深度学习与自然语言处理第一次作业报告

吴宣余

ZY2303121

wxy7334@buaa.edu.cn

一、问题描述

对给定的16篇金庸武侠小说语料库，验证Zipf's Law；阅读Entropy of English这篇文章，学习并理解信息熵的概念，分别以字和词为单位，利用一元、二元以及三元模型进行中文信息熵的计算。

二、信息熵

1948年，香农提出了"信息熵"的概念，解决了对信息的量化度量问题。信息熵常被用来作为一个系统的信息含量的量化指标，从而可以进一步用来作为系统方程优化的目标或者参数选择的判据。信息熵的定义公式为：

$$H(x) = -\sum p(x) lgp(x)$$

其中$p(x)$为每种对应信源发生的概率分布。

信息熵的性质有：

单调性：发生概率越高的事件，其携带的信息量越低；

非负性：收到一个信源所获得的信息量应为正值；

累加性：多事件同时发生存在的总不确定性可以表示为各事件不确定性的和；

极大值性：当概率分布的所有可能事件均等概率出现时，信息熵达到最大值。满足极大熵的概率分布是均匀分布；

条件熵特性：条件熵衡量已知某个随机变量的取值情况下，另一个随机变量的不确定性或信息量的平均量。条件熵的值始终大于等于零，并且在两个随机变量相互独立时达到最小值。

三、N-gram语言模型

N-gram是一种输入为一个句子，输出为该句子的概率的语言模型。N-gram模型基于"第N个词的出现只与前面N-1个词相关，而与其它词都无关"的假设。

假设S表示某个有意义的句子，其由一连串特定顺序排列的词$\omega_1, \omega_2, \dots, \omega_n$组成，n为句子长度。则S在文本中出现的可能性为：

$$p(s) = p(\omega_1, \omega_2, \dots, \omega_n)$$

利用条件概率公式得到：

$$p(\omega_1, \omega_2, \dots, \omega_n) = p(\omega_1)p(\omega_1|\omega_2)\dots p(\omega_n|\omega_1, \omega_2, \dots, \omega_n)$$

当N=1时，就是一元模型，N=2就是二元模型，以此类推。N元模型的数学表达如下所示：

$$P(\omega_i|\omega_{i-n+1}^{i-1}) = P(\omega_i|\omega_{i-n+1}, \dots, \omega_{i-1})$$

一元、二元和三元模型分别表示的含义为：N=1表示与前面单词都没有关系；N=2表示与前面一个单词有关；N=3表示与前面两个词有关。

四、实验

1. 数据预处理

本次给定的16篇金庸武侠小说语料库皆包含无用的广告信息以及符号。首先需要对其进行预处理：

(1) 删除换行符、分页符等隐藏符号:

(2) 删除标点符号

(3) 删除广告信息

(4) 删除阿拉伯数字与英文字母，仅保留中文

```
def preprocess(root_dir):
    corpus = []
    for file in os.listdir(root_dir):
        if file [-3:] == 'txt':
            path = os.path.join(root_dir, file)
            with open(path, 'r', encoding='ANSI') as f:
                text = [line.strip("\n").replace("\u3000", "").replace("\t",
"") for line in f][3:]
                corpus += text
    pattern = r'[^\u4E00-\u9FA5]'
    regex = re.compile(pattern)
    replacements = ["\t", "\n", "\u3000", "\u0020", "\u00A0", " "]
    for j in range(len(corpus)):
        corpus[j] = rid_of_ad(corpus[j])  # 去除广告
        corpus[j] = re.sub(regex, "", corpus[j])  # 只保留中文
        for replacement in replacements:
            corpus[j] = corpus[j].replace(replacement, "") # 去除换行符、分页符
等符号
    corpus = [x for x in corpus if x != ""] # 去除空字符串
    return corpus
```

```
def rid_of_ad(content):
    ad = ['本书来自 www.cr173.com 免费 txt 小说下载站', '更多更新免费电子书请关注
www.cr173.com', '新语丝电子文库']
    for ads in ad:
        content = content.replace(ads, '')
    return content
```

