DLNLP_2023 第一次作业:计算中文平均信息熵

ZY2203106 李文雯

理论推导

信息熵:

信息熵的概念最早由香农于1948年借鉴热力学中的"热熵"的概念提出，旨在表示信息的不确定性。熵值越大，则信息的不确定程度越大。其数学公式可以表示为

$$H(x)=\sum_{x\in X}P(x)\log\left(\frac{1}{P(x)}\right)=-\sum_{x\in X}P(x)\log\left(P(x)\right)$$

论文中假设:

$$X=\{\ldots,X_{-2},X_{-1},X_0,X_1,X_2,\ldots\}$$

是基于有限字母表的平稳随机过程， P为X的概率分布，Ep为P的数学期望，则X的信息熵定义为

$$H(X)\equiv H(P)\equiv -E_P\log P\left(X_0\mid X_{-1},X_{-2},\ldots\right)$$

当对数底数为2时，信息熵的单位为bit，相关理论说明H(P)可以表示为

$$H(P)=\lim_{n\to\infty}-E_P\log P\left(X_0\mid X_{-1},X_{-2},\ldots,X_{-n}\right)=\lim_{n\to\infty}-\frac{1}{n}E_P\log P\left(X_1X_2\ldots X_n\right)$$

如果P是遍历的，则Ep=1 。我们无法精确获取X的概率分布，即无法获取精确的P，但可以通过足够长的随机样本来估计P，通过建立P的随机平稳过程模型M来估算H(P)的上界，与上述推理过程相同，我们可以得到以下公式:

$$H(P,M)=\lim_{n\to\infty}-E_P\log M\left(X_0\mid X_{-1},X_{-2},\ldots,X_{-n}\right)=\lim_{n\to\infty}-\frac{1}{n}E_M\log P\left(X_1X_2\ldots X_n\right)$$

H(P, M)有较大的参考价值，因为其是H(P)的一个上界，即H(P)<H(P, M)，更加准确的模型能够产生更加精确的上界。从文本压缩的角度来理解信息熵，对于$X_1X_2\ldots X_n$的任意编码方式，$l\left(X_1X_2\ldots X_n\right)$为编码所需的比特数，均有

$$E_Pl\left(X_1X_2\ldots X_n\right)\geq -E_P\log P\left(X_1X_2\ldots X_n\right)$$

由上述分析知， H(P)是对从P中提取的长字符串进行编码所需的每个符号的平均位数的下限，每个符号编码时需要
的位数越多，即熵越高，说明混乱程度越高，单个字符携带的信息量越大。

分词模型:

当k=0时，对应的模型为一元模型，即Wi不与任何词相关，每个词都是相互独立

$$P\left(W_1,W_2,\ldots,W_k\right)=\prod_i P\left(W_i\right)$$

当k=1时，对应的模型为二元模型，即Wi只与它前面的一个词相关

$$P\left(W_1,W_2,\ldots,W_k\right)=\prod_i P\left(W_i\mid W_{i-1}\right)$$

当k=2时，对应的模型为三元模型，即Wi只与它前面的两个词相关

$$P(W_1, W_2, \ldots, W_k) = \prod_i P(W_i \mid W_{i-2}, W_{i-1})$$

程序简介

程序运行

运行程序work.py即可在命令行得到金庸小说全集和单本的信息熵表

算法流程

1.数据预处理

删除文章开头的无关信息、标点符号、换行符、分页符等以及cn_stopwords.txt给出的停词

```
def read_data(file_path,path):#获取file_path文件对应的内容
    corpus = ''
    r = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~
「」『』（）]+'
    with open(file_path, 'r', encoding='ANSI') as f:
        corpus = f.read()
        corpus = re.sub(r, '', corpus)
        corpus = corpus.replace('\n', '')
        corpus = corpus.replace('\u3000', '')
        corpus = corpus.replace('本书来自www.cr173.com免费txt小说下载站\n更多更新免费电
子书请关注www.cr173.com', '')
        f.close()
    with open(path, 'r', encoding='UTF-8') as f:
        stopwords = []
        for a in f:
            if a != '\n':
                stopwords.append(a.strip())
    for a in stopwords:
        corpus = corpus.replace(a, '')
    return corpus
```

2.计算词频

```
def word_tf(tf1, words):#获取一元词词频

    for i in range(len(words)):
        tf1[words[i]] = tf1.get(words[i], 0) + 1

def bigram_tf(tf2, words):#获取二元词词频

    for i in range(len(words)-1):
        tf2[(words[i], words[i+1])] = tf2.get((words[i], words[i+1]), 0) + 1

def trigram_tf(tf3, words):#获取三元词词频

    for i in range(len(words)-2):
        tf3[((words[i], words[i+1]), words[i+2])] = tf3.get(((words[i],
words[i+1]), words[i+2]), 0) + 1
```

