隐马尔可夫模型分词

徐翔

2021 年 4 月 2 日

目录

1 导入语料库

[1]:

```python
import re
import os
import time
from graphviz import *
from math import log
```

1.1 加载词典

[2]:

```python
def load_dictionary(dict_file):
    """
    加载词库
    :return: 一个 set 形式的词库
    """
    fr = open(dict_file,encoding="utf-8")
    word_list = [item.strip().split("\t")[0] for item in fr]
    return set(word_list)
```

[3]:

```python
sighan05 = "C:/Users/apple/Desktop/徐翔资料/大学课堂资料/数据科学/自然语言处理/实验课件/练习 2/第二届国际中文分词评测/icwb2-data/"
msr_dict = os.path.join(sighan05, 'gold', 'msr_training_words.utf8')
msr_train = os.path.join(sighan05, 'training', 'msr_training.utf8')
msr_test = os.path.join(sighan05, 'testing', 'msr_test.utf8')
msr_output = os.path.join(sighan05, 'testing', 'msr_output.txt')
msr_gold = os.path.join(sighan05, 'gold', 'msr_test_gold.utf8')

word_dict = load_dictionary(msr_dict)
```

1.2 加载训练集

[4]:

```python
fr = open(msr_train,encoding="utf-8")
sent_list = [item[:-1].split(' ') for item in fr]
for i in range(len(sent_list)):
    sent_list[i] = [word for word in sent_list[i] if len(word)>0]
sent_list = [li for li in sent_list if len(li)>0]

print(sent_list[0])
```

```
['“', '人们', '常', '说', '生活', '是', '一', '部', '教科书', '，', '
而', '血', '与', '火', '的',
'战争', '更', '是', '不可多得', '的', '教科书', '，', '她', '确实', '是',␣
↪'名副其实', '的', '‘', '我',
'的', '大学', '’', '。']
```

2 训练隐马尔可夫模型

2.1 {B,M,E,S} 序列标注

[5]:

```
states = ['B', 'M', 'E', 'S']
pi = {'B':0,'S':0}
transition_probability = dict(zip(states, [{} for i in range(len(states))]))
emission_probility = dict(zip(states, [{} for i in range(len(states))]))
```

[6]:

```
def add(string,dic):
    if string in dic:                          # 字典中已存在对应的键
        dic[string] = dic[string] + 1
    else:                                      # 字典中不存在对应的键
        dic[string] = 1

for li in sent_list:
    seq = []                                   # 用于存储序列标注
    for word in li:
        if len(word) ==1:
            seq.append('S')
            add(word,emission_probility['S'])
        elif len(word) == 2:
            seq.append('B')
            seq.append('E')
            add(word[0],emission_probility['B'])
            add(word[1],emission_probility['E'])
        else:
            seq.append('B')
            add(word[0],emission_probility['B'])
            for i in range(1,len(word)-1):
                seq.append('M')
                add(word[i],emission_probility['M'])
            seq.append('E')
            add(word[0],emission_probility['E'])

    if seq[0] == 'B':
        pi['B'] = pi['B']+1
    else:
        pi['S'] = pi['S']+1

    for i in range(len(seq)-1):
        add(seq[i+1],transition_probability[seq[i]])
```

2.2 估计初始状态概率向量

[7]:
```
print(pi)

MIN_FLOAT = -3.14e100
number = sum(pi.values())
for key in pi.keys():
    pi[key] = log(pi[key]/number)
pi['M'] = MIN_FLOAT
pi['E'] = MIN_FLOAT

print(pi)
```

```
{'B': 60460, 'S': 26459}
{'B': -0.3629946611387064, 'S': -1.1893802854123139, 'M': -3.14e+100, 'E':
-3.14e+100}
```

2.3 估计转移概率矩阵

[8]:
```
for state in states:
    number = sum(transition_probability[state].values())
    for key in transition_probability[state].keys():
        transition_probability[state][key] =␣
 ↪log(transition_probability[state][key]/number)
```

2.4 估计发射概率矩阵

[9]:
```
for state in states:
    number = sum(emission_probility[state].values())
    for key in emission_probility[state].keys():
        emission_probility[state][key] = log(emission_probility[state][key]/
↪number)
```

3 中文分词

3.1 维特比算法

[10]:

```
PrevStatus = {
    'B': 'ES',
    'M': 'MB',
    'S': 'SE',
    'E': 'BM'
}
```

