[Artificial Intelligence Programming

[CHINESE WORD SEGMENTATION]

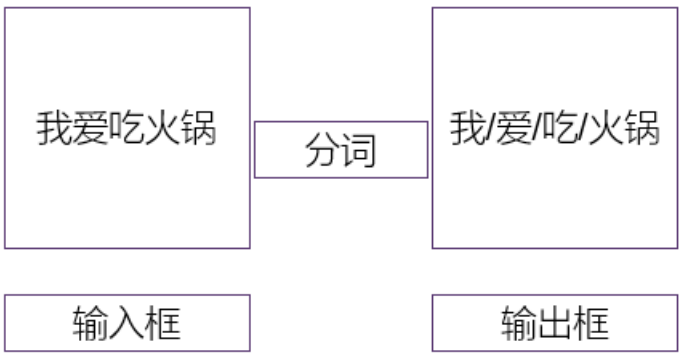
Emil Magni (ID: 2021400566)

Tsinghua University

Emil Magni – Tsinghua University
2021400566

课程大作业：根据现有的中文分词算法，实现一个中文分词系统，不允许调用现有分词工具。要求：

1. 实现一个传统的匹配算法；
2. 实现一个机器学习算法，包括但不限于深度学习；
3. 1-2 为基本要求，有能力的同学可以额外实现 2-3 个算法进行比较；
4. 禁止使用预训练模型，比如 bert；
5. 实验报告里给出所使用的算法介绍，并且对比其在测试集上的结果；
6. 做出一个 UI 界面，功能大概如下图所示，并在实验报告中附图展示；
7. 算法和 UI 均使用 python 进行编程。



数据集：http://sighan.cs.uchicago.edu/bakeoff2005/
数据集参考：https://zhuanlan.zhihu.com/p/45243642

Emil Magni – Tsinghua University
2021400566

Word Segmentation is the process of separating Chinese sentences into the correct separate words. Unlike in English we cannot rely on using spaces to separate each word. Having the correct understanding of where to split the Chinese characters for each word is the foundation for understanding the real meaning of the sentence using more advanced natural language processing.

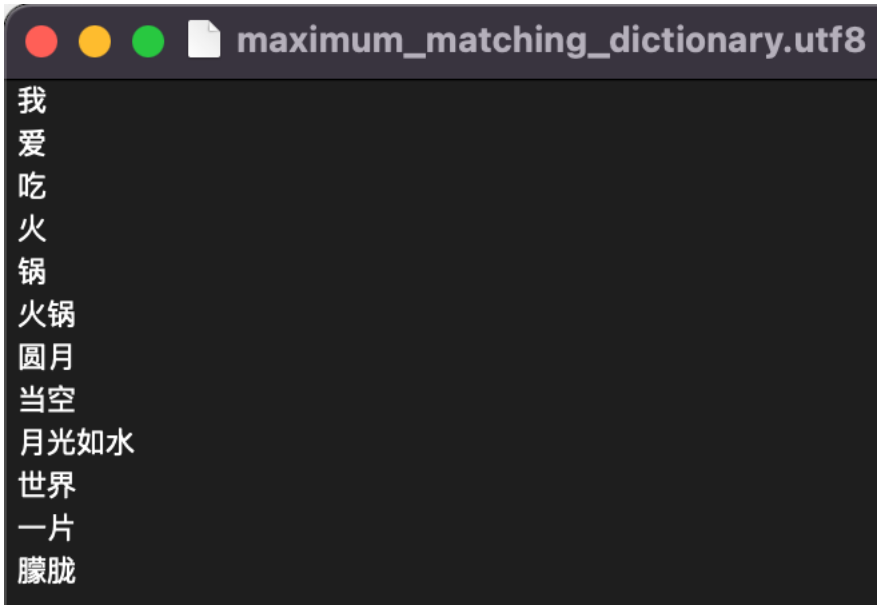
1. Implementing a traditional matching algorithm

最大匹配法（Maximum Matching method）

The forward maximum matching algorithm is one of the simplest ways to segment a Chinese sentence. The method relies on having a dictionary of known words and combined words. As the name suggests it moves forward from left to right seeing if it can find a match for the sentence in the dictionary.

