View all sources on GitHub

Chinese Segmentation Based on Maximum Probability

References:

- Origin corpus from internet: https://github.com/hankcs/OpenCorpus
- Bigram: https://en.wikipedia.org/wiki/Bigram
- Viterbi Algorithm
- n-gram
- max probability chinese segmentation
- MP-segmentation
- NLP Viterbi 算法应用之Unigrams、Bigrams
- 动态规划(DP)的整理-Python描述

Prepare Data

According to slides presented by Professor Wang. It is convenient to use Beijing University's corpus which can be found here.

This project is going to do some experiments on this corpus. So let's get start!

Download corpus from Internet

We could download `1998 people news in January` corpus into your colab via `wget` command.

Note that all the files your download from internet to virtual machine will be deleted when over 14 hours!

```
! wget https://github.com/hankcs/OpenCorpus/raw/master/pku98/199801.txt
```

```
--2018-12-13 03:27:00--  https://github.com/hankcs/OpenCorpus/raw/master/pku9
Resolving github.com (github.com)... 192.30.253.112, 192.30.253.113
```

Functions for data cleaning

```
--2018-12-13 03:27:06--  https://raw.githubusercontent.com/hankcs/OpenCorpus/
```

```python
import codecs
import chardet

def read_lines(corpus_name, encode_type='utf-8'):
  """Read lines from different corpus

  # Arguments:
    corpus_name: str in corpus list

  # Return:
    lines: a list contains the whole corpus
  """

  with codecs.open(corpus_name, 'r', encoding=encode_type) as file:
    lines = file.readlines()

  return lines


# test case for loading corpus
f = open('199801.txt', 'rb')
data = f.read()
print("the format of corpus is: ", chardet.detect(data))
lines = read_lines("199801.txt")
print("Examples of corpus 1998 People News:")
print("First sentence of corpus:", lines[0])
print("Second sentence of corpus:", lines[1])
```

```
the format of corpus is:  {'encoding': 'utf-8', 'confidence': 0.99, 'language
Examples of corpus 1998 People News:
First sentence of corpus: 迈向/v 充满/v   希望/n   的/u    新/a    世纪/n   ——/w

Second sentence of corpus: 中共中央/nt   总书记/n 、/w    国家/n   主席/n   江泽
```

Build dictionary: word counts dictionary and bi-gram dictionary

The origin corpus need to be clean. Notice that in each sentence, words are seperated by `tab` and tagged by default.

Our goal of this function is to build a dictionary that wipe off all the tagging and the other no meaning chars.

```python
def build_dictionary(corpus):
  """Build a dictonary based on differnet corpus for finding candidate words

  # Arguments:
    corpus: a list contains all sentences in corpus

  # Return:
    word_count_dict: a dictionary whose key represents the word and value
                     represents the count of its occurrence.
  """
  word_count_dict = {}
  i = 0
  for index, sentence in enumerate(corpus):
    if index == 0:
      print("Example of sentence in corpus: ", sentence)
    # seperate word by tab
    word_list = [word for word in sentence.split("\t")]
    if index == 0:
      print("seperate word by tab: ", word_list)
      print("----------")
    # drop tagging in word
    for i, word in enumerate(word_list):
      word_list[i] = word.split('/')[0]
      if index == 0:
        print("Word examples of splited by the backslash", word_list[i])
    # build dictionary via sentence
    for word in word_list:
      if word_count_dict.get(word) != None:
        word_count_dict[word] += 1
      else:
        word_count_dict[word] = 1

  return word_count_dict

# build dictonary using people news
word_count_dict = build_dictionary(lines)
```

```
Example of sentence in corpus:  迈向/v   充满/v   希望/n   的/u    新/a    世纪/

seperate word by tab:  ['迈向/v', '充满/v', '希望/n', '的/u', '新/a', '世纪/n',
----------
Word examples of splited by the backslash 迈向
Word examples of splited by the backslash 充满
Word examples of splited by the backslash 希望
Word examples of splited by the backslash 的
Word examples of splited by the backslash 新
Word examples of splited by the backslash 世纪
Word examples of splited by the backslash ——
Word examples of splited by the backslash 一九九八年
Word examples of splited by the backslash 新年
Word examples of splited by the backslash 讲话
Word examples of splited by the backslash （
Word examples of splited by the backslash 附
Word examples of splited by the backslash 图片
Word examples of splited by the backslash １
Word examples of splited by the backslash 张
Word examples of splited by the backslash ）
```

