

现代程序设计第4周作业

谢奕飞 20377077

代码

```
1 import pandas as pd
2 from tqdm import tqdm
3 import jieba
4 import matplotlib.pyplot as plt
5
6 def GetText(file_path):
7     '''
8     数据读取函数
9     传入txt文件绝对路径
10    返回text列表
11    '''
12    f=open(file_path,'r',encoding='utf8')
13    data=[]
14    for line in tqdm(f.readlines()):
15        text=''
16        message=line.split()
17        for each in message[2:-8]:
18            text+=' '+each
19        data.append(text)
20    f.close()
21    return data
22
23 class Tokenizer:
24     '''
25     对自然语言进行编码
26     '''
27     def __init__(self,texts,coding='c',PAD=0):
28         '''
29         输入将要需要操作的文本（一个字符串的列表）
30         这里需要完成词典的构建（即汉字到正整数的唯一映射的确定）
31         注意构建词典 一是要根据coding来选择按词构建（coding='w'），还是按字构建，默认
32         '''
33         self._texts=texts
34         self._coding=coding
35         self._PAD=PAD
36         self._dic={}
37         self._codes={PAD,}
38         self._dic.update({'PAD':PAD})
```

```

39         code=0
40         for text in tqdm(texts,desc='initing dictionary...'):
41             if coding=='w':
42                 text=jieba.lcut(text)
43             for word in text:
44                 if word not in self._dic:
45                     while code in self._codes:
46                         code+=1
47                     self._dic.update({word:code})
48                     self._codes.add(code)
49     def tokenize(self,sentence):
50         '''
51         输入一句话，返回分词(字)后的字符列表(list_of_chars)
52         '''
53         res=[]
54         for each in tqdm(jieba.cut(sentence),desc='tokenizing...'):
55             res.append(each)
56         return res
57
58     def encode(self,list_of_chars):
59         '''
60         输入字符(字或者词)的字符列表，返回转换后的数字列表(tokens)
61         '''
62         tokens=[]
63         for word in list_of_chars:
64             tokens.append(self._dic[word])
65         return tokens
66
67     def trim(self,tokens,seq_len):
68         '''
69         输入数字列表tokens，整理数字列表的长度。不足seq_len的部分用PAD补足，超过的
70         '''
71         if len(tokens)>=seq_len:
72             return tokens[:seq_len]
73         else:
74             return tokens.extend([self._PAD for i in range(len(tokens)-seq_1
75
76     def decode(self,tokens):
77         '''
78         将模型输出的数字列表翻译回句子。如果有PAD，输出'[PAD]'
79         '''
80         res=''
81         dic_list=[self._PAD for i in range(len(self._dic))]
82         for each in self._dic:
83             dic_list[self._dic[each]]=each
84         for each in tokens:
85             if each==self._PAD:
86                 res+='[PAD]'
87             else:
88                 res+=dic_list[each]
89         return res

```

```

90
91     def encode_all(self, seq_len):
92         '''
93         返回所有文本(chars)的长度为seq_len的tokens
94         '''
95         codes = [[] for i in range(len(self._texts))]
96         i = 0
97         for text in tqdm(self._texts, desc='encoding all...'):
98             if self._coding == 'w':
99                 text = jieba.lcut(text)
100             for word in text:
101                 codes[i].append(self._dic[word])
102             i += 1
103         for code in tqdm(codes, desc='triming...'):
104             if len(code) < seq_len:
105                 code.extend([self._PAD for i in range(seq_len - len(code))])
106             else:
107                 code = code[:seq_len]
108         return codes
109     def output_distribution(self):
110         '''
111         输出编码分布
112         '''
113         xy = {}
114         for text in self._texts:
115             if self._coding == 'w':
116                 text = jieba.lcut(text)
117             length = len(text)
118             if length in xy:
119                 xy[length] += 1
120             else:
121                 xy.update({length: 1})
122         x = sorted(xy.keys())
123         y = [xy[key] for key in tqdm(x, desc='loading y...')]
124         plt.bar(x, y)
125         plt.xlabel('length')
126         plt.ylabel('distribution')
127         plt.show()
128
129 if __name__ == '__main__': main()

```

测试encode_all()

```

1 def main():
2     '''
3     test
4     '''
5     file_path = 'D:\Project\Python\week5Tokenizer\\final_none_duplicate0.txt'
6     texts = GetText(file_path)
7     token = Tokenizer(texts)
8     print(token.encode_all(10))

