深度学习与自然语言处理报告

19377354 渠海榕 Hairong Qu
qhrbh2001@163.com

摘要

本文是深度学习与自然语言处理第一次作业的报告，本次作业的要求是以 16 本金庸武侠小说作为数据库，去除其中的中文停用词，利用信息熵和语言的 N 元模型，统计数据库中的字和词的一元、二元和三元模型的信息熵。信息熵用于描述一个随机变量的不确定性或者信息量大小。信息熵越大，代表这个随机变量越难以预测，不确定性越大，包含的信息量也就越大。最终经过计算得到的结果为字的一元信息熵为 9.951536369，字的二元信息熵为 7.022170161，字的三元信息熵为 3.494025429，词的一元信息熵 13.581265202，词的二元信息熵为 6.516757743，词的三元信息熵为 1.1751707674。通过观察，可以得出结论，随着元的增加，文本的信息熵会减少，因为元数的增加会导致文本中词组的分布变得更简单，文章变得更加有序，从而减少由字组成词和组成句的不确定性，减少文本的信息熵，这符合实际认知。

引言

信息熵是信息理论中的一个重要概念，用于描述随机变量的不确定性或信息量大小。简单来说，信息熵可以视为一个信源所产生的信息平均不确定度。信息论的奠基人之一，香农提出了信息熵这一重要概念。他认为，一个消息的信息量应该与其不确定性成正比，信息熵就是衡量随机变量不确定性的量度。信息熵越大，表示随机变量越难以预测，包含的信息量也越大。与英文或其他语言的信息熵概念相似，中文的信息熵用于衡量中文文本信息量的指标。在中文中，一个汉字或词语的出现往往具有一定的概率，文本中所有汉字或词语的出现概率分布构成中文文本的概率分布函数。通过计算该函数的信息熵，可以衡量中文文本的信息量大小和不确定性程度。中文的信息熵可用于文本分类、信息检索、语言模型等多个领域，对中文自然语言处理和文本分析具有重要意义。

原理

原理 1：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。如今熵 (Entropy)，信息熵，已经是机器学习中绕不开的一个概念。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。

信息熵的公式如下：

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

且规定：

$$0\log 0 \ = \ 0$$

根据英文论文《An Estimate of an Upper Bound for the Entropy of English》，对于文本 $X \ = \ \{...X_{-2},X_{-1},X_0,X_1,X_2...\}$ 来说，它是一个有限汉字表的随机过程，令 P 表示 X 的概率分布，E_p 表示关于 P 的期望，则它的信息熵定义为

$$H(X)\equiv H(P)\equiv -E_p\log P(X_0|X_{-1},X_{-2},...)$$

如果对数的底为 2，则熵以比特为单位。

其中信息熵具有三个基本性质：

1、单调性，发生概率越高的事件，其携带的信息量越低；

2、非负性，信息熵可以看作为一种广度量，非负性是一种合理的必然；

3、累加性，即多随机事件同时发生存在的总不确定性的量度是可以表示为各事件不确定性的量度的和，这也是广度量的一种体现。

原理 2：多随机变量系统中的信息熵

在自然语言处理中，文本的信息熵会涉及到多随机变量的情况，存在互信息、联合熵、条件熵等概念，以两随机变量系统为例，条件熵可以表示为：

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

该信息熵可以用于后文的二元模型(bi-gram)与三元模型(tri-gram)的计算。

对于中文数据库，如果计算字词的一元平均信息熵，那么其信息熵计算与前文无关系，计算公式为：

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

如果计算字词的二元和三元信息熵，其计算公式应该为除了最后一个字或词之前的字词的基础上，出现最后一个字的条件概率，因此计算公式为：

$$H(X)=-\sum_{x_1,x_2,\ldots,x_n\in X}P(x_1,x_2,\ldots,x_n)\log P(x_n|x_1,x_2,\ldots,x_{n-1})$$

实验过程

过程 1：数据读取

从压缩包解压得到的数据集中，包含 inf.txt，为书名的目录文档，其余为 16 本金庸武侠小说集，另外还有一个 cn_stopwords.txt 的文档，其中为中文的停用词表，是一些没有实际意义的词语，在处理时需要删除停用词。

