计算中文信息熵

郭润堂

guoruntang@buaa.edu.cn

摘要

本文为深度学习与自然语言处理课程的第一次大作业，通过阅读《An Estimate of an Upper Bound for the Entropy of English》，借鉴英文信息熵的方法对中文信息熵进行计算。使用语料为 16 本金庸武侠小说，去除其中中文停用词、特殊符号及空格，基于 jieba 分词，分别建立 1-Gram、2-Gram、3-Gram 模型。计算得出 1-Gram、2-Gram、3-Gram 模型的信息熵为 13.93153707433563、6.1361563015769915、0.9814784939104394。

简介

在自然语言处理领域，信息熵是评估语言模型的一种重要指标。它是从信息论的角度对自然语言进行建模和分析的方法，表示随机变量的不确定性或信息量。信息熵主要用于量化语言的规律性和不确定度，反映了单个字、词在语料库中出现的不确定性，即它们的出现频率对于预测下一个字、词的概率的贡献。在语言模型中，信息熵越小，则表示这些字、词出现的频率越集中，语言的规律性就越强，反之则越不确定。利用信息熵的概念，可以进行语言模型的优化、词法分析和文本分类等自然语言处理任务。

下面依次对信息熵、文本信息熵、N-Gram 语言模型等进行介绍。

信息熵的公式为：

$$H(x) = -\Sigma(p(i) * \log2\ p(i))$$

其中，H(x)表示随机变量 X 的信息熵，p(i)表示事件 i 发生的概率，log2 p(i)是以 2 为底的对数函数，用于表示事件 i 的信息量大小。Σ表示对所有事件 i 的概率乘以对应的信息量进行求和。

在文本信息熵中，我们可以把每个字符或者词作为一个随机变量，因此可将文本表示为：

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

它的信息熵表示为：

$$H(X) \equiv H(P) \equiv -E_P logP(X_0|X_{-1}, X_{-2}, \dots)$$

根据 Shannon-McMillan-Breiman 定理可推导出 H (X)的上界为

$$H(P) \leq \lim_{n\to\infty} - \log M(X_1\ X_2 \dots X_n)/n$$

其中 M 代表其中词概率 P 的一种模型。

N-Gram 语言模型：在对不同语言信息熵分析的过程中，由于数据稀疏和系统处理能力的限制，N-Gram 语言模型引入了马尔科夫假设，即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。简单来说，只考虑长度为 n-1 的历史窗口。因此 1-Gram 模型仅考虑一个独立的词出现的概率来进行计算；2-Gram 概率模型考虑在前一个词出现的基础之上，当前词出现的概率；3-Gram 概率模型考虑在前两个词出现的基础之上，当前词出现的概率。当 N 大于 1 时，N-Gram 语言模型的概率模型使用条件概率进行表示。此时信息熵由条件信息熵进行表示，并根据极大似然估计准则进行计算。

方法

1.首先对 txt 文件列表进行读取

```
def Read_file_list(dict_name):
    stack = []
    result_txt = []
    stack.append(dict_name)
    while len(stack) != 0:
        temp_name = stack.pop()
        try:
            temp_name2 = os.listdir(temp_name)
            for eve in temp_name2:
                stack.append(temp_name + "\\" + eve)
        except :
            result_txt.append(temp_name)
    return result_txt
```

2. 按段落对 txt 文件进行读取，去除空格、tab 等字符

```
path_list = Read_file_list(r".\txt")
corpus = []
for path in path_list:
    with open(path, "r", encoding="ANSI") as file:
        text = [line.strip("\n").replace("\u3000", "").replace("\t", "").replace(" ", "") for line in file]
        corpus += text
```

3. 读取停词 txt 文件，对停词进行剔除

```
with open("cn_stopwords.txt", "r", encoding="utf-8") as f:
    lines = f.readlines()
lines = [line.strip('\n') for line in lines]
for j in range(len(corpus)):
    for line in lines:
        corpus[j]=corpus[j].replace(line, "")
```

4. 对除了中文字符以外的其它特殊符号进行剔除

```
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])
count_ = Counter(symbol)
count_symbol = count_.most_common()
noise_symbol = []
for eve_tuple in count_symbol:
    if eve_tuple[1] < 200:
        noise_symbol.append(eve_tuple[0])
noise_number = 0
for line in corpus:
    for noise in noise_symbol:
        line=line.replace(noise, "")
        noise_number += 1
print("完成的噪声数据替换点：", noise_number)
```