2. 统计字频和词频

为了验证Zipf's Law以及计算中文信息熵，需要对语料库进行字频和词频的统计。字频统计直接依次读取计算即可，词频统计需要采用jieba库先对语料库进行分词操作。

```
def cha_fre(file,n):
    adict = {}
    if n == 1:
        for line in file:
            for i in line:
                if i in adict:
                    adict[i] += 1
                else:adict[i] = 1
    elif n == 2:
        for line in file:
            for i in range(len(line)-1):
                if (line[i]+line[i+1]) in adict:
                    adict[line[i]+line[i+1]] += 1
                else:adict[line[i]+line[i+1]] = 1
    else:
        for line in file:
            for i in range(len(line)-2):
                if (line[i]+line[i+1]+line[i+2]) in adict:
                    adict[line[i]+line[i+1]+line[i+2]] += 1
                else:adict[line[i]+line[i+1]+line[i+2]] = 1
    return adict
```

```
def word_fre(file,n):
    adict = {}
    if n == 1:
        for line in file:
            words = list(jieba.cut(line))
            for i in range(len(words)):
                if tuple(words[i:i+1]) in adict:
                    adict[tuple(words[i:i+1])] += 1
                else:adict[tuple(words[i:i+1])] = 1
    elif n == 2:
        for line in file:
            words = list(jieba.cut(line))
            for i in range(len(words)-1):
                if tuple(words[i:i+2]) in adict:
                    adict[tuple(words[i:i+2])] += 1
                else:adict[tuple(words[i:i+2])] = 1
    else:
        for line in file:
            words = list(jieba.cut(line))
            for i in range(len(words)-2):
                if tuple(words[i:i+3]) in adict:
                    adict[tuple(words[i:i+3])] += 1
                else:adict[tuple(words[i:i+3])] = 1
    return adict
```

字频统计的结果如下所示（以白马啸西风.txt为例）：

一元：{'白': 128, '马': 231, '啸': 4, '西': 54, '风': 72, '得': 369, '在': 582, '黄': 31, '沙': 78, '莽': 6, '的':1561, '回': 139, '疆': 27, '大': 397, '漠': 54, '之': 290, '上': 429, '尘': 4, '飞': 31, '起': 225, '两': 188, '丈': 24, '来': 571, '高': 79, '骑': 24, '一': 1315, '前': 100, '後': 142}

二元：{'白马': 76, '马啸': 1, '啸西': 1, '西风': 2, '风得': 1, '得得': 10, '得在': 2, '在黄': 1, '黄沙': 12,'沙莽': 2, '莽莽': 3, '莽的': 1, '的回': 3, '回疆': 27, '疆大': 1, '大漠': 17, '漠之': 9, '之上': 11, '尘沙':2, '沙飞': 4, '飞起': 3, '起两': 1, '两丈': 1, '丈来': 1, '来高': 1, '两骑': 1, '骑马': 6}

三元：{'白马啸': 1, '马啸西': 1, '啸西风': 1, '西风得': 1, '风得得': 1, '得得得': 6, '得得在': 1, '得在黄': 1, '在黄沙': 1, '黄沙莽': 2, '沙莽莽': 2, '莽莽的': 1, '莽的回': 1, '的回疆': 1, '回疆大': 1, '疆大漠': 1, '大漠之': 6, '漠之上': 3, '尘沙飞': 2, '沙飞起': 1}

词频统计的结果如下所示（以白马啸西风.txt为例）：

一元：{('白马',): 75, ('啸',): 1, ('西风',): 2, ('得',): 168, ('在',): 494, ('黄沙',): 12, ('莽莽',): 3, ('的',):1484, ('回疆',): 22, ('大漠',): 17, ('之上',): 11, ('尘沙',): 2, ('飞',): 9, ('起',): 18, ('两丈',): 1, ('来',):145, ('高',): 9, ('两',): 7, ('骑马',): 5, ('一前',): 1, ('一',): 96, ('後',): 102, ('急驰',): 1}

二元：{('白马', '啸'): 1, ('啸', '西风'): 1, ('西风', '得'): 1, ('得', '得'): 10, ('得', '在'): 1, ('在', '黄沙'): 1,('黄沙', '莽莽'): 2, ('莽莽', '的'): 1, ('的', '回疆'): 1, ('回疆', '大漠'): 1, ('大漠', '之上'): 3, ('尘沙', '飞'): 1, ('飞', '起'): 1, ('起', '两丈'): 1, ('两丈', '来'): 1, ('来', '高'): 1}