Build bigram dictionary

For the sake of accerlating Bigram Model and reducing the time spending during computing probability. We need to build a dictionary to record Bigram word pairs.

```
def bigram_dictionary(corpus):
  """Build Bigram dictionary by corpus
  The key is a bigram, and The value is times of bigram occurance.

  # Arguments:
    corpus: a list contains all sentences in corpus

  # Returns:
    bigram_dict: a dictionary whose key represents the word and value
                 represents the count of its occurrence.

  """
  bigram_dict = {}
  for epi, sentence in enumerate(corpus):
    # seperate word by tab
```

```python
    word_list = [word for word in sentence.split("\t")]
    # drop tagging in word
    for i, word in enumerate(word_list):
      word_list[i] = word.split('/')[0]
    # build bigram count for dictionary
    for pos, word in enumerate(word_list):
      if pos < len(word_list) - 1:
        bigram = word_list[pos] + ' ' + word_list[pos + 1]
        if bigram_dict.get(bigram) != None:
          bigram_dict[bigram] += 1
        else:
          bigram_dict[bigram] = 1
    if epi == 0:
      print("Example of bigram dictionary in first sentence:\n", bigram_dict)
  return bigram_dict

bigram_dict = bigram_dictionary(lines)
print("We have ", len(bigram_dict), " bigram pairs!!")

```

```
Example of bigram dictionary in first sentence:
 {'迈向 充满': 1, '充满 希望': 1, '希望 的': 1, '的 新': 1, '新 世纪': 1, '世纪 —— '
We have  458507  bigram pairs!!
```

N-gram model

What is N-gram model?

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application.

What is Bigram?

A bigram or digram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. A bigram is an n-gram for n=2. The frequency distribution of every bigram in a string is commonly used for simple statistical analysis of text in many applications, including in computational linguistics, cryptography, speech recognition, and so on.

Gappy bigrams or skipping bigrams are word pairs which allow gaps (perhaps avoiding connecting words, or allowing some simulation of dependencies, as in a dependency grammar).

Head word bigrams are gappy bigrams with an explicit dependency relationship.

Bigrams help provide the conditional probability of a token given the preceding token, when the relation of the conditional probability is applied:

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

That is, the probability $P()$ of a token W_n given the preceding token W_{n-1} is equal to the probability of their bigram, or the co-occurrence of the two tokens $P(W_{n-1}, W_n)$, divided by the probability of the preceding token.

