HW1_1_Dictionary_based_Tokenization_to_Student_2024

January 9, 2025

1 HW1.1: Dictionary-based Tokenization

In this exercise, you are to implement a dictionary-based word segmentation algorithm. There are two Python functions that you need to complete: * maximal_matching * backtrack

Also, you have to find how to use word_tokenize() in PythaiNLP along with customer_dict by yourselves.

[1]:
```
# !pip install pythainlp
# !pip install marisa_trie
from pythainlp.tokenize import word_tokenize
from pythainlp.corpus import get_corpus
from marisa_trie import Trie
```

1.1 Part 1) Maximal Matching from PythaiNLP

1.1.1 Create a toy dictionary to test the algorithm

This is based on the example shown in the lecture. You will tokenize the following text string: “   !” The toy dictionary provided in this exercise includes all the charaters, syllables, and words that appear that the text string.

[2]:
```
thai_vocab = [" "," "," "," "," "," "," "," ",""," ","  ","   ","  "," ","   ","!"]
input_text = "     !"
```

1.1.2 Example Dictionary

Write the `word_tokenize` function of PyThaiNLP with a custom dictionary above and using: 1. Longest matching algorithm `longest` 2. Maximal-matching algorithm `newmm`

Study `word_tokenize()` from PythaiNLP in the link below. Note: `custom_dict` will accept Trie structures as `Trie(iterable)`.

https://pythainlp.org/docs/5.0/api/tokenize.html#pythainlp.tokenize.word_tokenize

[3]:
```
####FILL CODE HERE####
# Tokenize using 'longest' algorithm
tokens_longest = word_tokenize(input_text, custom_dict=Trie(thai_vocab),␣
 ↪engine='longest')
print("Longest:", tokens_longest)
```

```
# Tokenize using 'newmm' algorithm
tokens_newmm = word_tokenize(input_text, custom_dict=Trie(thai_vocab),␣
 ↪engine='newmm')
print("Maximal-matching:", tokens_newmm)
######################
```

```
Longest: [' ', '  ', ' ', ' ', '!']
Maximal-matching: [' ', ' ', '  ', '!']
```

1.2 Part 2) Maximal Matching from Scratch

1.2.1 Maximal matching

Complete the maximal matching function below with dynamic programming to tokenize the input text and output the 2D numerical array shown in class.

[4]:
```
# from math import inf #infinity
import numpy as np

def maximal_matching(c, dictionary=Trie(thai_vocab)):
    #Initialize an empty 2D list
    n = len(c)
    # d = [[None]*n for _ in range(n)]
    d = np.full((n, n), None)
    ####FILL CODE HERE####
    for i in range(n):
        for j in range(i, n):
            if i==0 and c[0:j+1] in dictionary:
                d[i][j] = 1
            elif c[i:j+1] in dictionary:
                d[i][j] = 1 + np.min(d[:i,i-1])
            else:
                d[i][j] = np.inf
    ######################

    return d
```

1.2.2 Test your maximal matching algorithm on a toy dictionary

Expected output:

[1, 1, inf, inf, inf, inf, inf, inf, inf, inf] [None, 2, inf, inf, inf, inf, inf, inf, inf, inf] [None, None, 2, 2, 2, inf, inf, inf, inf, inf] [None, None, None, 3, inf, inf, inf, inf, inf, inf] [None, None, None, None, 3, inf, inf, inf, 3, inf] [None, None, None, None, None, 3, 3, inf, inf, inf] [None, None, None, None, None, None, 4, inf, inf, inf] [None, None, None, None, None, None, None, 4, 4, inf] [None, None, None, None, None, None, None, None, 5, inf] [None, None, None, None, None, None, None, None, None, 4] !

[5]:

```
input_text = "   !"
out = maximal_matching(input_text)
for i in range(len(out)):
    print(out[i],input_text[i])
```

```
[1 1 inf inf inf inf inf inf inf inf]
[None 2 inf inf inf inf inf inf inf inf]
[None None 2 2 2 inf inf inf inf inf]
[None None None 3 inf inf inf inf inf inf]
[None None None None 3 inf inf inf 3 inf]
[None None None None None 3 3 inf inf inf]
[None None None None None None 4 inf inf inf]
[None None None None None None None 4 4 inf]
[None None None None None None None None 5 inf]
[None None None None None None None None None 4] !
```

1.2.3 Backtracking

Complete the backtracking function below to find the tokenzied words. It should return a list containing a pair of the beginning position and the ending position of each word. In this example, it should return: [(0, 1),(2, 3),(4, 8),(9, 9)] #### Each pair contains the position of each word as follows: (0, 1) (2, 3) (4, 8)   (9, 9) !

