node...
Bit tree growing complete...
Calculate perplexity on testing data set...
perplexity is: 15.5708
```

4.1 Tie breaking procedure

For the histories that neither $l_i \notin \mathcal{A}_i$ nor $l_i \notin \bar{\mathcal{A}}_i$, I simply randomly assign them into $\mathcal{A}_i$ or $\bar{\mathcal{A}}_i$ without no preference, that is 50 % into $\mathcal{A}_i$ or $\bar{\mathcal{A}}_i$. Such tie breaking procedure also works on test data set as well.

4.2 Compute Perplexity

I repeat the experiments 4 times. As shown in the output in Section 4, we can see the perplexity are 4.93075, 17.0487, 9.01359, 15.5708. We can see that Chou's Algorithm highly depends on its initialization. Sometimes, Chou's algorithm can outperform Bit-encoding tree, sometimes, it performs worse or competitively with Bit-encoding tree.