3.计算信息熵

```
def calculate_entropy1(words_tf, length_data):#计算一元词的信息熵
    t1 = time.time()
    words_num = sum([item[1] for item in words_tf.items()])

    print("分词种类数：{}".format(words_num))
    print('不同词个数：{}'.format((len(words_tf))))
    print("平均词长：{:.4f}".format(length_data/float(words_num)))

    entropy = 0
    for item in words_tf.items():
        entropy += -(item[1]/words_num) * math.log(item[1]/words_num, 2)
    print("基于分词的一元模型中文信息熵为：{:.4f} 比特/词".format(entropy))

    print("一元模型运行时间：{:.4f} s".format(time.time() - t1))
    return ['unigram model', length_data, len(words_tf),
round(length_data/float(words_num), 4), round(entropy, 4)]

def calculate_entropy2(words_tf, bigram_tf, length_data):#计算二元词的信息熵
    t1 = time.time()
    bi_words_num = sum([item[1] for item in bigram_tf.items()])
    avg_word_length = sum(len(item[0][i]) for item in bigram_tf.items() for i in
range(len(item[0]))) / len(bigram_tf)

    print("分词种类数：{}".format(bi_words_num))
    print('不同词个数：{}'.format((len(bigram_tf))))
    print("平均词长：{:.4f}".format(avg_word_length))

    entropy = 0
    for bi_item in bigram_tf.items():
        jp = bi_item[1] / bi_words_num
        cp = bi_item[1] / words_tf[bi_item[0][0]]
        entropy += -jp * math.log(cp, 2)
    print("基于分词的二元模型中文信息熵为：{:.4f} 比特/词".format(entropy))

    print("二元模型运行时间：{:.4f} s".format(time.time() - t1))
    return ['bigram model', length_data, len(bigram_tf), round(avg_word_length,
4), round(entropy, 4)]

def calculate_entropy3(bigram_tf, trigram_tf, length_data):#计算三元词的信息熵

    t1 = time.time()
    tri_words_num = sum([item[1] for item in trigram_tf.items()])
    avg_word_length = sum(len(item[0][i]) for item in trigram_tf.items() for i
in range(len(item[0])))/len(trigram_tf)

    print("分词种类数：{}".format(tri_words_num))
    print('不同词个数：{}'.format((len(trigram_tf))))
    print("平均词长：{:.4f}".format(avg_word_length))

    entropy = 0
    for tri_item in trigram_tf.items():
        jp = tri_item[1] / tri_words_num
        cp = tri_item[1] / bigram_tf[tri_item[0][0]]
        entropy += -jp * math.log(cp, 2)
    print("基于分词的三元模型中文信息熵为：{:.4f} 比特/词".format(entropy))
```

```
    print("三元模型运行时间：{:.4f} s".format(time.time() - t1))
    return ['trigram model', length_data, len(trigram_tf),
round(avg_word_length, 4), round(entropy, 4)]
```

