NLP Programming Tutorial 4 - Word Segmentation

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Introduction
What is Word Segmentation

• Sentences in Japanese or Chinese are written without spaces.

• **Word segmentation** adds spaces between words.

• For Japanese, there are tools like MeCab, KyTea.
Tools Required: Substring

- In order to do word segmentation, we need to find substrings of a word

```python
my_str = "hello world"

# Print the first 5 letters
print my_str[:5]
# Print all letters from position 6
print my_str[6:]
# Print all letters from position 3 until 7
print my_str[3:8]
```

```
$ ./my-program.py
hello
world
lo wo
```
Handling Unicode Characters with Substr

- The “unicode()” and “encode()” functions handle UTF-8

```python
input_file = open("test_file.txt", "r")
for my_str in input_file:
    my_utf_str = unicode(my_str, "utf-8")

    # Handle the string as a byte string
    print "str: %s %s" % (my_str[0:2], my_str[2:4])

    # Handle the string as a unicode string
    print "utf_str: %s %s" % (my_utf_str[0:2].encode("utf-8"),
                              my_utf_str[2:4].encode("utf-8"))
```

$ cat test_file.txt
単語分割
$ ./my-program.py
str: ?? ??
utf_str: 単語 分割
Word Segmentation is Hard!

- Many analyses for each sentence, only one correct

農産物価格安定法

駄産 物 価格 安定 法

(agricultural product price stabilization law) (agricultural cost of living discount measurement)

- How do we choose the correct analysis?
One Solution: Use a Language Model!

- Choose the analysis with the highest probability

\[
P(\text{農産 物 価格 安定 法}) = 4.12 \times 10^{-23}
\]
\[
P(\text{農産 物価 格安 定法}) = 3.53 \times 10^{-24}
\]
\[
P(\text{農産 物 価 格安 定法}) = 6.53 \times 10^{-25}
\]
\[
P(\text{農産 物 価格 安 定法}) = 6.53 \times 10^{-27}
\]

- Here, we will use a unigram language model
Problem: HUGE Number of Possibilities

• How do we find the best answer efficiently?
This Man Has an Answer!

Andrew Viterbi
(Professor UCLA → Founder of Qualcomm)
Viterbi Algorithm
The Viterbi Algorithm

- Efficient way to find the shortest path through a graph
Graph?! What?!

(Let Me Explain!)
Word Segmentations as Graphs

0 → 1 → 2 → 3

農 産 物
Word Segmentations as Graphs

農産 1.4 物

0 → 1.4 → 2.3 → 3

Each edge is a word
Word Segmentations as Graphs

- Each edge is a word
- Each edge weight is a negative log probability
  - \(- \log(P( \text{農産})) = 1.4\)
- Why?! (hint, we want the shortest path)
Word Segmentations as Graphs

Each path is a segmentation for the sentence
Word Segmentations as Graphs

Each path is a segmentation for the sentence

Each path weight is a sentence unigram negative log probability

\[- \log(P( \text{農産})) + - \log(P( \text{物})) = 1.4 + 2.3 = 3.7\]
Ok Viterbi, Tell Me More!

- The Viterbi Algorithm has two steps
  - In **forward** order, find the score of the best path to each node
  - In **backward** order, create the best path
Forward Step
Forward Step

best_score[0] = 0
for each node in the graph (ascending order)
    best_score[node] = \infty
for each incoming edge of node
    score = best_score[edge.prev_node] + edge.score
    if score < best_score[node]
        best_score[node] = score
        best_edge[node] = edge
Example:

Initialize:
best_score[0] = 0
Example:

Initialize:

best_score[0] = 0

Check $e_1$:

score = 0 + 2.5 = 2.5 (< $\infty$)
best_score[1] = 2.5
best_edge[1] = $e_1$
Example:

Initialize:
\[ \text{best_score}[0] = 0 \]

Check \( e_1 \):
\[
\begin{align*}
\text{score} &= 0 + 2.5 = 2.5 (< \infty) \\
\text{best_score}[1] &= 2.5 \\
\text{best_edge}[1] &= e_1
\end{align*}
\]

Check \( e_2 \):
\[
\begin{align*}
\text{score} &= 0 + 1.4 = 1.4 (< \infty) \\
\text{best_score}[2] &= 1.4 \\
\text{best_edge}[2] &= e_2
\end{align*}
\]
Example:

Initialize:
best_score[0] = 0

Check $e_1$:
score = $0 + 2.5 = 2.5$ ($< \infty$)
best_score[1] = 2.5
best_edge[1] = $e_1$

Check $e_2$:
score = $0 + 1.4 = 1.4$ ($< \infty$)
best_score[2] = 1.4
best_edge[2] = $e_2$

Check $e_3$:

score = $2.5 + 4.0 = 6.5$ ($> 1.4$)
No change!
Example:

Initialize:
best_score[0] = 0

Check $e_1$:
\[
\text{score} = 0 + 2.5 = 2.5 (< \infty) \\
\text{best_score}[1] = 2.5 \\
\text{best_edge}[1] = e_1
\]

Check $e_2$:
\[
\text{score} = 0 + 1.4 = 1.4 (< \infty) \\
\text{best_score}[2] = 1.4 \\
\text{best_edge}[2] = e_2
\]

Check $e_3$:
\[
\text{score} = 2.5 + 4.0 = 6.5 (> 1.4) \\
\text{No change!}
\]

Check $e_4$:
\[
\text{score} = 2.5 + 2.1 = 4.6 (< \infty) \\
\text{best_score}[3] = 4.6 \\
\text{best_edge}[3] = e_4
\]
Example:

Initialize:
best_score[0] = 0

Check e_1:
  score = 0 + 2.5 = 2.5 (< ∞)
  best_score[1] = 2.5
  best_edge[1] = e_1

Check e_2:
  score = 0 + 1.4 = 1.4 (< ∞)
  best_score[2] = 1.4
  best_edge[2] = e_2

Check e_3:
  score = 2.5 + 4.0 = 6.5 (> 1.4)
  No change!