Build a class with Bi-gram Model

The details of Bi-gram Model have writen on the comments of each function. The code style is learning from Keras.

```python
import numpy as np
import re

class BiGram(object):
  """Apply N-Gram model to Bigram.

  A bigram or digram is a sequence of two adjacent elements from a string of
  tokens, which are typically letters, syllables, or words.
  A bigram is an n-gram for n=2. The frequency distribution of every bigram in
  a string is commonly used for simple statistical analysis of text in many
  applications, including in computational linguistics, cryptography, speech
  recognition, and so on.

  # Arguments:
    corpus:
    word_count_dict:
    bigram_dict:
    max_split: a int numrepresenting the sliding window's size
    word_num: a int num representing length of dictionary's words

  # References:
    - [Bigram](https://en.wikipedia.org/wiki/Bigram)

  """


  def __init__(self, corpus, word_count_dict, bigram_dict, max_split=4):
    self.corpus = corpus
    self.word_count_dict = word_count_dict
    self.max_split = max_split
    self.bigram_dict = bigram_dict
    self.word_num = len(word_count_dict)


  def split_sentence(self, sentence):
    """ reduce calculation complexity through split the sentence into small one
        split subsentence according to punctuation

    # Aruguments:
      sentence: a str represents the sentence to be splitted

    # Returns:
      split_list: a list of sentences containing the split results
    """
    split_list = []

    com_sens = sentence.split('，')
    for com_sen in com_sens:
      if com_sen != com_sens[-1]:
        com_sen += '，'

      stop_sens = com_sen.split('。')
      for stop_sen in stop_sens:
        if stop_sen != stop_sens[-1]:
          stop_sen += '。'

        pause_sens = stop_sen.split('、')
        for pause_sen in pause_sens:
          if pause_sen != pause_sens[-1]:
            pause_sen += '、'

          l_brack_sens = pause_sen.split('（')
          for l_brack_sen in l_brack_sens:
            if l_brack_sen != l_brack_sens[-1]:
              l_brack_sen += '（'

            r_brack_sens = l_brack_sen.split('）')
            for r_brack_sen in r_brack_sens:
              if r_brack_sen != r_brack_sens[-1]:
                r_brack_sen += '）'

              semi_sens = r_brack_sen.split('；')
              for semi_sen in semi_sens:
                if semi_sen == semi_sens[-1]:
                  split_list.append(semi_sen)
                else:
                  split_list.append(semi_sen+"; ")


    return split_list

  def candidate_words(self, sentence):
    """scan the whole sentence from left to right, we will obtain candidate
    words for determining left-neighbor words.
    And saving postions of each candidate words for the next step

    # Arguments:
      sentence: str, content of corpus

    # Returns:
      candidate_words: a list of candidate words.
    """
    candidate_words = []

    # search the whole sentence
    start_pos = 0
    while start_pos < len(sentence):
      word = sentence[start_pos]
      candidate_words.append([word, start_pos, start_pos])
      # check out the range of object sentence
      for max_pos in range(1, self.max_split):
        current_pos = start_pos + max_pos
        if current_pos < len(sentence):
          word = word + sentence[current_pos]
          # add words according to dictionary and num list!
          if word in self.word_count_dict.keys():
            candidate_words.append([word, start_pos, current_pos])

      start_pos += 1

    return candidate_words

  def probable_segments(self, sentence):
    """generate all probable segments based on candidate words
    and index of words.
    # Arguments:
      sentence: a str to be scaned to segment.

    # Returns:
      possible_segments: a list contains all possible segment results.

    # Example:
      Given sentence: 太阳当空照，江泽民对我笑
      You will obstain: ['太 阳 当空 照 ， 江 泽民 对 我 笑', '太 阳 当空 照 ，
      江 泽 民 对 我 笑', '太阳 当空 照 ， 江泽民 对 我 笑', .etc]
    """
    # get candidate words
    candidate_words = self.candidate_words(sentence)
    c = 0

    for word in candidate_words:
    # find the first candidate word of sentence
      if c > 4000:
        break
      if word[1] == 0 and word[2] != len(sentence) - 1:
        for word in candidate_words:
          if word[1] == 0 and word[2] != len(sentence) - 1:
            end = word[2]
            for later_word in candidate_words:
              if later_word[1] == end + 1:
                word_seq = [word[0] + ' ' + later_word[0], word[1], later_word[
                candidate_words.append(word_seq)
                c += 1
            candidate_words.remove(word)

    print(candidate_words)
    print(len(candidate_words))
    word_segment_res_list = []
    for seque in candidate_words:
      if seque[1] == 0:
        if seque[2] == len(sentence) - 1:
          word_segment_res_list.append(seque[0])

    return word_segment_res_list


  def log_prob(self, sequence, display=False):
    """Avoiding overflow from computing probability, it is necessary to use log
       probability instead of float probability.
       The goal is to compute the probability of a sequence of words.
       Note that we are using laplace smoothing when the bigram is not appear i
       the corpus or the word is OOV.

    # Arguments:
      sequence: a str that contains a word sequence to be computed.
      display: a boolean to determine whether to display or not

    # Returns:
      prob_total: a float number represents log probability.
    """
    word_list = sequence.split(' ')
    prob_total = 0.0
    word_start = word_list[0]
    # first word prob with laplace smooth
    if word_start in self.word_count_dict.keys():
      count = self.word_count_dict[word_start]
      d = 0
    else:
      count = 1
      d = len(sequence)

    prob_total += np.log(count / (self.word_num + d))
    # calculate Bigram Probability
    for i in range(len(word_list) - 1):
      prev_w = word_list[i]
      later_w = word_list[i+1]
      bigram = prev_w + " " + later_w
      if bigram in self.bigram_dict.keys():
        count = self.bigram_dict[bigram]
        d = 0
      else:
        count = 1
        d = len(sequence)
      prob_total += np.log(count / (self.word_num + d))

    prob = np.power(np.e, prob_total)
    if display:
      print('The probability of ', sequence, " is ", prob)

    return prob_total


  def maximum_probability(self, sequence_list, sentence_len, display=False):
    """Choose the sequence has maximum probability from all probable sequences.

    # Arguments:
      sequence_list: a sequences contains all the probable sequences.
      display: a boolean to determine whether to display or not

    # Returns:
      seg_result: a str represents the result
    """
    max_prob = -float('inf')
    seg_result = ""
    for seq in sequence_list:
      if len(seq.split(' ')) <= 10:
        prob = self.log_prob(seq)
        if max_prob < prob:
          max_prob, seg_result = prob, seq
      else:
        if len(seq.split(' ')) <= sentence_len * 0.9:
          prob = self.log_prob(seq)
          if max_prob < prob:
            max_prob, seg_result = prob, seq
    if display:
      print('The segment result whose probability is %s : %s' % (np.power(np.e,
    return seg_result

  def segment_testset(self, test_set):
    """
    """
    corpus_result = ''
    for sentence in test_set:
      print("origin sentence is:", sentence)
      tmp_result = ''
      sub_sentences = self.split_sentence(sentence)
      print("sub_sentences:", sub_sentences)
      if sub_sentences[-1] == '\r\n':
        sub_sentences = sub_sentences[:-1]
        print("regular sub:", sub_sentences)
        for sen in sub_sentences:
          prob_seqs = self.probable_segments(sen)
          max_seg = self.maximum_probability(prob_seqs, len(sen), display=True)
          # every seg need a space!
          max_seg += " "
          tmp_result += max_seg
      else:
        sen = sub_sentences[0][:-2]
        print("special sub:", sen)
        prob_seqs = self.probable_segments(sen)
        max_seg = self.maximum_probability(prob_seqs, len(sen), display=True)
        tmp_result += max_seg

      # line break after we complete one sentence.
      tmp_result += '\r\n'
      print('segment result of this sentence is: ', tmp_result)
      print("*" * 30)
      corpus_result += tmp_result

    return corpus_result