If a match is found with the sentence, then it means that the given input was in fact a word and it is returned. If the match is unsuccessful, it will remove the last character from the string and see if it can find a match for this shorter string of Chinese characters. This will be repeated until all words are segmented, and each word will be returned.

To test this example of a traditional matching algorithm I created a very simple matching dictionary: maximum_matching_dictionary.utf8



When inputting the unsegmented text string as: "圆月当空月光如水世界一片朦胧"

The algorithm returns the response as shown below:

```
['圆月', '当空', '月光如水', '世界', '一片', '朦胧']
```

As shown, it is successfully completing the forward matching based on the dictionary file and returning the segmented parts of the text input given.

逆向最大匹配法（Reverse Maximum method）

This method works similarly, but the matching direction is the opposite of the forward MM method, from right to left.

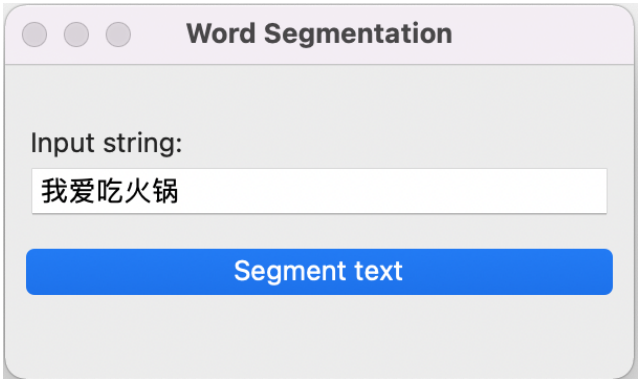
Experiments show that the reverse maximum matching method is more effective than the (positive) maximum matching method for Chinese.

Graphical User Interface for the reverse maximum matching

I used the library tkinter (“Tk interface”), a framework in Python, to create a simple graphical user interface for the reverse maximum matching algorithm.
It simply works by accepting a string as the input and when the button below is pressed it will display the resulting segmented words.
I created the GUI with a ttk.Entry widget to create a textbox that stores a string variable.

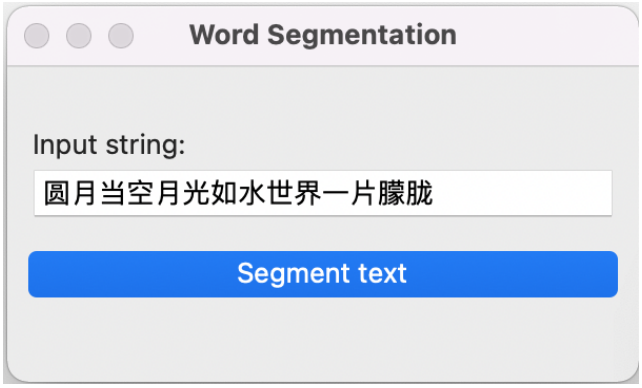
Example 1:

Input: 我爱吃火锅



Example 2:

Input: 圆月当空月光如水世界一片朦胧



2. Machine Learning algorithm based on the Hidden Markov Model

In this section I will be making a more generally useable solution that can work on more sentences with machine learning. The model will be based on the Hidden Markov chain to make the Chinese word segmentation.

A Hidden Markov Model is a statistical model, and it is generally used for sequential data. The name means that the Markov model that produces the data is hidden. What is visible is instead the observations.

I have used the provided SIGHAN msr dataset by Microsoft Research, which contains simplified Chinese corpus.

