深度学习与自然语言处理第一次作业

李家哲 ZY2203105
lijiazhebuaa@buaa.edu.cn

作业要求

阅读 Entropy Of English，计算中文(分别以词和字为单位)的平均信息熵。

主要方法

M1：信息熵

信息熵的概念最早由香农（1916-2001）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：

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

论文中假设 $X=\{...,X_{-2},X_{-1},X_0,X_1,X_2,...\}$ 是基于有限字母表的平稳随机过程， P 为 X 的概率分布， E_P 为 P 的数学期望，则 X 的信息熵定义为

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

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

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

如果 P 是遍历的，则 $E_P=1$。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 $H\left(P\right)$ 的上界，与上述推理过程相同，我们可以得到以下公式：

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

$H(P,M)$ 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

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

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。

M2：统计语言模型

统计语言模型是基于预先人为收集的大规模语料数据，以真实的人类语言为标准，预测文本序列在语料库中可能出现的概率，并以此概率去判断文本是否“合法”，是否能被人所理解。

给定一个句子（词语序列）：$S=W_1,W_2,\ldots,W_k$，它的概率可以表示为：

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

但由于直接这样计算会导致参数空间过大，数据稀疏严重等问题，假设一个语料库中单词的数量为$|V|$个，一个句子由n个词组成，那么每个单词都可能有$|V|$个取值，那么由这些词组成的n元组合的数目为$|V|^n$种，也就是说，组合数会随着n的增加而呈现指数级别的增长，随着n的增加，语料数据库能够提供的数据是非常有限的，也就是说依据最大似然估计得到的概率将会是0，模型可能仅仅能够计算寥寥几个句子。所以可以引入 N-Gram 模型。在马尔可夫假设下，随意一个词出现的概率只与它前面出现的有限的一个或者几个词有关，这样我们前面得到的条件概率的计算可以简化如下

$$P\left(W_i|\ W_1,\ldots,W_{i-1}\right)\approx P\left(W_i|\ W_{i-k}\ldots W_{i-1}\right)$$

当$k=0$时对应的模型为一元模型(unigram)，即W_i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(bigram)，即W_i只与它前面的一个词相关

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

当$k=2$时对应的模型为三元模型(trigram)，即W_i只与它前面的两个词相关

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

M3：算法流程

step1：数据处理

所用的数据集为金庸先生的 16 本小说，首先对数据集进行预处理：1.删除开头的无关信息；2.删除文中的标点符号；3.删除隐藏符号（换行符、分页符等）；4.删除所有中文停词。

在分词模式下，用 Python 中的中文分词组件 Jieba 的精确分词模式对预处理后的数据集进行分词；在分字模式下，将每个字试做分的词。并求解三个统计模型下的词频。

```
    def get_unigram_tf(tf_dic, words):
    """
    获取一元词词频
    :return:一元词词频 tf_dic
    """
    for i in range(len(words)):
        tf_dic[words[i]] = tf_dic.get(words[i], 0) + 1


def get_bigram_tf(tf_dic, words):
    """
    获取二元词词频
    :return:二元词词频 tf_dic
    """
    for i in range(len(words) - 1):
        tf_dic[(words[i], words[i + 1])] = tf_dic.get((words[i],
words[i + 1]), 0) + 1


def get_trigram_tf(tf_dic, words):
    """
    获取三元词词频
    :return:三元词词频 tf_dic
```

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


def data_processing(file_path, flag):
    """
    获取文件信息并预处理
    :param file_path: 文件名对应路径
    :param flag: 选择词/字为单位，0=词，1=字
    :return data: 字符串形式的语料库
    :return words: 分词
    :return [uf_dic, bf_dic, tf_dic]: 三种模型的词频
    """
    delete_symbol = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@，。?★、…【】
《》？“”‘’！[\\]^_`{|}~「」『』（）]+'
    with open(file_path, 'r', encoding='ANSI') as f:
        data = f.read()
        data = data.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', '')
        data = re.sub(delete_symbol, '', data)
        data = data.replace('\n', '')
        data = data.replace('\u3000', '')
        f.close()

    with open('./cn_stopwords.txt', 'r', encoding='utf-8') as f:
        stopwords = []
        for a in f:
            if a != '\n':
                stopwords.append(a.strip())
    for a in stopwords:
        data = data.replace(a, '')

    words = []
    if flag == 0:
        words = list(jieba.cut(data))
    elif flag == 1:
        words = [c for c in data]
    uf_dic = {}
    bf_dic = {}
    tf_dic = {}
    get_unigram_tf(uf_dic, words)
    get_bigram_tf(bf_dic, words)
```

