深度学习与自然语言处理第一次作业

（中文平均信息熵的计算）

姓名：刘千歌

学号：BY2139121

一、 实验目标

阅读参考文献 *Entropy of English PeterBrown*，计算课程提供中文文献的平均信息熵。

二、实验原理

2.1 信息熵

香农（1916-2001）于 1948 年为解决信息量化的问题，从热力学中借用了热熵的概念提出了“信息熵”。所谓信息熵，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。对于一个随机变量 X，其信息熵数学公式可以表示为：

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

对于联合分布随机变量$(X,Y)\sim P(X,Y)$在两变量相互独立的情况下，其联合信息熵的数学公式表示为如下。后文的二元模型（Bi-Gram）与三元模型（Tri-Gram）将应用到该公式。

$$\begin{aligned}H(X|Y)&=-\sum_{y\in Y}P(y)log(P(x|y))\\&=-\sum_{y\in Y}P(y)\sum_{x\in X}P(x)log(P(x|y))\\&=-\sum_{y\in Y}\sum_{x\in X}P(x)P(y)log(P(x|y))\\&=-\sum_{y\in Y}\sum_{x\in X}P(x,y)log(P(x|y))\end{aligned}$$

、

2.2 N-gram 语言模型

N-gram 模型是一种语言模型（Language Model，LM），它的输入是一句话（单词的顺序序列），输出是这句话的概率，即这些单词的联合概率（joint probability）。

N-gram 的实现是基于“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。该模型首先将文本内容以字节为单位进行大小为 N 的滑动窗口操作，每一个 N 长度的字节片段称为 gram；进一步，对所有 gram 的出现频度进行统计，并且按照事先设定好的阈值进行过滤，形成关键 gram 列表（即，文本的向量特征空间），而列表中的每一种 gram

就是一个特征向量维度；最后，基于特征向量维度的频次统计获取概率，输出句中各个词出现概率的乘积获取整个句子的概率，其公式可表示为：

$$P(w_1, w_2, \cdots, w_m) = \prod_{i=1}^{m} P(w_i)$$

N-gram 常用的是二元的 Bi-Gram 模型和三元的 Tri-Gram 模型。在二元模型的应用场景下，每个词出现的概率只与它的前一个词有关；而三元模型的应用场景下，与前两个词有关。

三、实验步骤

3.1 语料库预处理

将本次作业提供的语料库放置于代码根目录下的 Dataset 文件夹下。其中的每一个文本皆包含无用的广告信息以及无用的符号。首先做以下预处理：

1）删除隐藏符号（如换行符、分页符等）

2）删除爬虫得到的无关信息与字符。

3）删除文中的标点符号，因为标点符号对于中文语料库的信息并无意义。

4）广告信息，例如文中出现的“免费电子书请关注..”等。

```
# corpus 用来存储处理过的语料库
corpus = []
#root_path 为当前代码目录下的 dataset 文件夹路径，其中存放中文语料库
for i in os.listdir(root_path):
#判定所选文件以 txt 结尾
    if i[-3:] == 'txt':
    path = root_path + r'/' + i
    # 使用的设备是 Mac M1，由此 ANSI 编码方式对我不适用，代码使用的是 gb18030
    with open(path, 'r', encoding='gb18030') as f:
         text = [line.strip("\n").replace("\u3000",\
                 "").replace("\t", "") for line in f][3:]
         corpus += text
#利用正则匹配，消除无关字符
regex_str = ".*?([^\u4E00-\u9FA5]).*?"
english = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;「<=>?@,。?★、…【】《》？
“”‘’！[\\]^_`{|}~]+'
symbol = []
for j in range(len(corpus)):
    corpus[j] = re.sub(english, "", corpus[j])
```

```
    symbol += re.findall(regex_str, corpus[j])

#对出现的字符进行匹配与统计
count_ = Counter(symbol)
count_symbol = count_.most_common()
noise_symbol = []

for eve_tuple in count_symbol:
    #统计次数小于 200 次的字符
    if eve_tuple[1] < 200:
        noise_symbol.append(eve_tuple[0])

#定义待消除的广告
ad = ['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', '新语丝电子文库']

#将处理过的语料库存进本地代码目录下的 processed_content.txt 文件中
with open("processed_content.txt", "a", encoding="utf-8") as f:
    for line in corpus:
        #消除噪声符号
        for noise in noise_symbol:
               line.replace(noise, "")
        #消除广告
        for a in ad:
               line.replace(a, "")
        f.write(line)
```

3.2 分词处理与词语频统计

3.2.1 以字为单位

在以字为单位时无需使用分词工具，即直接以字符为单位依次读取语料库。

1） 1-gram

```
#1-gram 时每个字在句中出现的概率视作独立

#corpus 即为处理后的语料库，遍历每个段落，获取独立字符
for para in corpus:
    for i in range(len(para)):
        chars += para[i]
#获取字库中总字数 char_num
char_num = len(chars)
ct = Counter(chars)
#获取字频表
vocab1 = ct.most_common()
```