Check e_4:
  score = 2.5 + 2.1 = 4.6 (< ∞)
  best_score[3] = 4.6
  best_edge[3] = e_4

Check e_5:
  score = 1.4 + 2.3 = 3.7 (< 4.6)
  best_score[3] = 3.7
  best_edge[3] = e_5
Result of Forward Step

$$best\_score = ( 0.0, 2.5, 1.4, 3.7 )$$

$$best\_edge = ( NULL, e_1, e_2, e_5 )$$
Backward Step
**Backward Step**

```
best_path = []
next_edge = best_edge[next_edge.length - 1]
while next_edge != NULL
    add next_edge to best_path
    next_edge = best_edge[next_edge.prev_node]
reverse best_path
```
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e_5
Example of Backward Step

**Initialize:**

```python
best_path = []
next_edge = best_edge[3] = e_5
```

**Process e_5:**

```python
best_path = [e_5]
next_edge = best_edge[2] = e_2
```
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e_5

Process e_2:
best_path = [e_5, e_2]
next_edge = best_edge[0] = NULL

Process e_5:
best_path = [e_5]
next_edge = best_edge[2] = e_2
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e₅

Process e₅:
best_path = [e₅, e₂]
next_edge = best_edge[0] = NULL

Reverse:
best_path = [e₂, e₅]
Tools Required: Reverse

- We must reverse the order of the edges

```python
my_list = [1, 2, 3, 4, 5]
my_list.reverse()
print(my_list)
```

```
$ ./my-program.py
[5, 4, 3, 2, 1]
```
Word Segmentation with the Viterbi Algorithm
Forward Step for Unigram Word Segmentation

農 産 物

\[
\begin{align*}
0.0 + -\log(P(\text{農})) \\
0.0 + -\log(P(\text{農産})) \\
\text{best(1)} + -\log(P(\text{産})) \\
0.0 + -\log(P(\text{農産物})) \\
\text{best(1)} + -\log(P(\text{産物})) \\
\text{best(2)} + -\log(P(\text{物}))
\end{align*}
\]
Note: Unknown Word Model

- Remember our probabilities from the unigram model

\[ P(w_i) = \lambda_1 P_{ML}(w_i) + (1 - \lambda_1) \frac{1}{N} \]

- Model gives equal probability to all unknown words

\[ P_{\text{unk}}("\text{proof}\") = \frac{1}{N} \]
\[ P_{\text{unk}}("\text{校正 (こうせい、英: proof)}\") = \frac{1}{N} \]

- This is bad for word segmentation

- **Solutions:**
  - Make better unknown word model (hard but better)
  - Only allow unknown words of length 1 (easy)
Word Segmentation Algorithm (1)

**load** a map of *unigram* probabilities  
# From exercise 1, unigram LM

**for each** line in the **input**
  # Forward step
  **remove** newline and **convert** line with “unicode()”
  **best_edge[0]** = **NULL**
  **best_score[0]** = 0

**for each** word_end in [1, 2, …, length(line)]
  **best_score[word_end]** = 10^{10}  
  # Set to a very large value

**for each** word_begin in [0, 1, …, word_end − 1]
  word = line[word_begin:word_end]  
  # Get the substring
  if word is in unigram or length(word) = 1  
  # Only known words
  prob = P_{uni}(word)  
  # Same as exercise 1

  my_score = best_score[word_begin] + -log( prob )
  if my_score < best_score[word_end]
    best_score[word_end] = my_score
    best_edge[word_end] = (word_begin, word_end)
Word Segmentation Algorithm (2)

```python
# Backward step
words = []
next_edge = best_edge[length(best_edge) - 1]
while next_edge != NULL:
    # Add the substring for this edge to the words
    word = line[next_edge[0]:next_edge[1]]
    encode word with the “encode()” function
    append word to words
    next_edge = best_edge[next_edge[0]]
words.reverse()
join words into a string and print
```
Exercise
Exercise

• **Write** a word segmentation program

• **Test** the program
  
  • Model: `test/04-unigram.txt`
  
  • Input: `test/04-input.txt`
  
  • Answer: `test/04-answer.txt`

• **Train** a unigram model on `data/wiki-ja-train.word` and **run** the program on `data/wiki-ja-test.txt`

• **Measure** the accuracy of your segmentation with `script/gradews.pl data/wiki-ja-test.word my_answer.word`

• **Report** the column F-meas
Challenges

- Use data/big-ws-model.txt and measure the accuracy
- Improve the unknown word model
- Use a bigram model
Thank You!