```
    get_trigram_tf(tf_dic, words)

    return data, words, [uf_dic, bf_dic, tf_dic]
```

---

step2：计算信息熵

利用求解得到的各模型的词频，根据 M2 节公式可分别求解信息熵。各函数输出为模型分的不同词个数、平均词长、信息熵等信息。

```
    def unigram_model(dic, sum_data):
    """
    一元模型
    :param dic: 词频
    :param sum_data: 语料库字数
    :return: [模型,一元模型分词个数,一元模型不同词个数,一元模型平均词长,一元模型信息熵,运行时间]
    """
    begin = time.time()

    unigram_sum = sum([item[1] for item in dic[0].items()])  # 一元模型分词个数
    unigram_dic = len(dic[0])  # 一元模型不同词个数
    unigram_avg = round(sum_data / float(unigram_sum), 4)  # 一元模型平均词长

    entropy = 0  # 一元模型信息熵
    for item in dic[0].items():
        entropy += -(item[1] / unigram_sum) * math.log(item[1] / unigram_sum, 2)
    entropy = round(entropy, 4)

    end = time.time()
    runtime = round(end - begin, 4)

    return ['unigram', unigram_sum, unigram_dic, unigram_avg, entropy, runtime]


def bigram_model(dic):
```

```
    """
    二元模型
    :param dic: 词频
    :return: [模型,二元模型分词个数,二元模型不同词个数,二元模型平均词长,二元模型信息熵,运行时间]
    """
    begin = time.time()

    bigram_num = sum([item[1] for item in dic[1].items()])  # 二元模型分词个数
    bigram_dic = len(dic[1])  # 二元模型不同词个数
    bigram_avg = sum(len(item[0][i]) for item in dic[1].items() for i in range(len(item[0]))) / len(dic[1])
    bigram_avg = round(bigram_avg, 4)  # 二元模型平均词长

    entropy = 0  # 二元模型信息熵
    for bi_item in dic[1].items():
        jp = bi_item[1] / bigram_num
        cp = bi_item[1] / dic[0][bi_item[0][0]]
        entropy += -jp * math.log(cp, 2)
    entropy = round(entropy, 4)

    end = time.time()
    runtime = round(end - begin, 4)

    return ['bigram', bigram_num, bigram_dic, bigram_avg, entropy, runtime]


def trigram_model(dic):
    """
    三元模型
    :param dic: 词频
    :return: [模型,三元模型分词个数,三元模型不同词个数,三元模型平均词长,三元模型信息熵,运行时间]
    """
    begin = time.time()

    trigram_num = sum([item[1] for item in dic[2].items()])  # 三元模型分词个数
    trigram_dic = len(dic[2])  # 三元模型不同词个数
    trigram_avg = sum(len(item[0][i]) for item in dic[2].items() for i in range(len(item[0]))) / len(dic[2])
    trigram_avg = round(trigram_avg, 4)  # 三元模型平均词长
```

```
    entropy = 0  # 三元模型信息熵
    for tri_item in dic[2].items():
        jp = tri_item[1] / trigram_num
        cp = tri_item[1] / dic[1][tri_item[0][0]]
        entropy += -jp * math.log(cp, 2)
    entropy = round(entropy, 4)

    end = time.time()
    runtime = round(end - begin, 4)

    return ['trigram', trigram_num, trigram_dic, trigram_avg,
entropy, runtime]
```

---

step3：输出显示

将结果输出。

```
    def information_entropy(file_path, flag):
    """
    分别计算三个模型下的信息熵
    :param file_path: 文件名对应路径
    :param flag: 选择词/字为单位，0=词，1=字
    :return: [语料库,分词,词频,一元模型结果,二元模型结果,三元模型结果,平均信息
熵]
    """
    begin = time.time()
    data, words, dic = data_processing(file_path, flag)  # 数据处理
    end = time.time()
    runtime = round(end - begin, 4)

    # 以下三个值应与一元分词模型结果相同
    sum_data = len(data)  # 语料库字数
    sum_words = len(words)  # jieba 分词/分字 个数
    avg_word = round(sum_data / sum_words, 4)  # jieba 分词/分字 平均词长

    print('语料库字数：', sum_data)
    if flag == 0:
        print('jieba 分词个数：', sum_words)
```