```

Test cases for BiGram model

Codes below show that results of each function. We make some pseudo data as input. Let's see what happened in the inner mechanism.

Function: split_sentence

```
bigram = BiGram(lines, word_count_dict, bigram_dict, 4)
long_str = '李岚清指出，要克服困难，群策群力，千方百计使高校教职工、特别是青年教师的住房困难
punc_str = "大，大的。大（得）的；大大、得的。"
print("origin long sentence: \n", long_str)
print("origin punctuation sentence: \n", punc_str)
long_sens = bigram.split_sentence(long_str)
punc_sens = bigram.split_sentence(punc_str)
print("-"*20)
print("splitted long sentences: ")
for each in long_sens:
  print(each)
print("-"*20)
print("splitted punctuation sentences: ")
for each in punc_sens:
  print(each)
```

```
origin long sentence:
 李岚清指出，要克服困难，群策群力，千方百计使高校教职工、特别是青年教师的住房困难问题在３年
origin punctuation sentence:
 大，大的。大（得）的；大大、得的。
--------------------
splitted long sentences:
李岚清指出，
要克服困难，
群策群力，
千方百计使高校教职工、
特别是青年教师的住房困难问题在３年内有一个较大突破。

--------------------
splitted punctuation sentences:
大，
大的。
大（
得）
的；
大大、
```

Function: candidate_words

```python
bigram = BiGram(lines, word_count_dict, bigram_dict, 4)
cand_w = bigram.candidate_words("太阳当空照，江泽民对我笑")
print("test caset with normal sentence:", cand_w)
cand_w = bigram.candidate_words("以１９９８年的２０００个新闻")
print("test case with numbers:", cand_w)
# cand_w = bigram.candidate_words("全国已有３０个省、自治区、直辖市和１４个中央行业主管
cand_w = bigram.candidate_words("图为吴艳艳（右）和陈妍（左）在终点庆贺胜利。")
print("test case with multi numbers:", cand_w)
```

```
entence: [['太', 0, 0], ['太阳', 0, 1], ['阳', 1, 1], ['当', 2, 2], ['当空', 2,
[['以', 0, 0], ['１', 1, 1], ['１９', 1, 2], ['１９９', 1, 3], ['１９９８', 1, 4]
bers: [['图', 0, 0], ['为', 1, 1], ['吴', 2, 2], ['吴艳艳', 2, 4], ['艳', 3, 3],
```