5. 使用 jieba.lcut()对语料库中的每一行语料进行精细分词。使用 1-Gram 语言模型公式进行计算。

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

```
token = []
for para in corpus:
    token += jieba.lcut(para)
token_num = len(token)
ct = Counter(token)
vocab1 = ct.most_common()
entropy_1gram = sum([-(eve[1]/token_num)*math.log((eve[1]/token_num),2) for eve in vocab1])
print("1-Gram 词库总词数：", token_num, " ", "不同词的个数：", len(vocab1))
print("出现频率前10的1-gram词语：", vocab1[:10])
print("entropy_1gram:", entropy_1gram)
```

6. 将切词后的列表组合成双词列表，中间使用空格进行连接。

```
for para in corpus:
    cutword_list = jieba.lcut(para)
    token_2gram += combine2gram(cutword_list)
```

```
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
```

7. 使用 2-Gram 语言模型公式对信息熵进行计算

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

```
token_2gram = []
for para in corpus:
    cutword_list = jieba.lcut(para)
    token_2gram += combine2gram(cutword_list)
token_2gram_num = len(token_2gram)
ct2 = Counter(token_2gram)
vocab2 = ct2.most_common()
same_1st_word = [eve.split(" ")[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())
entropy_2gram = 0
for eve in vocab2:
    p_xy = eve[1]/token_2gram_num
    first_word = eve[0].split(" ")[0]
    entropy_2gram += -p_xy*math.log(eve[1]/vocab_1st[first_word], 2)
print("2-Gram 词库总词数：", token_2gram_num, " ", "不同词的个数：", len(vocab2))
print("出现频率前10的2-gram词语：", vocab2[:10])
print("entropy_2gram:", entropy_2gram)
```

8. 将切词后的列表组合成三词列表，中间使用空格对一二词和三词之前进行连接。

```
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
```

9. 使用 3-Gram 语言模型公式对信息熵进行计算。

$$H(X|Y,Z) = -\sum_{x\in X,y\in Y,z\in Z} P(x,y,z)logP(x|y,z)$$

```
token_3gram = []
for para in corpus:
    cutword_list = jieba.lcut(para)
    token_3gram += combine3gram(cutword_list)
token_3gram_num = len(token_3gram)
ct3 = Counter(token_3gram)
vocab3 = ct3.most_common()
same_2st_word = [eve.split(" ")[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())
entropy_3gram = 0
for eve in vocab3:
    p_xyz = eve[1]/token_3gram_num
    first_2word = eve[0].split(" ")[0]
    entropy_3gram += -p_xyz*math.log(eve[1]/vocab_2st[first_2word], 2)
print("3-Gram词库总词数：", token_3gram_num, " ", "不同词的个数：", len(vocab3))
print("出现频率前10的3-gram词语：", vocab3[:10])
print("entropy_3gram:", entropy_3gram)
```

实验结果

Table 1: 16 本金庸小说在 N-Gram 语言模型下的信息熵

	1-Gram	2-Gram	3-Gram
Entropy	13.93153707433563	6.1361563015769915	0.9814784939104394

Table 2: 16 本金庸小说在 N-Gram 中的词库量

	词库总量	不同词数
1-Gram	2414377	265056
2-Gram	2354953	1701129
3-Gram	2296310	2205513

Table 3: 16 本金庸小说在 N-Gram 中的高频词

	1-Gram		2-Gram		3-Gram	
	高频词	出现次数	高频词	出现次数	高频词	出现次数
1	道	43430	笑 道	3932	韦宝笑 道	351
2	说	18799	甚 麽	1550	五岳剑 派	252
3	便	18050	忽 听	1268	微微笑 说道	198
4	说道	13434	突然 间	1195	新语丝电子 文库	145
5	见	13193	低声 道	1187	怦怦乱 跳	139
6	中	13068	道 说	940	令狐笑 道	137
7	听	11616	句 话	835	句话 说	133
8	甚	7868	笑 说道	824	说甚 麽	127
9	笑	7644	令狐 道	799	笑道 说	111
10	想	7481	微笑 道	630	微微笑 道	106

结论

计算结果显示金庸小说的信息量相对较大，其中 1-Gram 模型的信息熵最高，接近 14, 这符合我们对于武侠小说中充满复杂情节和人物关系的直觉感受。2-Gram 和 3-Gram 模型的信息熵显然要低很多，表明相邻两个和三个字符（或词）在文本中的组合情况较为有限，同样与实际感受一致。去除停用词、特殊符号等预处理方式，为结果的准确性做出了保障。同时也需要注意到，n-Gram 模型的适用范围是有限的，对于存在长距离依赖关系、结构复杂的文本数据可能性能仍可提升。