```
    elif flag == 1:
        print('按字分词个数：', sum_words)
    print('数据处理时间(s)：', runtime)
    print('                模型分词个数 | 平均词长 | 信息熵 | 运行时间(s)')
    # unigram 一元模型
    unigram = unigram_model(dic, sum_data)
    print('*unigram 一元模型：', unigram[2], ' | ', unigram[3], ' | ',
unigram[4], ' | ', unigram[5])

    # bigram 二元模型
    bigram = bigram_model(dic)
    print('*bigram 二元模型：', bigram[2], ' | ', bigram[3], ' | ',
bigram[4], ' | ', bigram[5])

    # trigram 三元模型
    trigram = trigram_model(dic)
    print('*trigram 三元模型：', trigram[2], ' | ', trigram[3], ' | ',
trigram[4], ' | ', trigram[5])

    avg_entropy = np.mean([unigram[4], bigram[4], trigram[4]])  # 平均
信息熵
    print('平均信息熵：  %.4f' % avg_entropy)

    return [data, words, dic, unigram, bigram, trigram, avg_entropy]
```

实验结果

信息熵实验结果如表 1 和表 2 所示，完整结果详见“record_230325_删除停词.txt”。

表 1 分词模式下信息熵

	语料字数	unigram 一元模型 信息熵(bit/词)	bigram 二元模型 信息熵(bit/词)	trigram 三元模型 信息熵(bit/词)	平均信息熵
白马啸西风	33901	11.1888	2.7008	0.266	4.7185
碧血剑	260877	12.9338	3.733	0.3833	5.6834
飞狐外传	232335	12.7394	3.7634	0.3813	5.6280
连城诀	116254	12.2958	3.3131	0.3003	5.3031
鹿鼎记	615875	12.9088	4.6814	0.6597	6.0833
三十三剑客图	34670	12.4441	1.6571	0.0693	4.7235
射雕英雄传	483702	13.1385	4.3284	0.4664	5.9778
神雕侠侣	523285	12.5768	4.7826	0.7308	6.0301

书剑恩仇录	277704	12.7835	3.9215	0.4311	5.7120
天龙八部	622503	13.223	4.505	0.5581	6.0954
侠客行	185604	12.3915	3.7073	0.4466	5.5151
笑傲江湖	495456	12.629	4.5675	0.7275	5.9747
雪山飞狐	69401	12.1486	2.7643	0.2293	5.0474
倚天屠龙记	509245	13.0212	4.4135	0.5642	5.9996
鸳鸯刀	18555	10.9819	2.1031	0.1885	4.4245
越女剑	8843	10.2596	1.7207	0.2311	4.0705

表 2 分字模式下信息熵

	语料字数	unigram 一元模型 信息熵(bit/词)	bigram 二元模型 信息熵(bit/词)	trigram 三元模型 信息熵(bit/词)	平均信息熵
白马啸西风	33901	9.2225	4.082	1.2102	4.8382
碧血剑	260877	9.7606	5.6686	1.7886	5.7393
飞狐外传	232335	9.6278	5.5684	1.8619	5.6860
连城诀	116254	9.5209	5.0824	1.635	5.4128
鹿鼎记	615875	9.6623	6.016	2.4024	6.0269
三十三剑客图	34670	10.0116	4.2777	0.6492	4.9795
射雕英雄传	483702	9.7522	5.9655	2.1919	5.9699
神雕侠侣	523285	9.5703	6.1215	2.3973	6.0297
书剑恩仇录	277704	9.759	5.5959	1.8576	5.7375
天龙八部	622503	9.787	6.1108	2.3457	6.0812
侠客行	185604	9.4393	5.3722	1.8144	5.5420
笑傲江湖	495456	9.5141	5.8553	2.3574	5.9089
雪山飞狐	69401	9.504	4.7952	1.2992	5.1995
倚天屠龙记	509245	9.7062	5.9817	2.2729	2.2729
鸳鸯刀	18555	9.2112	3.6512	0.8952	4.5859
越女剑	8843	8.7828	3.1053	0.8433	4.2438

结论

1. 对于 N-gram 模型来说，随着 N 的增大，信息熵在下降。这是因为 N 的增大使得通过分词后得到的文本中词组的分布变得简单，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。

2. 对于两种模式来说，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。

参考文献

[1] Brown P F, Della Pietra S A, Della Pietra V J, et al. An estimate of an upper bound for the entropy of English[J]. Computational Linguistics, 1992, 18(1): 31-40.