Function: probable_segments

```python
bigram = BiGram(lines, word_count_dict, bigram_dict, 4)
# test segment
print("Basic example:")
seg_result1 = bigram.probable_segments("太阳当空照，江泽民对我笑")
print(seg_result1)
print("-----")
print("Example with number:")
seg_result2 = bigram.probable_segments("以１９９８年的２０００个新闻")
print(seg_result2)
print("-----")
print("Example with special case:")
seg_result3 = bigram.probable_segments("（右）")
print(seg_result3)
# print("\n"*2 + "-"*50 + "\n"*2)
```

```
Basic example:
[['阳', 1, 1], ['当', 2, 2], ['当空', 2, 3], ['空', 3, 3], ['照', 4, 4], ['，',
26
['太 阳 当空 照 ， 江 泽民 对 我 笑', '太 阳 当空 照 ， 江 泽 民 对 我 笑', '太阳 当空
-----
Example with number:
[['１', 1, 1], ['１９', 1, 2], ['１９９', 1, 3], ['１９９８', 1, 4], ['９', 2, 2
87
['以 １ ９９８ 年 的 ２０００ 个 新闻', '以 １ ９９８ 年 的 ２０００ 个 新 闻', '以 １
```

Function: log_prob

test computation of sentence probability.

Note that in this function, the outputs displayed are tranform into float probability.

```
for each_seg in seg_result1:
  bigram.log_prob(each_seg, display=True)

print("-----")
for each_seg in seg_result2:
  bigram.log_prob(each_seg, display=True)

print("-----")
for each_seg in seg_result3:
  bigram.log_prob(each_seg, display=True)
```

```
The probability of  以 1  998 年 的 2  0  0  0 个 新闻  is  1.810009146865689
The probability of  以 1  998 年 的 20  0  0 个 新闻  is  2.6424323651338977
The probability of  以 19  9  8 年 的 2000 个 新 闻  is  1.1057563127944933
The probability of  以 1  998 年 的 2  0  0  0 个 新 闻  is  2.47954948555549
The probability of  以 1  998 年 的 20  0  0 个 新 闻  is  3.620018293731351
The probability of  以 199  8 年 的 200  0 个 新闻  is  1.614238357967577e-
The probability of  以 19  9  8 年 的 20  0  0 个 新闻  is  5.917337595522429
The probability of  以 19  9  8 年 的 2  0  0  0 个 新闻  is  4.05310973600413
The probability of  以 19  9  8 年 的 20  0  0 个 新 闻  is  8.10621947200831
The probability of  以 199  8 年 的 2000 个 新闻  is  4.712930318216237e-3
The probability of  以 1  998 年 的 200  0 个 新闻  is  1.2344175678575462e
The probability of  以 199  8 年 的 2000 个 新 闻  is  6.456953431870241e-
The probability of  以 1  998 年 的 200  0 个 新 闻  is  1.6911567136857056
The probability of  以 1998 年 的 200  0 个 新闻  is  5.542629708024691e-3
The probability of  以 1998 年 的 200  0 个 新 闻  is  7.593553599599704e-
The probability of  以 199  8 年 的 200  0 个 新 闻  is  2.211512625588991e
The probability of  以 19  9  8 年 的 2  0  0  0 个 新 闻  is  5.5522050722235
The probability of  以 1  9  98 年 的 2000 个 新闻  is  6.17208783928772e-3
The probability of  以 1998 年 的 2000 个 新闻  is  1.618198459254263e-27
The probability of  以 1  9  98 年 的 20  0  0 个 新闻  is  4.525022867164164
The probability of  以 1998 年 的 2000 个 新 闻  is  2.2170518832098843e-
The probability of  以 1998 年 的 2  0  0  0 个 新闻  is  8.127358470537498e
The probability of  以 1  99  8 年 的 2  0  0  0 个 新闻  is  5.27129951541419
```

```python
bigram.maximum_probability(seg_result1, 12, display=True)
bigram.maximum_probability(seg_result2, 16, display=True)
bigram.maximum_probability(seg_result3, 100, display=True)
# print("\n"*2 + "-"*50 + "\n"*2)
```

```
The segment result whose probability is 1.959758893130026e-33 : 太阳 当空 照 ,
The segment result whose probability is 1.618198459254263e-27 : 以 １９９８ 年
The segment result whose probability is 3.5163681335659964e-09 :  （ 右 ）
'（ 右 ）'

The probability of  以 19  98 年 的 20  0  0 个 新闻  is  2.0318396176343473
```