[6]:

```
def backtrack(d):
    eow = len(d)-1 # End of Word position
    word_pos = [] # Word position # list of pairs

    ####FILL CODE HERE####
    d = np.where(d==None, np.inf, d) # replace None with INF
    while eow >= 0:
        bow = np.argmin(d[:,eow], axis=0) # bow = beginning of word
        word_pos.append((bow, eow))
        eow = bow - 1
    ######################
    word_pos.reverse()
    return word_pos

backtrack(out)
```

[6]:

```
[(0, 1), (2, 3), (4, 8), (9, 9)]
```

1.2.4 Test your backtracking algorithm on a toy dictionary

Compare your results with the result from PyThaiNLP `newmm`.

Expected output: | | |!

[7]:

```
def print_tokenized_text(d, input_text):
    tokenized_text=[]
    for pos in backtrack(d):
        #print(pos)
        tokenized_text.append(input_text[pos[0]:pos[1]+1])
    print(len(tokenized_text))
    print("|".join(tokenized_text))

print_tokenized_text(out,input_text)
```

```
4
 | |  |!
```

1.2.5 Question 1

Using your maximal matching code with the toy dictionary, how many "words" did you get when tokenizing this input text.

Answer this question in question #1 in MyCourseVille. Also print out the answer in this notebook as well.

[8]:
```
input_text = "         !"

print_tokenized_text(maximal_matching(input_text),input_text)
```

```
10
 | | | | |  | | |  |!
```

1.3 Part 3) Your Maximal Matching with Real Dictionary

For UNIX-based OS users, the following cell will download a dictionary (it's just a list of thai words). Alternatively, you can download it from this link: https://raw.githubusercontent.com/PyThaiNLP/pythainlp/dev/pythainlp/corpus/words_th.txt

[9]:
```
# !wget https://raw.githubusercontent.com/PyThaiNLP/pythainlp/dev/pythainlp/
 ↪corpus/words_th.txt
# !curl -O https://raw.githubusercontent.com/PyThaiNLP/pythainlp/dev/pythainlp/
 ↪corpus/words_th.txt
```

[10]:
```
with open("words_th.txt",encoding='utf-8-sig') as f:
    thai_vocab = f.read().splitlines()
print("Vocab size:", len(thai_vocab))
print(thai_vocab[:10])

thai_vocab.extend([" ","!"])
```

```
Vocab size: 62077
['       ', ' .', ' . .', ' . .', ' . . .', ' . .', ' . . .', ' . .',
' . .', ' . ']
```

1.3.1 Part 3.1) The output of YOUR maximal matching algoithm on the new dictionary

Expected output: [inf, 1, inf, 1, inf, inf, inf, inf, inf] [None, inf, inf, inf, inf, inf, inf, inf, inf] [None, None, inf, 2, 2, inf, inf, inf, inf] [None, None, None, inf, inf, inf, inf, inf, inf] [None, None, None, None, inf, inf, inf, inf, 2] [None, None, None, None, None, inf, 3, inf, inf] [None, None, None, None, None, None, inf, inf, inf] [None, None, None, None, None, None, None, inf, 4] [None, None, None, None, None, None, None, None, inf]

1.3.2 Expected tokenized text

|

Question: Why are the resulting tokens different?

[11]:

```
input_text = "     "
out = maximal_matching(input_text, dictionary=Trie(thai_vocab))
for i in range(len(out)):
    print(out[i],input_text[i])

print_tokenized_text(out,input_text)
```

```
[inf 1 inf 1 inf inf inf inf inf]
[None inf inf inf inf inf inf inf inf]
[None None inf 2 2 inf inf inf inf]
[None None None inf inf inf inf inf inf]
[None None None None inf inf inf inf 2]
[None None None None None inf 3 inf inf]
[None None None None None None inf inf inf]
[None None None None None None None inf 4]
[None None None None None None None None inf]
2
  |
```

Question: Why are the resulting tokens different?
dict real dict ‘ ’ dict

1.3.3 Question 2

Using your maximal matching algorithm and the actual Thai dictionary, how many "words" did you get when tokenizing this input text.

Answer this question in question #2 in MyCourseVille. Also print out the answer in this notebook as well.