```

```
100% 1000/1000 [00:00<00:00, 249928.73it/s]
initing dictionary...: 100% 1000/1000 [00:00<00:00, 88472.49it/s]
encoding all...: 100% 1000/1000 [00:00<00:00, 83321.16it/s]
triming...: 100% 1000/1000 [00:00<00:00, 499857.47it/s]
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], [1, 17, 18, 18, 19, 20, 21, 22, 23, 24, 25, 8, 26, 27, 13, 28,
29, 30, 31, 32, 33, 34, 20, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 24, 47, 48, 13, 49, 50, 49, 37, 21, 51, 37,
24, 47, 48, 13, 20, 50, 20, 37, 52, 53, 28, 29, 30, 32, 54, 55, 14, 56, 57, 55, 9, 58, 59, 33, 60, 61, 62, 63, 64, 65,
37, 66, 67, 48, 68, 69, 35, 36], [1, 70, 71, 72, 14, 73, 74, 75, 76, 13, 77, 78, 79, 80, 81, 82, 83, 84, 85, 81], [1, 86
, 87, 48, 88, 89, 90, 91, 92, 93, 5, 94, 95, 96, 13, 97, 98, 99, 23, 61, 100, 101, 102, 48, 103, 89, 104, 105, 72, 106,
97, 13, 107, 108, 109, 110, 111, 112, 96, 113, 64, 114, 115, 116, 117, 118, 119, 13, 120, 96, 25, 121, 122, 107, 116, 96
, 123, 118, 45, 124, 125, 126, 127, 128, 129, 130, 131, 106, 132, 119, 133, 134, 124, 135, 136, 137, 138, 139, 133, 134,
124, 75, 4, 140, 141, 133, 113, 69, 3, 142, 143, 144, 107, 108, 121, 145, 111, 116, 96, 113, 77, 146, 86, 87, 147, 26,
148, 61, 48, 5, 149, 5, 142, 150, 94, 95, 151, 98, 99, 25, 152, 153, 154, 13, 75, 110, 155, 156, 157, 157, 158, 159, 113
], [1, 160, 11, 161, 162, 11, 161, 163, 164, 165, 165, 163, 164, 2, 166, 11, 161, 33, 167, 168, 169, 170, 171, 172, 173,
1, 174, 175, 176, 177], [1, 11, 92, 178, 179, 180, 96, 0, 0, 0], [1, 181, 182, 183, 184, 48, 185, 186, 187, 154, 188, 1
, 189], [1, 190, 25, 191, 192, 193, 24, 37, 194, 195, 196, 48, 197, 198, 150, 94, 95, 151], [1, 25, 118, 5, 48, 199, 14,
200, 32, 201, 202, 203, 75, 48, 204, 18, 13, 205, 108, 206, 207, 208, 43, 44, 48, 209, 210, 13, 62, 211, 209, 210, 212,
48, 213, 204, 13, 214, 215, 11, 216, 217, 218, 219, 21, 13, 26, 156, 11, 220, 221, 53], [1, 25, 222, 223, 104, 224, 225
, 226, 227, 65, 93, 41, 228, 229, 230, 48, 231, 232, 37, 233, 234, 235, 13, 236, 237, 81, 238, 239, 240, 241, 144, 53],
[1, 37, 194, 242, 243, 13, 107, 194, 244, 0], [1, 245, 246, 98, 99, 25, 146, 13, 72, 194, 97, 247, 248, 25, 249, 241, 14
4, 22, 250, 251, 252, 243, 253, 208, 96, 254, 13, 255, 190, 256, 50, 257, 258, 259, 33, 96], [1, 11, 260, 261, 11, 21, 2
62, 25, 263, 50, 37, 213, 13, 70, 203, 264, 265, 11, 212, 48, 193, 266, 37, 121, 33, 13, 267, 32, 268, 269, 96, 51, 11,
37, 270, 271, 272, 273, 61, 39, 274, 203, 206, 275, 13, 11, 276, 277, 48, 278, 33, 48, 193, 92, 203, 96, 13, 213, 203, 2
79, 73, 11, 212, 48, 280, 202, 37, 96, 13, 281, 11, 50, 8, 275, 282, 283, 13, 107, 108, 284, 6, 96, 111, 285, 286, 52, 2
47, 247, 287, 287, 288, 289, 290, 121, 33, 13, 291, 292, 11, 293, 96, 294, 34, 295, 8, 296, 297, 298], [1, 213, 194, 61,
5, 299, 11, 300, 301, 150, 213, 302, 151], [1, 303, 292, 47, 304, 52, 305, 61, 146, 303, 306, 307, 308, 307, 309, 310,
311, 170, 310, 312, 1, 150, 313, 47, 151, 150, 313, 47, 151, 150, 314, 314, 151, 150, 314, 314, 151, 150, 314, 151,
62, 315, 316, 317, 97, 47, 70, 318, 319, 70, 320, 173, 169, 169, 321, 81, 322, 323, 24, 324, 325, 81, 155, 156, 326, 327
, 123, 328, 329, 97, 327, 25, 328, 209, 330, 324, 331, 332, 48, 333, 313, 81, 77, 146, 116, 334, 47, 335, 336, 195, 62,
315, 337, 14, 62, 315, 338, 62, 315, 339, 340, 150, 314, 314, 151, 150, 314, 314, 151, 150, 314, 314, 151, 212, 24, 341, 2
```