Loading Data

[1]:
```
# Import the necessary libraries
import os
import numpy as np

import re
# Use regular expression to detect Chinese characters
regex = re.compile("[^\u4e00-\u9fff]")
```

[2]:
```
# loading of data
# I store the data in a list with an element for each of the sentences.
train_data = open(os.path.join("icwb2-data/training", "msr_training.utf8"), encoding="utf-8").read().splitlines()
test_data = open(os.path.join("icwb2-data/testing", "msr_test.utf8"), encoding="utf-8").read().splitlines()
test_gold = open(os.path.join("icwb2-data/gold", "msr_test_gold.utf8"), encoding="utf-8").read().splitlines()
```

The training data contains examples of segmented Chinese words separated by spaces. The testing dataset contains sentences without the segmentation and the gold set contains the correct labeling for calculating the precision and recall.

Data Preprocessing

First the preprocessing is done to provide the labels for the training and testing data. The 4 labels will show where the character is placed according to the segmentation.

- "S" means it is a single character token standing alone.
- "E" means it is the ending character of a word.
- "M" means it is an internal character in the middle of a word.
- "B" means the character is the beginning of a new word.

Emil Magni – Tsinghua University
2021400566

Assess

Then I get the start, transition, and emission probabilities

Start probabilities

```
# Creating a dictionary for the state count
state_count = {}
states = ["B", "M", "E", "S"]
for state in states:
    state_count[state] = 0

for i in range(len(train_hs)):
    length = len(train_hs[i])
    if length > 0:
        for j in range(length - 1):
            # Update the state count
            state_count[train_hs[i][j]] += 1
        state_count[train_hs[i][length - 1]] += 1

total_states = sum(state_count.values())  # Getting the total number of states in the sample
start_prob = {}

for state in states:
    # Normalize the state count with the total number of states
    start_prob[state] = state_count[state] / total_states

print(start_prob)
```

```
{'B': 0.34577982777611344, 'M': 0.09436133454366076, 'E': 0.34577982777611344, 'S': 0.21407900990411233}
```

Transition probabilities

```
# Initializing the transition probabilities
trans_prob = {}
for state in states:
    trans_prob[state] = {}
    for state_i in states:
        trans_prob[state][state_i] = 0

for i in range(len(train_hs)):
    length = len(train_hs[i])
    if length > 0:
        for j in range(length - 1):
            # Update the transition probabilities
            s_from = train_hs[i][j]
            s_to = train_hs[i][j+1]
            trans_prob[s_from][s_to] += 1

for i in states:
    for j in states:
        # Normalize the frequency of the transition with the state counts
        trans_prob[i][j] /= float(state_count[i])

print(trans_prob)
```

```
{'B': {'B': 0.0, 'M': 0.14804196105988887, 'E': 0.8519580389401111, 'S': 0.0}, 'M': {'B': 0.0, 'M': 0.4575116593413479, 'E': 0.542488340658652, 'S': 0.0}, 'E': {'B': 0.5755412261794932, 'M': 0.0, 'E': 0.0, 'S': 0.36684089371227374}, 'S': {'B': 0.606061801705827, 'M': 0.0, 'E': 0.0, 'S': 0.3727001913630176 3}}
```

Emission probabilities

```
# Initialize the emission probabilities
emission_prob = {}

# Get all the vocabulary in the corpus (train and test sets)
vocab = list(set(train_wc.keys()) | set(test_wc.keys()))

# Initialize the emission probabilities
for state in states:
    emission_prob[state] = {}
    for word in vocab:
        emission_prob[state][word] = 1

for i in range(len(train_hs)):
    length = len(train_hs[i])
    for j in range(length):
        # Update the emission probabilities
        obs = train_data[i][j]
        hidden = train_hs[i][j]
        emission_prob[hidden][obs] += 1

for state in states:
    for word in vocab:
        if emission_prob[state][word] == 0:
            continue
        else:
            # Normalize the emission probabilities
            emission_prob[state][word] /= float(state_count[state])
```

Decode – Viterbi Algorithm

The Viterbi Algorithm is used with to label the states in the testing data. The Viterbi algorithm gets the maximum posteriori probability of the most likely sequence of the hidden states in each path. After going through each step, the Viterbi algorithm then goes back to get the most probable state at each step. In other words, the Viterbi Algorithm is used to answer the question of what sequence of states that best explains a sequence of observations. The Viterbi algorithm is looking at each word in the sentences and establishing the most likely state at any given time.