1-gram 分词结果：字库总字数 7299088，不同字的个数为 5782

出现频率前 10 的 1-gram 词语：（'一', 139397), ('不', 134150), ('的', 121671), ('是', 112708), ('了', 111927), ('道', 111057), ('人', 84305), (', 68993), ('我', 67000)

2）2-gram

```
#2-gram 时每个字在句中出现的概率之与它前一个字有关

# 获取 2-gram 词表函数
def combine2gram(cutword_list):
    if len(cutword_list) == 1:
        return []
    res = []
    for i in range(len(cutword_list) - 1):
        res.append(cutword_list[i] + cutword_list[i + 1])
return res

#char_2gram 用于存储 2-gram 的词表
char_2gram = []
for para in corpus:
    cutword_list = []
    for i in range(len(para)):
        cutword_list += para[i]
        char_2gram += combine2gram(cutword_list)
#获取字库中总词数 char_num
char_2gram_num = len(char_2gram)
ct2 = Counter(char_2gram)
#获取字频表
vocab2 = ct2.most_common()
```

2-gram 分词结果：字库总字数 7239476，不同字的个数为 731515

出现频率前 10 的 2-gram 词语： ('说道', 13525), ('了一', 12174), ('一个', 10571), ('自己', 10319), ('道你', 10262), ('小宝', 9942), ('韦小', 9856), ('也不', 9304), ('道我', 8472), ('笑道', 8140)。

3）3-gram

```
#3-gram 时认为每个字出现的概率只与它的前两个字有关
# 获取 3-gram 词表函数
def combine3gram(cutword_list):
    if len(cutword_list) <= 2:
        return []
    res = []
    for i in range(len(cutword_list) - 2):
        res.append(cutword_list[i] + cutword_list[i + 1] + cutword_list[i + 2])
    return res

#其他代码同理，在此省略
```

3-gram 分词结果：字库总字数 7180160，不同字的个数为 3173243

出现频率前 10 的 3-gram 词语： ('韦小宝', 9803), ('令狐冲', 5889), ('张无忌', 4645), ('的一声', 3478), ('袁承志', 3037), ('小宝道', 2417), ('陈家洛', 2115), ('小龙女', 2081), ('石破天', 1818), ('不由得', 1803)

3.2.2 以词为单位

第二种情况，即以词为单位计算信息熵时，使用 Python 提供的中文分词组件 jieba 工具进行分词。其获取词频表的方式与以字为单位进行切分相同，本报告中即以 2-gram 的情况为代表陈列示例代码。

```
#2-gram 时认为每个词出现的概率只与它的前两个词有关
# 获取 2-gram 词表函数
def combine2gram(cutword_list):
    if len(cutword_list) == 1:
        return []
    res = []
    for i in range(len(cutword_list) - 1):
        #这里添加"s"是为了后面计算的时候方便切分
        res.append(cutword_list[i] + "s" + cutword_list[i + 1])
        return res

#token_2gram 用于存储 2-gram 的词表
token_2gram = []
for para in corpus:
    #此处不再以字符为单位读取，使用 jieba 分词后以词为单位读取
    cutword_list = jieba.lcut(para)
    token_2gram += combine2gram(cutword_list)
```

```
# 2-gram 的频率统计
token_2gram_num = len(token_2gram)
ct2 = Counter(token_2gram)
vocab2 = ct2.most_common()
```

1-gram 分词结果：词库总词数 4314285，不同词的个数为 173004

出现频率前 10 的 1-gram 词语： ('的', 115607), ('了', 104538), ('他', 64714), ('是', 64458), ('道', 58625), ('我', 57483), ('你', 56680), ('在', 43692), ('也', 32606), ('这', 32199)

2-gram 分词结果：词库总词数 4254673，不同词的个数为 1950581

出现频率前 10 的 2-gram 词语： ('道你', 5738), ('叫道', 5009), ('道我', 4953), ('笑道', 4271), ('听得', 4202), ('都是', 3905), ('了他', 3638), ('他的', 3497), ('也是', 3201), ('的一声', 3102)

3-gram 分词结果：词库总词数 4195606，不同词的个数为 3482115

出现频率前 10 的 3-gram 词语： ('只听得', 1611), ('忽听得', 1137), ('站起身来', 733), ('哼了一声', 573), ('笑道你', 566), ('吃了一惊', 534), ('s ', 513), ('点了点头', 503), ('啊的一声', 481) ,('说到 s 这里', 477), ('了他的', 454)

3.3 信息熵计算

在依照两种情况处理好文本后，即可按照 1-gram， 2-gram， 3-gram 三种语言模型计算中文的信息熵，其中，2,3-gram 使用条件熵计算。因为以字为单位与以词为单位的处理方式是一致的，由此，我仅将以字为单位的三种情况处理代码陈列出当作示例。两种情况下所得的信息熵，将在第四章《计算结果》中完整陈列。