Measure the performance of Bi-Gram Model

the test set I have upload to github which only contains 100 lines. You can upload into Colab virtual machine from here

```
The probability of  以 1998 年 的 2  0  0  0 个 新 闻  is  1.1133939648244836
```

Download test set from Internet

Note that it is necessary to check out the formation of dataset prevent from reading messy code.

```
The probability of  以 199  8 年 的 2  0  0  0 个 新闻  is  2.366935038208936
```

```python
!wget https://github.com/824zzy/blogResources/raw/master/txtRestources/bigram_t
f = open('bigram_test.txt', 'rb')
data = f.read()
print("the format of corpus is: ", chardet.detect(data))
```

```
--2018-12-13 04:53:46--  https://github.com/824zzy/blogResources/raw/master/t
Resolving github.com (github.com)... 192.30.253.113, 192.30.253.112
Connecting to github.com (github.com)|192.30.253.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/824zzy/blogResources/master/txtRe
--2018-12-13 04:53:46--  https://raw.githubusercontent.com/824zzy/blogResourc
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.
```

Show some cases from test set

With the result above, we should pass 'gbk' to function `read_lines`

```python
# test case for loading corpus
import codecs
test_lines = read_lines('bigram_test.txt', 'gbk')

print("Examples of corpus 1998 People News:")
print("First sentence of corpus:", test_lines[0])
print("Second sentence of corpus:", test_lines[1])
print(test_lines)
```

```
Examples of corpus 1998 People News:
First sentence of corpus: 本报讯春节临近，全国各地积极开展走访慰问困难企业和特困职工的

Second sentence of corpus: 各地党委、政府及有关部门按照党中央的部署，以高度的政治责任感

['本报讯春节临近，全国各地积极开展走访慰问困难企业和特困职工的送温暖活动，并广泛动员社会各方
```

```python
bigram_test = BiGram(lines, word_count_dict, bigram_dict, 4)
result = bigram_test.segment_testset(test_lines)
```

```
The segment result whose probability is 1.5084979888400226e-14 : 甘 肃 、
[['吉林', 0, 1], ['林', 1, 1], ['、', 2, 2], ['吉 林 、', 0, 2]]
4
The segment result whose probability is 6.537384594373619e-13 : 吉 林 、
[['湖南', 0, 1], ['南', 1, 1], ['、', 2, 2], ['湖 南 、', 0, 2]]
4
The segment result whose probability is 1.2067983910720204e-13 : 湖 南 、
[['山东', 0, 1], ['东', 1, 1], ['、', 2, 2], ['山 东 、', 0, 2]]
4
The segment result whose probability is 4.2241561994414005e-12 : 山 东 、
[['福建', 0, 1], ['建', 1, 1], ['、', 2, 2], ['福 建 、', 0, 2]]
4
The segment result whose probability is 9.050987933040163e-14 : 福 建 、
[['河北', 0, 1], ['北', 1, 1], ['、', 2, 2], ['河 北 、', 0, 2]]
4
The segment result whose probability is 1.4582147225453592e-13 : 河 北 、
[['广东', 0, 1], ['东', 1, 1], ['、', 2, 2], ['广 东 、', 0, 2]]
4
The segment result whose probability is 1.7600650831005862e-12 : 广 东 、
[['湖北', 0, 1], ['北', 1, 1], ['、', 2, 2], ['湖 北 、', 0, 2]]
4
The segment result whose probability is 1.2067983910720204e-13 : 湖 北 、
[['天津', 0, 1], ['津', 1, 1], ['、', 2, 2], ['天 津 、', 0, 2]]
4
The segment result whose probability is 8.840555474545232e-12 : 天 津 、
[['州', 1, 1], ['等', 2, 2], ['地', 3, 3], ['，', 4, 4], ['贵州 等', 0, 2], ['贵
6
The segment result whose probability is 1.4344832568573035e-19 : 贵 州 等 地 ，
[['由', 1, 1], ['党', 2, 2], ['党政', 2, 3], ['政', 3, 3], ['主', 4, 4], ['主要
48
The segment result whose probability is 2.0173516600582954e-31 : 都 由 党政 主
```