RESULTS

I calculate the precision and recall for each label and then the F1 score.

```
F1 score for state B: 0.6738491525681829
F1 score for state M: 0.24104662322574127
F1 score for state E: 0.6812865953626356
F1 score for state S: 0.44521343198634034
Macro-F1 score: 0.5103489507857251
```

I can see that the middle characters M is where most of the classifications go wrong.

When looking at the earlier transition probabilities it shows that M is also here lower than the other states, which shows that this state is seen less often than the others, which may have an impact on why it is very often classified wrong.

```
{'B': {'B': 0.0, 'M': 0.14804196105988887, 'E': 0.8519580389401111, 'S': 0.0}, 'M': {'B': 0.0, 'M': 0.4575116593413479, 'E': 0.542488340658652, 'S': 0.0}, 'E': {'B': 0.5755412261794932, 'M': 0.0, 'E': 0.0, 'S': 0.3668408937122737 4}, 'S': {'B': 0.6060 61801705827, 'M': 0.0, 'E': 0.0, 'S': 0.3727001913630176 3}}
```

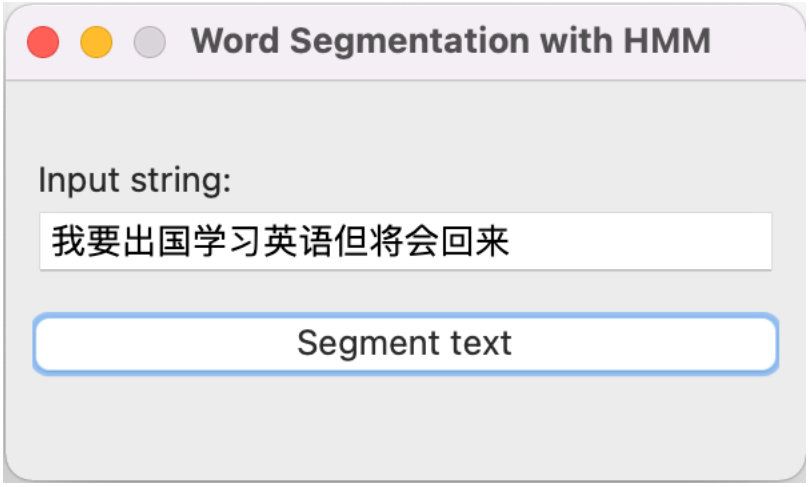
Finally, I apply the word breaking to all the testing data and then save the results in the file my_prediction.txt

```
# I apply the word breaking to the testing data
def word_segmentation(test_data, pred):
    segmented = ""
    i = 0 # Counter for the test data index
    j = 0 # Counter for the prediction index
    while i < len(test_data):
        segmented += test_data[i]
        # Check for Chinese character
        if test_data[i] > u"\u4e00" and test_data[i] < u"\u9fff":
            # Add space after the character if label
            # is either "E" or "S"
            if pred[j] in ["E", "S"]:
                segmented += " "
            j += 1
        i += 1
    return segmented

segmented = []
for i in range(len(test_data)):
    # Converting BMES label to word segmentation
    segmented.append(word_segmentation(test_data[i], test_pred[i]))
```

```
# I save the file with segmentation as my_prediction.txt
with open("my_prediction.txt", "w", encoding="utf-8") as fout:
    for item in segmented:
        fout.write("%s\n" % item)
```

GUI Example of segmentation based on HMM



As seen below the segmentation is still not very precise, as the correct segmentation would be: 我　要　出国　学习　英语　，　但　将　会　回来