在 1-gram 情况下，参考信息熵计算公式。在该场景下，*P(x)*近似等于每个词在语料库中出现的频率（即词出现次数处以总词数），计算信息熵的代码如下：

```
#1-gram
#vocab1 为 1-gram 的词表
entropy_1gram = sum([-(eve[1] / char_num) * math.log((eve[1]
/ char_num), 2) for eve in vocab1])
```

在 2-gram 情况下，中联合概率 *P(x,y)*可近似等于每个二元词组在语料库中出现的频

率，条件概率 $P(x|y)$可近似等于每个二元词组在语料库中出现的频数与以该二元词组的第一个词为词首的二元词组的频数的比值。

```
# 2-gram 相同句首的频率统计
same_1st_word = [eve.split("s")[0] for eve in token_2gram]
assert token_2gram_num == len(same_1st_word)
ct_1st = Counter(same_1st_word)
vocab_1st = dict(ct_1st.most_common())

#vocab2 为 2-gram 词表
entropy_2gram = 0
for eve in vocab2:
    p_xy = eve[1] / token_2gram_num
    #联合概率等于每两个二元次组在语料库中出现的频率
    first_word = eve[0].split("s")[0]
    #统计以每个二元词组第一个词为词首的二元词组数目
    entropy_2gram    +=    -p_xy    *    math.log(eve[1]    /
vocab_1st[first_word], 2)
    #利用联合熵公式计算熵值
```

在 3-gram 情况下，联合概率 $P(x,y,z)$可近似等于每个三元词组在语料库中出现的频率，条件概率 $P(x|y,z)$可近似等于每个三元词组在语料库中出现的频数与以该三元词组的前两个词为词首的三元词组的频数的比值。代码同上

```
# 3-gram 相同句首的频率统计
same_2st_word = [eve.split("s")[0] for eve in token_3gram]
assert token_3gram_num == len(same_2st_word)
ct_2st = Counter(same_2st_word)
vocab_2st = dict(ct_2st.most_common())

#vocab3 为 3-gram 词表
entropy_3gram = 0
for eve in vocab3:
    p_xyz = eve[1] / token_3gram_num
    #联合概率等于每个三元词组在语料库中出现的频率
    first_2word = eve[0].split("s")[0]
    #统计以每个三元词组第一个词为词首的三元词组数目
    entropy_3gram    +=    -p_xyz    *    math.log(eve[1]    /
vocab_2st[first_2word], 2)
    #利用联合熵公式计算熵值
```

四、计算结果

表格 1 以字为单位计算信息熵

模型	语料库字数	分词个数	平均信息熵
1-gram	7299088	5782	9.53812895254137
2-gram	7239476	731515	6.7159971584519305
3-gram	7180160	3173243	3.9385939455361054

表格 2 以词为单位计算信息熵

模型	语料库字数	分词个数	平均信息熵
1-gram	4314285	173004	12.166366977810295
2-gram	4254673	1950581	6.945688123876337
3-gram	4192870	3481235	2.30248478214879

五、结论

在分词方面：

- 若单独考虑 N 取值带来的影响

根据上方陈列结果可知，当 N 值取得越大分词出现的个数越多。其原因是，N 的增大意味着前后文获取长度的增大，这为不同字组合成不同分词提供更大的可能性。

- 若单独考虑以字为单位或以词为单位带来的影响

由上方结果总体看来，以词为单位进行分词会获取的分词数目更多。其原因是，在用 jieba 进行中文分词时不限于单字，即较单字捕获的前后文长度更大，同理，为组成不同分词组合带来更大的可能性。

在**熵值**方面：

- 若单独考虑 N 取值带来的影响

由计算结果可知，随着 N 取值变大文本的信息熵越小。其原因是，N 值的增加意味着文本经过分词处理后得到的 gram 列表（文本的向量特征空间）分布就越为简单——前后文取得越长则捕获的文章上下文信息更为准确（或者说，固定的词汇组合越多），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，即减小了文本的熵。基于以上讨论内容，该现象符合实际认知。

- 若单独考虑分词或者分字带来的影响

由上方结果总体看来，以词为单位采用 1-gram 模型计算获取的信息熵值最大；我猜测其原因是：使用 jieba 划分词组时，往往把助词（例如“的”）等单独划分出来，在使用

1-gram 模型的时候不考虑其前后动名词上下文（且前后有意义的动、名词组成有意义的词汇后重复率低），无意义助词的重复率高，使无意义助词变成文本向量特征空间中的高频词语，在不考虑上下文时，这种词语的出现打乱了文章的秩序，从而使得信息熵增大；在 2-gram 的时候依然是类似的问题，但是因为考虑了部分上下文，使得以词为单位划分的熵值只是略高于以字为单位划分的；最后，在 3-gram 的时候等于基本考虑了语法结构，两种划分方式获得的分词数目差不多，且以词为单位的划分方式更好的考虑了语法结构上下文，由此获取了最低的熵值。

参考资料

[1] 深度学习与自然语言处理：中文信息熵的计算 CSDN，
https://blog.csdn.net/qq_40412713/article/details/115742092