双击（或按回车键）即可修改

```python
print("total result is: ")
print(result)
```

```
刘方仁 当选 贵州省 人大 主任 吴亦侠 当选 贵州省 省长
新华社 贵阳 １月 １６日 电 （ 记者 龙向超 、 石新荣 ） 贵州省 第九 届 人民 代表大会 第一
北京 增 开 列车 疏散 滞 京 旅客
据 新华社 北京 １月 １６日 电 （ 许燕莉 、 潘善棠 ） 据 铁路 部门 介绍 ， 因 华北 地区 替
为 使 震区 灾民 过 上 温暖 的 春节 （ 附 图 片 ２ 张 ）
１ ： 为 加紧 搭建 防寒屋 ， １５日 又 有 ３０００ 多 名 官兵 冒 雪 进驻 地震 灾区 。
（ 新华社 记者 李刚 摄 ）
２ ： 张北县 乱石山村 村民 白秀梅 （ 右 ） 一 家住 进 了 解放军 搭建 的 保温屋 。
（ 周 文 广 摄 ） （ 新华社 稿 ）
李岚清 在 全 国 教职工 住房 建设 会议 上 要求 几 年内 根本 解决 教职工 住房 困难
本报 南京 １月 １６日 讯 新华社 记者 尹鸿祝 、 本报 记者 毕全忠 报道 ： 中共中央 政治局 常
李岚清 指出 ， １９９４ 年 以来 ， 国务院 办公厅 每年 召开 一 次 全国性 会议 ， 专门 部署
李 岚 清 说 ， 看到 他们 能 住 上 上百 平方米 的 住房 ， 感 到 很 高 兴 。 他 对 江苏省
李岚清 强调 ， 教职工 住房 建设 的 发展 还 不 平衡 ， 要 进一步 提高 认识 ， 采取 切实 措
（ Ａ 、 Ｂ ）
李岚清 指出 ， 青年 教师 是 高校 教师 队伍 中 的 一个 特殊 而 重要 的 群体 。 他们 工资
李岚清 指出 ， 要 克服 困难 ， 群策群力 ， 特别 是 青年 教师 的 住房 困难 问题 在 ３ 年
１５日 至 １６日 ， 李岚清 同志 在 国家教委 和 江苏 省委 、 省政府 负责 同志 朱开轩 、 陈
这次 会议 是 １２日 召开 的 。 国家教委 党组 书记 、 副 主任 陈至立 作 了 工作 报告 ， 国
李岚清 致信 全国 海关 关长 会议 指出 把 海关 改革 与 建设 提高 到 新 水平
本报 北京 １月 １６日 讯 记者 赵志文 报 道 ： １月 １２日 至 １６日 ， 全国 海关 关长 会
李 岚 清 说 ， １９９７ 年 ， 海关 征税 、 上 缴 打击 走私 罚没 收入 以及 其他 行政 性 收
李岚清 指出 ， 海关 是 国家 关税 和 进口 环节税 征收 机关 ， 担负 着 维护 国家 利益 ， 保
 充分 认清 形势 ， 以 邓小平 理论 和 党 的 十五大 精神 统揽全局 ， 认真 履行 监管 职能 ，
今年 的 全国 海关 关长 会议 确定 了 实 现 海关 改革 和 建设 两步走 的 跨 世纪 发展 战略
会 议 还 提 出 ， 在 建立 现代 海关 制度 过程 中 ， 通关 作业 改革 是 中心 环节 和 突破口
国务院 新闻办 就 当前 金融 形势 举行 发布会 我国 金融业 在 改革 中 平稳 发展
● 人民币 不 贬值 是 中国 对 亚洲 金融 稳定 作出 的 贡献
● 支持 香港 特区 政府 捍卫 联汇制
● 认真 汲取 东南亚 金融 风波 教
本报 北京 一月 十六日 讯 记者 施明慎 报道 ： 今天 上午 ， 北京 国际 饭店 彩虹厅 挤满 了
 一九九七 年 我国 金融业 在 改革 中 平稳 发展 ， 金融业 为 促进 经济 发展 和 维护 社会 稳
 东南亚 国家 货币 纷纷 贬值 ， 人民币 是否 也 会 贬值 时 ， 戴相龙 回答 ， 中央 银行 的
```