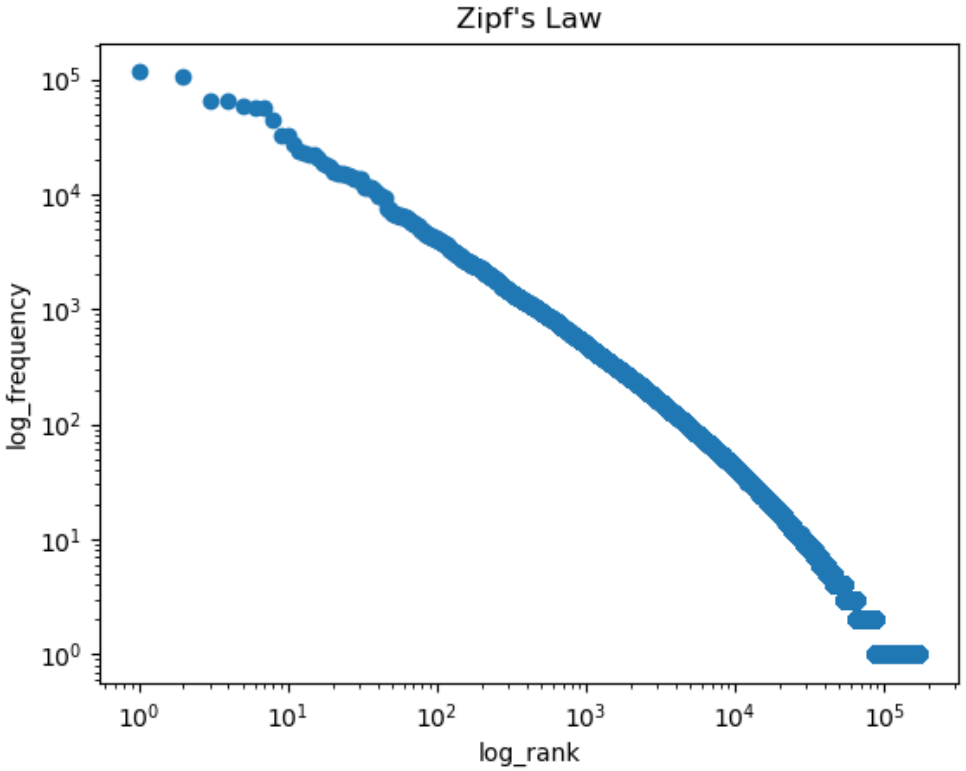
三元：{('白马', '啸', '西风'): 1, ('啸', '西风', '得'): 1, ('西风', '得', '得'): 1, ('得', '得', '得'): 6, ('得', '得', '在'): 1, ('得', '在', '黄沙'): 1, ('在', '黄沙', '莽莽'): 1, ('黄沙', '莽莽', '的'): 1, ('莽莽', '的', '回疆'):1, ('的', '回疆', '大漠'): 1, ('回疆', '大漠', '之上'): 1, ('尘沙', '飞', '起'): 1}

3. 验证Zipf’s Law

根据词频统计结果，绘制rank-fre曲线，验证Zipf’s Law。

```
def plot_zipfs_law(ranks, frequencies):
    # 绘制 Log-log 图
    plt.scatter(ranks, frequencies)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('log_rank')
    plt.ylabel('log_frequency')
    plt.title("Zipf's Law")
    plt.show()
```

曲线如下图所示，符合Zipf’s Law的规律：



4. 计算中文信息熵

根据第二节提到的信息熵公式分别计算基于字和词为单位的中文信息熵：

```
def cal_cha_entropy(file,n):
    if n == 1:
        frequency = cha_fre(file,1)
        sums = np.sum(list(frequency.values()))
        entropy = -np.sum([i*np.log2(i/sums) for i in
frequency.values()])/sums
    elif n == 2:
        frequency1 = cha_fre(file,1)
        frequency2 = cha_fre(file,2)
        sums = np.sum(list(frequency2.values()))
        entropy = -np.sum([v*np.log2(v/frequency1[k[:n-1]]) for k,v in
frequency2.items()])/sums
    else:
        frequency2 = cha_fre(file,2)
        frequency3 = cha_fre(file,3)
        sums = np.sum(list(frequency3.values()))
        entropy = -np.sum([v*np.log2(v/frequency2[k[:n-1]]) for k,v in
frequency3.items()])/sums
    return entropy
```

```
def cal_word_entropy(file,n):
    if n == 1:
        frequency = word_fre(file,1)
        sums = np.sum(list(frequency.values()))
        entropy      =      -np.sum([i*np.log2(i/sums)      for      i      in
frequency.values()])/sums
    elif n == 2:
        frequency1 = word_fre(file,1)
        frequency2 = word_fre(file,2)
        sums = np.sum(list(frequency2.values()))
        entropy  =  -np.sum([v*np.log2(v/frequency1[k[:n-1]])  for  k,v  in
frequency2.items()])/sums
    else:
        frequency2 = word_fre(file,2)
        frequency3 = word_fre(file,3)
        sums = np.sum(list(frequency3.values()))
        entropy  =  -np.sum([v*np.log2(v/frequency2[k[:n-1]])  for  k,v  in
frequency3.items()])/sums
    return entropy
```

计算结果如下表所示：

模型	以字为单位信息熵	以词为单位信息熵
1-gram	9.526977532562082	12.179999848519683
2-gram	6.718079939408664	6.952804600537577
3-gram	3.9513982417660998	2.305610878602338

五、结论

由计算结果可知，随着N的取值变大，文本的信息熵逐渐变小。分析其原因是：N值的增加意味着文本经过分词处理后捕获的文章上下文信息更为准确，从而减少了由字或短词打乱文章语义的机会，使文章被更加有序地组织起来，减少了独立字符带来的不确定性，也就减小了文本的信息熵。

以词为单位采用N-gram模型计算的信息熵在N比较小的时候值较高，而N较大时值较低，并且低于以字为单位的信息熵。猜测其原因是：在使用1-gram模型和2-gram模型的时候考虑其前后上下文较少，无意义助词的重复率高，这种词语的出现打乱了文章的秩序，从而使得信息熵增大；而在使用3-gram模型的时候基本考虑了语法结构，更好地考虑了上下文，因此得到了最低的熵值。