修改传参

```
1 def main():
2     '''
3     test
4     '''
5     file_path='D:\Project\Python\week5Tokenizer\\final_none_duplicate0.tx
6     t'
7     texts=GetText(file_path)
8     token=Tokenizer(texts,coding='w',PAD=-1)
9     print(token.encode_all(10))
```

```
100% 1000/1000 [00:00<00:00, 248669.24it/s]
initing dictionary...: 0%
building prefix dict from the default dictionary ...
Loading model from cache C:\Users\NO_THA_1\AppData\Local\Temp\jieba.cache
Loading model cost 0.550 seconds.
Prefix dict has been built successfully.
initing dictionary...: 100% 1000/1000 [00:00<00:00, 1063.56it/s]
encoding all...: 100% 1000/1000 [00:00<00:00, 2603.44it/s]
triming...: 100% 1000/1000 [00:00<00:00, 498313.41it/s]
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [0, 11, 12, 13, 14, 15, 16, 17, 8, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 8, 33, 34, 33, 35, 36, 37, 31, 32, 8, 38, 34, 38, 39, 40, 18, 19, 41, 42, 43, 44, 45, 46, 47, 48, 49, 32, 50
, 23, 24], [0, 51, 52, 53, 54, 8, 55, 56, 57, 58, 59, 57], [0, 60, 32, 61, 62, 63, 64, 65, 66, 67, 8, 68, 69, 70, 71, 72
, 32, 73, 74, 75, 68, 8, 76, 77, 78, 79, 67, 80, 81, 82, 83, 84, 85, 8, 86, 67, 87, 88, 89, 83, 67, 90, 29, 91, 92, 93,
94, 85, 95, 96, 91, 97, 98, 99, 95, 96, 91, 100, 95, 80, 101, 102, 76, 103, 78, 83, 67, 80, 104, 60, 105, 32, 106, 107,
66, 108, 69, 109, 110, 8, 111, 112, 113, 114, 80], [0, 115, 116, 117, 118, 119, 120, 121, 122, 117, 21, 123, 124, 0, 125
, 126, 127], [0, 6, 63, 128, 129, 67, -1, -1, -1, -1], [0, 130, 32, 131, 132, 0, 133, -1, -1, -1], [0, 134, 135, 136, 13
7, 138, 139, 32, 140, 107, 66, 108], [0, 141, 65, 32, 142, 143, 144, 145, 32, 146, 8, 147, 148, 149, 28, 32, 150, 8, 151
, 150, 152, 153, 8, 154, 6, 155, 156, 14, 8, 157, 6, 158, 40], [0, 159, 160, 161, 162, 163, 164, 165, 64, 166, 167, 168,
32, 169, 170, 8, 171, 57, 172, 173, 174, 40], [0, 138, 175, 8, 176, 177, -1, -1, -1, -1], [0, 178, 69, 179, 8, 180, 181
, 182, 174, 15, 183, 184, 185, 67, 186, 8, 187, 188, 34, 189, 190, 67], [0, 6, 191, 192, 193, 194, 34, 195, 8, 196, 197,
6, 152, 136, 198, 48, 199, 8, 200, 201, 67, 36, 6, 48, 202, 203, 204, 205, 206, 207, 8, 6, 208, 32, 209, 32, 136, 63, 2
06, 67, 8, 210, 211, 6, 152, 212, 213, 8, 214, 6, 34, 215, 216, 8, 76, 217, 218, 67, 78, 219, 220, 221, 222, 199, 8, 223
, 6, 224, 225, 226, 227], [0, 228, 229, 230, 231, 6, 232, 107, 228, 233, 108], [0, 234, 235, 31, 236, 237, 238, 239, 234
, 240, 241, 0, 107, 242, 31, 108, 107, 242, 31, 108, 107, 243, 108, 107, 243, 108, 244, 245, 246, 31,
247, 248, 247, 249, 57, 250, 137, 251, 57, 112, 252, 253, 254, 68, 255, 256, 257, 258, 32, 259, 57, 104, 260, 31, 261,
262, 244, 263, 244, 264, 244, 265, 107, 243, 108, 107, 243, 108, 107, 243, 108, 266, 267, 268, 269, 57, 270, 271, 272, 3
2, 140, 269, 107, 273, 108, 107, 273, 108], [0, 38, 274, 0, 6, 275, 234, 276, 277, 234], [0, 6, 275, 278, 40, 279, -1, -
1, -1, -1], [0, 280, 281, 32, 48, 242, 282, 8, 283, -1], [0, 69, 284, 32, 285, 286, 31, 287, 288, 8, 289, 290], [0, 234,
291, 87, 292, 293, 234, 291, 87, 32, 294, 295, 107, 296, 108, 107, 296, 108, 107, 296, 108, 291, 87, 297, 298, 299, 300
, 301, 162, 297, 32, 302, 8, 303, 304, 305, 306, 307, 308, 28, 32, 309, 8, 157, 310, 311, 312, 313, 314, 67, 8, 315, 316 2
```