[11]:

```
def viterbi(obs, states, start_p, trans_p, emit_p):
    V = [{}]  # tabular
    path = {}
    for y in states:  # init
        V[0][y] = start_p[y] + emit_p[y].get(obs[0], MIN_FLOAT) # emit_p[y].
↪get(obs[0], MIN_FLOAT) 从状态 y 发射到 obs[0] 的概率
        path[y] = [y]
    for t in range(1, len(obs)):
        V.append({})
        newpath = {}
        for y in states:
            em_p = emit_p[y].get(obs[t], MIN_FLOAT)
            (prob, state) = max(
                [(V[t - 1][y0] + trans_p[y0].get(y, MIN_FLOAT) + em_p, y0) for␣
↪y0 in PrevStatus[y]])
            V[t][y] = prob
            newpath[y] = path[state] + [y]
        path = newpath
    (prob, state) = max((V[len(obs) - 1][y], y) for y in 'ES')

    return (prob, path[state])
```

```python
sentence = "商品和服务"
print(viterbi(sentence, 'BMES', pi, transition_probability, emission_probility))
```

```
(-30.507856628263557, ['B', 'E', 'S', 'B', 'E'])
```

[12]:

```python
def __cut(sentence):
    global emit_P # 在函数内部对函数外的变量进行操作
    prob, pos_list = viterbi(sentence, 'BMES', pi, transition_probability,␣
 ↪emission_probility)
    begin, nexti = 0, 0
    for i, char in enumerate(sentence):
        pos = pos_list[i]
        if pos == 'B':
            begin = i
        elif pos == 'E':
            yield sentence[begin:i + 1]
            nexti = i + 1
        elif pos == 'S':
            yield char
            nexti = i + 1
    if nexti < len(sentence):
        yield sentence[nexti:]

print(list(__cut(sentence)))
```

```
['商品', '和', '服务']
```

3.2 评测准确率和速度

[13]:

```python
def to_region(segmentation: str) -> list:
    """
    将分词结果转换为区间
    :param segmentation: 商品 和 服务
    :return: [(0, 2), (2, 3), (3, 5)]
    """
    region = []
    start = 0
    for word in re.compile("\\s+").split(segmentation.strip()):
        end = start + len(word)
        region.append((start, end))
        start = end
    return region


def prf(gold: str, pred: str, dic) -> tuple:
    """
    计算 P、R、F1
    :param gold: 标准答案文件，比如“商品 和 服务”
    :param pred: 分词结果文件，比如“商品 和服 务”
    :param dic: 词典
    :return: (P, R, F1, OOV_R, IV_R)
    """
    A_size, B_size, A_cap_B_size, OOV, IV, OOV_R, IV_R = 0, 0, 0, 0, 0, 0, 0
    with open(gold,encoding="utf-8") as gd, open(pred,encoding="utf-8") as pd:
        for g, p in zip(gd, pd):
            A, B = set(to_region(g)), set(to_region(p))
            A_size += len(A)
            B_size += len(B)
            A_cap_B_size += len(A & B) # A 与 B 的交集，分词结果中正确的部分
            text = re.sub("\\s+", "", g)
            for (start, end) in A:
                word = text[start: end]
                if word in dic:
                    IV += 1  # 登陆词的词频统计
                else:
                    OOV += 1 # 未登陆词的词频统计

            for (start, end) in A & B:
                word = text[start: end]
                if word in dic:
                    IV_R += 1
                else:
                    OOV_R += 1
```

```python
    p, r = A_cap_B_size / B_size * 100, A_cap_B_size / A_size * 100
    return p, r, 2 * p * r / (p + r), OOV_R / OOV * 100, IV_R / IV * 100
```

[14]:

```python
with open(msr_gold,encoding="utf-8") as test, open(msr_output, 'w', encoding="utf-8") as output:
    time0 = time.time()
    for line in test:
        output.write("  ".join(list(__cut(re.sub("\\s+", "", line)))))
        output.write("\n")
print("Time:%.2f" % float(time.time()-time0))
print("P:%.2f R:%.2f F1:%.2f OOV-R:%.2f IV-R:%.2f" % prf(msr_gold, msr_output, word_dict))
```

```
Time:6.11
P:77.20 R:79.43 F1:78.30 OOV-R:34.61 IV-R:80.65
```

4 整理输出的样式

[15]:

```python
with open(msr_gold,encoding="utf-8") as test, open(r'C:\Users\apple\i love python\自然语言处理\output.txt','w') as output:
    for line in test:
        sent = line.strip().replace(" ", "")
        prob, pos_list = viterbi(sent, 'BMES', pi, transition_probability, emission_probility)
        for i in range(len(sent)):
            output.write(sent[i]+' '+pos_list[i])
            output.write("\n")
        output.write("\n")
```