[12]:

```
input_text = "                                                "

out = maximal_matching(input_text, dictionary=Trie(thai_vocab))
print_tokenized_text(out,input_text)
```

```
26
    | |  |    | |  |  | |   | |  |  | | |  |
|   |  | |   | |  |  |  | |
```

1.3.4 Part 3.2) Use PyThaiNLP `word_tokenize` with custom dictionary

Try tokenizing the following text with `word_tokenize` in `newmm` algorithm and default real dictionary.

```
[13]: text='                '

      ####FILL CODE HERE####
      word_tokenize(text, custom_dict=Trie(thai_vocab), engine='newmm')
      ######################
```

```
[13]: ['  ', '   ', '  ', '  ', '', '  ', '    ', '  ', '   ']
```

Add ‘   ’ into dictionary and then tokenize again

```
[14]: ####FILL CODE HERE####
      thai_vocab.append("       ")
      word_tokenize(text, custom_dict=Trie(thai_vocab), engine='newmm')
      ######################
```

```
[14]: ['  ', '   ', '  ', '  ', '', '  ', '        ']
```

1.3.5 Question 3

Using the code from part three only, how many “words” did you get when tokenizing this input text **after adding the new vocabs**.

Answer this question in question #3 in MyCourseVille. Also print out the answer in this notebook as well.

```
[15]: new_vocab = ["      ", "   ", "   "]
      input_text =␣
        ↪"

      thai_vocab.extend(new_vocab)
      out = maximal_matching(input_text, dictionary=Trie(thai_vocab))
      print_tokenized_text(out,input_text)
```

```
51
 | | | | |  |    |   |  |  |   |    |  | |
  |  | |     |  |   |   |    |  | | |
|    | |    |   | | |  |  |   |   |  |   |
 |    | |  |  | |   |  |   |  |  |  |
```

1.4 Part 4) Use maximal matching on real dataset

To complete this exercise, we will use the maximal matching algorithm on NECTEC’s BEST corpus.

The corpus has a structure of characters with target whether it's a beginning of a word (True/False).

[16]:
```
#Download dataset
# !gdown "1EcrlXYUyIEM3aeIJse6nPpiv_UjSKgoU&confirm=t"
```

[17]:
```
# !tar xvf corpora.tar.gz
```

[18]:
```
import pandas as pd
import os
```

[19]:
```
# Path to the preprocessed data
best_processed_path = 'corpora/BEST'
option = "test"

df = []
# article types in BEST corpus
article_types = ['article', 'encyclopedia', 'news', 'novel']
for article_type in article_types:
    df.append(pd.read_csv(os.path.join(best_processed_path, option,␣
  ↪'df_best_{}_{}.csv'.format(article_type, option))))
df = pd.concat(df)
df
```

[19]:
```
         char  target
0              True
1             False
2            False
3             False
4            False
…      …      …
644911        False
644912        False
644913         True
644914        False
644915    "    True

[2271932 rows x 2 columns]
```

[20]:
```
len(df)
```

[20]:
```
2271932
```

[21]:
```
# Some text in this corpus
all_text = "".join(df['char'].tolist())
all_text[:1000]
```

[21]:
```
'       :                    The Reformation of
Eucation from A Thai Perspective
```

```
   "       "


(                       )


                         (
                  )


                       "    - -
    "
  .
(   )   '
```

1.4.1 Question 4

Using PyThaiNLP `newmm`, how many words did you get in the BEST corpus (test)? [Runtime is around 7 mins] What are the accuracy, f1, precision, recall scores for each character?

Answer this question in question #4 in MyCourseVille. Also print out the answer in this notebook as well.

Question: What main metric should we look at? Why?

[22]:

```python
####FILL CODE HERE####
tokens = word_tokenize(text=all_text, engine='newmm')

# save
import pickle
with open('tokens.pkl', 'wb') as f:
    pickle.dump(tokens, f)
######################
```

[26]:

```python
import pickle
# load
with open('tokens.pkl', 'rb') as f:
    tokens = pickle.load(f)
```

[29]:

```python
len(tokens)
```

[29]:

```
569631
```

[23]:

```python
def convert_to_character(_tokens):
  char_list = [0]*len("".join(_tokens))
  char_count = 0
  for word in _tokens:
    char_list[char_count] = 1
    char_count += len(word)
    return char_list

chars = convert_to_character(tokens)
chars[:20]
```

[23]:
```
[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0]
```

[37]:
```python
from sklearn.metrics import f1_score,precision_score,recall_score,accuracy_score

####FILL CODE HERE####
accuracy = accuracy_score(df['target'], chars)
f1 = f1_score(df['target'], chars)
precision = precision_score(df['target'], chars)
recall = recall_score(df['target'], chars)

print(f"Accuracy: {accuracy:.2f}")
print(f"F1: {f1:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
######################
```

```
Accuracy: 0.94
F1: 0.89
Precision: 0.94
Recall: 0.85
```

Question: What main metric should we look at? Why?

Answer: F1, balance precision recall Positive class accuracy
data negative class