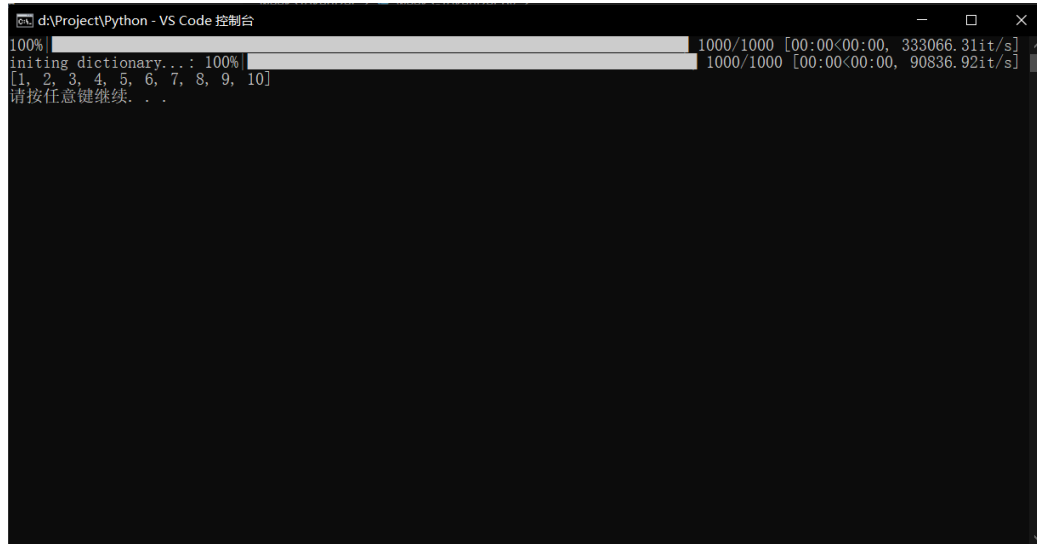
(以前1000行数据为例)

测试tokenize ()

```
1 def main():
2     '''
3     test
4     '''
5     file_path='D:\Project\Python\week5Tokenizer\\final_none_duplicate0.txt'
6     texts=GetText(file_path)
7     token=Tokenizer(texts)
8     print(token.tokenize(texts[0]))
```