4.以表格形式输出结果

```
def print_md(table_name, head, row_title, col_title, data):
    """
    table_name: 表名 head: 表头 row_title: 行名，编号，1，2，3…… col_title: 列名，词
数，运行时间等 data: {ndarray(H, W)}
    """
    element = " {} |"

    h, w = len(data), len(data[0])
    lines = ['#### {}'.format(table_name)]

    lines += ["| {} | {} |".format(head, ' | '.join(col_title))]

    # 分割线
    split = "{}:{}"
    line = "| {} |".format(split.format('-' * len(head), '-' * len(head)))
    for i in range(w):
        line = "{} {} |".format(line, split.format('-' * len(col_title[i]), '-'
* len(col_title[i])))
    lines += [line]

    # 数据部分
    for i in range(h):
        d = list(map(str, list(data[i])))
        lines += ["| {} | {} |".format(row_title[i], ' | '.join(d))]

    table = '\n'.join(lines)
    print(table)
    return table


def calculate_entropy_all(inf, mode):#计算一系列文件的信息熵
    tf1 = {}
    tf2 = {}
    tf3 = {}
    split_words = []
    data = all_corpus(inf)

    print("语料库字数：{}".format(len(data)))
    if mode == 'token':
        split_words = list(jieba.cut(data))
        words_num = len(split_words)
    elif mode == 'char':
        split_words = [ch for ch in data]
    word_tf(tf1, split_words)
    bigram_tf(tf2, split_words)
    trigram_tf(tf3, split_words)
    rows = []
    print('--------------------------------')
    rows.append(calculate_entropy1(tf1, len(data)))
    print('--------------------------------')
```

```
    rows.append(calculate_entropy2(tf1, tf2, len(data)))
    print('----------------------------------')
    rows.append(calculate_entropy3(tf2, tf3, len(data)))
    print('----------------------------------')

    head = "#"
    row_title = [str(i + 1) for i in range(len(rows))]
    col_title = ['分词模型', '语料字数', '分词种类数', '平均词长', '信息熵']
    print_md('金庸小说全集信息熵表', head, row_title, col_title, rows)


def calculate_entropy_every(mode):#计算DATA_PATH下每个文件单独的信息熵
    tf1 = {}
    tf2 = {}
    tf3 = {}
    split_words = []
    rows = []

    for file in os.listdir(DATA_PATH):
        print("\n当前计算信息熵的文件为：{}".format(file))
        file_path = DATA_PATH + file

        data = read_data(file_path,path)
        print("语料库字数：{}".format(len(data)))
        if mode == 'token':
            split_words = list(jieba.cut(data))
            words_num = len(split_words)
        elif mode == 'char':
            split_words = [ch for ch in data]
        word_tf(tf1, split_words)
        bigram_tf(tf2, split_words)
        trigram_tf(tf3, split_words)
        rows1 = []
        print('----------------------------------')
        rows1.append(calculate_entropy1(tf1, len(data)))
        print('----------------------------------')
        rows1.append(calculate_entropy2(tf1, tf2, len(data)))
        print('----------------------------------')
        rows1.append(calculate_entropy3(tf2, tf3, len(data)))
        print('----------------------------------')


        rows.append([file.split('.')[0], rows1[0][1], rows1[0][2], rows1[0][3],
rows1[0][4],
                     rows1[1][2], rows1[1][3], rows1[1][4],
                     rows1[2][2], rows1[2][3], rows1[2][4]])
    head = '#'
    row_title = [str(i + 1) for i in range(len(rows))]
    col_title = ['小说名', '语料字数', '一元分词个数', '一元平均词长', '一元模型信息熵',
'二元分词个数', '二元平均词长', '二元模型信息熵',
                 '三元分词个数', '三元平均词长', '三元模型信息熵']
    print_md('金庸小说单本信息熵表', head, row_title, col_title, rows)
```

运行结果

金庸小说单本信息熵表(按词)

#	小说名	语料字数	一元分词个数	一元平均词长	一元模型信息熵	二元分词个数	二元平均词长	二元模型信息熵	三元分词个数	三元平均词长	三元模型信息熵	平均信息熵
1	三十三剑客图	34704	9964	1.8599	12.4445	17961	3.7402	1.6574	18505	3.8616	0.0693	4.7237
2	书剑恩仇录	277738	42803	1.6603	13.0502	145048	3.7889	3.8595	164587	3.8709	0.3999	5.7699
3	侠客行	185638	58659	0.6949	13.2122	225063	3.8012	4.2831	261800	3.8683	0.4784	5.9912
4	倚天屠龙记	509279	94112	0.9412	13.4603	435758	3.8269	4.8961	527551	3.8675	0.6216	6.3260
5	天龙八部	622537	132670	0.7105	13.6604	685512	3.8433	5.2845	850638	3.8671	0.7167	6.5539
6	射雕英雄传	483736	160302	0.4263	13.7795	878582	3.8562	5.5006	1101392	3.8703	0.7531	6.6777
7	白马啸西风	33935	162010	0.0294	13.7826	892027	3.8559	5.5171	1119367	3.8697	0.7567	6.6855
8	碧血剑	260911	176401	0.2019	13.8476	995694	3.8604	5.6002	1254370	3.871	0.7726	6.7401
9	神雕侠侣	523319	201150	0.3294	13.8085	1192343	3.8481	5.8346	1536569	3.8548	0.8633	6.8355
10	笑傲江湖	495490	221278	0.2668	13.8063	1366095	3.8564	5.978	1789482	3.8553	0.9339	6.9061
11	越女剑	8877	221752	0.0048	13.8087	1369601	3.8563	5.9792	1794136	3.8553	0.9341	6.9073
12	连城诀	116254	226850	0.0604	13.8226	1413179	3.8578	6.0062	1854506	3.8554	0.9406	6.9231
13	雪山飞狐	69435	230403	0.0354	13.8349	1439380	3.8591	6.0186	1890050	3.8556	0.9431	6.9322
14	飞狐外传	232335	240327	0.1113	13.8595	1524662	3.8634	6.0677	2009375	3.8566	0.9567	6.9613
15	鸳鸯刀	18589	241072	0.0089	13.8612	1531436	3.8637	6.0716	2019002	3.8566	0.9579	6.9636
16	鹿鼎记	615909	266983	0.2534	13.9023	1751624	3.8713	6.1906	2334204	3.8581	1.0022	7.0317