write result into txt file with different format

```
戴相龙 对于 一九九八 年 我国 经济 增长 速度 将 达 百分之八 以上 ， 以及 物价 涨幅 保持 
```

```python
utf8_f = open('result_utf-8.txt', 'w')
utf8_f.write(result)

gbk_f = open('result_gbk.txt', 'wb')
gbk_f.write(result.encode('gbk'))
```

```
27360
```

```
李 铁 映 说 ， 反对 使用 兴奋剂 是 一 场 长期 的 斗争 。 中国 政府 反对 使用 兴奋剂 的 
```

check out the formatoin

```python
utf8_f = open('result_utf-8.txt', 'rb')
data = utf8_f.read()
print(chardet.detect(data))

gbk_f = open('result_gbk.txt', 'rb')
data = gbk_f.read()
print(chardet.detect(data))
```

```
{'encoding': 'utf-8', 'confidence': 0.99, 'language': ''}
{'encoding': 'GB2312', 'confidence': 0.99, 'language': 'Chinese'}
```

Extension: Apply Viterbi algorithm to Bigram Model

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models. from wikipedia

```python
import numpy as np

class Viterbi(BiGram):
  """Apply Viterbi algorithm to Bigram.

  The Viterbi algorithm is a dynamic programming algorithm for finding the most
  likely sequence of hidden states—called the Viterbi path—that results in a
  sequence of observed events, especially in the context of Markov information
  sources and hidden Markov models.

  # Arguments:
    corpus:
    word_count_dict:
    bigram_dict:
    max_split: a int numrepresenting the sliding window's size
    word_num: a int num representing length of dictionary's words

  # References:
    - [Viterbi_algorithm](https://en.wikipedia.org/wiki/Viterbi_algorithm)

  """


  def __init__(self, corpus, word_count_dict, bigram_dict, max_split=4):
    self.corpus = corpus
    self.word_count_dict = word_count_dict
    self.max_split = max_split
    self.bigram_dict = bigram_dict
    self.word_num = len(word_count_dict)

  def segment_position(self, sentence):

    len_sen = len(sentence)

    s = np.full((len_sen, len_sen), -float('inf'))
    h = []
    for i in range(len_sen):
      for j in range(i+1):
        if j == 0:
          s[0][i] = self.log_prob(len_sen, sentence[0:i+1])
        else:
          tmp = []
          for k in range(j):
            print(sentence[j:i+1], sentence[k:j])
            print(i, j, k)
            print("---")
            tmp_s = s[k][j-1] + self.log_prob(len_sen, sentence[k:j], sentence[
            tmp.append(tmp_s)
          print("temp:", tmp)
          print("max:", np.max(tmp))
          print("----")
          s[j][i] = np.max(tmp)

      pos = np.where(s[:, i] == np.max(s[:, i]))[0][0]
      h.append(pos)

  return h




  def log_prob(self, len_sen, obs_word, con_word = None, display=False):
    count = 1

    if not con_word:
      if obs_word in self.word_count_dict.keys():
        count = self.word_count_dict[obs_word]
        prob = np.log(count / (self.word_num + len_sen))
      else:
        prob = -float('inf')
    else:
      bigram = obs_word + ' ' + con_word
      if bigram in self.bigram_dict.keys():
        count = self.bigram_dict[bigram]
        prob = np.log(count / (self.word_num + len_sen))
      else:
        if con_word or obs_word not in self.word_count_dict.keys():
          prob = -float('inf')

    return prob

```