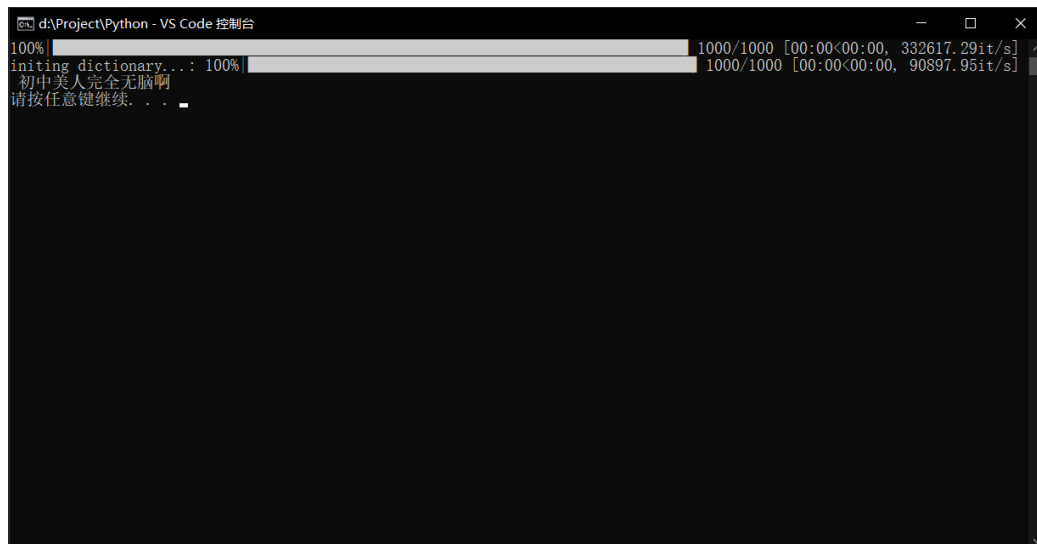


```
9 print(token.trim(code,10))
```



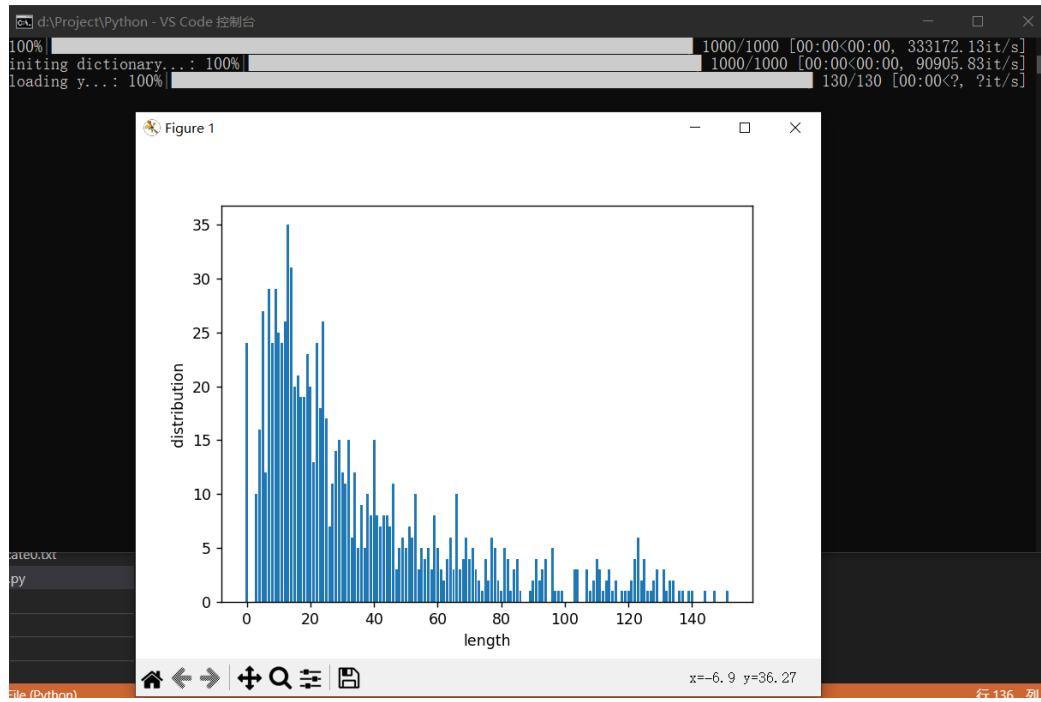
测试decode()

```
1 def main():
2     '''
3     test
4     '''
5     file_path='D:\Project\Python\week5Tokenizer\\final_none_duplicate0.txt'
6     texts=GetText(file_path)
7     token=Tokenizer(texts)
8     code=token.encode(texts[0])
9     code=token.trim(code,10)
10    print(token.decode(code))
```



测试output_distribution()

```
1 def main():
2     '''
3     test
4     '''
5     file_path='D:\Project\Python\week5Tokenizer\\final_none_duplicate0.txt'
6     texts=GetText(file_path)
7     token=Tokenizer(texts)
```



从柱状图形状来看，长度的分布类似服从泊松分布

应该取60左右的长度作为标准长度，因为大部分句子的长度 < 40