金庸小说单本信息熵表(按字)

#	小说名	语料字数	一元分词个数	一元平均词长	一元模型信息熵	二元分词个数	二元平均词长	二元模型信息熵	三元分词个数	三元平均词长	三元模型信息熵	平均信息熵
1	三十三剑客图	34704	2586	1.0	10.0117	25602	2.0	4.2789	32687	3.0	0.6488	4.9798
2	书剑恩仇录	277738	4018	0.8889	9.8743	135915	2.0	5.7552	249805	3.0	1.8221	5.8172
3	侠客行	185638	4229	0.3727	9.838	186883	2.0	6.061	382588	3.0	2.1159	6.0050
4	倚天屠龙记	509279	4528	0.5056	9.86	299128	2.0	6.4201	724008	3.0	2.5672	6.2824
5	天龙八部	622537	4843	0.3819	9.9067	414640	2.0	6.6395	1119754	3.0	2.8577	6.4680
6	射雕英雄传	483736	5194	0.2289	9.9382	496497	2.0	6.7572	1421305	3.0	3.008	6.5678
7	白马啸西风	33935	5209	0.0158	9.9408	501789	2.0	6.7645	1442694	3.0	3.0185	6.5746
8	碧血剑	260911	5334	0.1083	9.9612	545167	2.0	6.8288	1608417	3.0	3.0769	6.6223
9	神雕侠侣	523319	5385	0.1785	9.9415	608178	2.0	6.9135	1915687	3.0	3.2249	6.6933
10	笑傲江湖	495490	5415	0.1446	9.9275	661074	2.0	6.9455	2177257	3.0	3.333	6.7353
11	越女剑	8877	5418	0.0026	9.9281	662350	2.0	6.9472	2182694	3.0	3.3343	6.7365
12	连城诀	116254	5430	0.0327	9.9327	676386	2.0	6.9592	2248825	3.0	3.3569	6.7496
13	雪山飞狐	69435	5439	0.0192	9.9345	684784	2.0	6.9702	2289138	3.0	3.3686	6.7578

#	小说名	语料字数	一元分词个数	一元平均词长	一元模型信息熵	二元分词个数	二元平均词长	二元模型信息熵	三元分词个数	三元平均词长	三元模型信息熵	平均信息熵
14	飞狐外传	232335	5449	0.0603	9.9369	710278	2.0	6.9927	2417030	3.0	3.4082	6.7793
15	鸳鸯刀	18589	5449	0.0048	9.9369	712249	2.0	6.9953	2427464	3.0	3.4112	6.7811
16	鹿鼎记	615909	5554	0.1372	9.9483	781579	2.0	7.0431	2759854	3.0	3.4962	6.8292

金庸小说全集信息熵表(按词)

#	分词模型	语料字数	分词种类数	平均词长	信息熵
1	unigram model	4488686	266984	1.8469	13.9023
2	bigram model	4488686	1751629	3.8713	6.1905
3	trigram model	4488686	2334212	3.8581	1.0022

金庸小说全集信息熵表(按字)

#	分词模型	语料字数	分词种类数	平均词长	信息熵
1	unigram model	4488686	5554	1.0	9.9483
2	bigram model	4488686	781581	2.0	7.0431
3	trigram model	4488686	2759862	3.0	3.4961

结论

N-gram区别：

在分词模式下，一元词的信息熵为13.9023it，二元词的信息熵为6.1905bit，三元词的信息熵为1.0022bit，可以看出随着N的增大，平均信息熵在下降，这是因为N取值越大，通过分词后得到的文本中词组的分布就越简单，N越大使得固定的词数量越多，固定的词能减少由字或者短词打乱文章的机会，使得文章变得更加有序，减少了由字组成词和组成句的不确定性，也即减少了文本的信息熵，符合实际认知。但在分字符模式下，一元字符的信息熵为9.9483bit，二元字符的信息熵为7.0431bit，三元字符的信息熵为3.4961bit，也符合上述规律。

两种模式区别：

分词模式在一元模型下的信息熵比分字模式高。在分词模式下，一元词的数目远大于一元字的数目，这是由于不同字组合成词的方式非常多样，这就导致了词这一单位信息熵的增大。当N增大时，由于词与词之间常常有一些固定的组合搭配，会形成一些固定的意思，这就使得整体的信息熵降了下来。对于字符，虽然也有同样的趋势，但字符间的组合往往比词语的组合要更加随机，因此在二元和三元时，分字模式的信息熵要比分词模式高。

参考文档

中文信息熵的计算
中文平均信息熵
中文平均信息熵的计算