过程 2：数据预处理

中文预处理中，首先需要读取 inf.txt 和 cn_stopwords.txt 的内容，然后遍历每一个出现在 inf.txt 中的文本，分别建立字典和词典。字典的建立过程如下，首先建立空字典，接着利用 str.replace()函数删除'\n'，'\u3000'，' '，'本书来自 www.cr173.com 免费 txt 小说下载站'和'更多更新免费电子书请关注 www.cr173.com'这些文本，之后遍历每一个字，如果这个字在停词表里则删除，不在则留下，判断字典中有无该字，没有则加入字典，出现次数为 1，已经存在则次数+1，生成字典。词典的建立过程如下，首先建立空字典，接着删除删'\n'，'\u3000'，' '，'本书来自 www.cr173.com 免费 txt 小说下载站'和'更多更新免费电子书请关注 www.cr173.com'这些文本，之后利用 jieba.cut()函数对文本内容进行分词，之后遍历每一个词，如果这个词在停词表里则删除，不在则留下，判断词典中有无该词，没有则加入词典，出现次数为 1，已经存在则次数+1，生成词典。

过程 3：数据处理

对于一元的模型，直接利用词典和字典生成。对于二元和三元的模型，分别遍历词典和字典生成二元字典和词典，三元字典和词典，然后利用公式即可得到结果。表 1 为统计一元、二元和三元的字的信息熵结果，表 2 为统计一元、二元和三元的词的信息熵结果。

表 1: 统计字的信息熵结果

书名	一元字	二元字	三元字
白马啸西风	9.225088438	4.088661761	1.211628493
碧血剑	9.754924249	5.675399767	1.79551116
飞狐外传	9.630063598	5.569083349	1.864978216
连城诀	9.515225215	5.09015163	1.638961809
鹿鼎记	9.658504948	6.01991119	2.409586468
三十三剑客图	10.00670228	4.284459665	0.650668379
射雕英雄传	9.751662326	5.965194686	2.196099645
神雕侠侣	9.662570987	6.002821622	2.283057878
书剑恩仇录	9.757709696	5.598027594	1.86160024
天龙八部	9.782441946	6.114860989	2.351603688
侠客行	9.434941719	5.380029257	1.819657283
笑傲江湖	9.515810087	5.856498737	2.361396239
雪山飞狐	9.50103951	4.801486096	1.303221092
倚天屠龙记	9.70642741	5.983031495	2.276120761
鸳鸯刀	9.21076864	3.656518274	0.896111121
越女剑	8.782444128	3.109167348	0.842115101
所有文章	9.951536369	7.022170161	3.494025429

表 2: 统计词的信息熵结果

书名	一元词	二元词	三元词
白马啸西风	11.12675812	2.912274028	0.370373725
碧血剑	12.88504353	3.96214504	0.430754846
飞狐外传	12.62570653	4.040617992	0.460855889
连城诀	12.20665339	3.5889817	0.368526835
鹿鼎记	12.63224504	4.992705021	0.838050026
三十三剑客图	12.53445375	1.808929425	0.090861795
射雕英雄传	13.03556051	4.592478038	0.537740386
神雕侠侣	12.74028327	4.684405215	0.63640612
书剑恩仇录	12.72735664	4.136925337	0.497420902
天龙八部	13.01890385	4.838749625	0.663325731
侠客行	12.28736023	3.992604985	0.512543379
笑傲江湖	12.52345978	4.838958475	0.795512539
雪山飞狐	12.05652502	3.065747927	0.290698042
倚天屠龙记	12.89063608	4.684704451	0.644051491
鸳鸯刀	11.09772869	2.156200112	0.2431723
越女剑	10.46726936	1.738085301	0.243693724
所有文章	13.581265202	6.516757743	1.1751707674

结论

我们无法准确地知道一个语言中特定文字的出现概率，甚至有时难以统计某种语言中究竟有多少种字符。信息论科学家只能通过各种手段来估计各个语言的信息熵，比如 Shannon 认为英语的信息熵在 0.6 到 1.3bits/字之间[1]，Cover 和 King 则认为英语的信息熵是 1.25bits/字[2]。信息论前辈们也排除万难，用统计采样的方式计算了汉语的信息熵，文字的信息熵为 9.63bits/字。我们可以发现，本次作业中所得到的文字的一元信息熵为 9.95bits/字，大于英文的信息熵，可见汉语的信息量更多。此外，我们也可以看到，随着元的增加，文本的信息熵会减少，因为元数的增加会导致文本中词组的分布变得更简单，文章变得更加有序，从而减少由字组成词和组成句的不确定性，减少文本的信息熵，这符合实际认知。

References

[1] Shannon C E. Prediction and entropy of printed English[J]. Bell system technical journal,1951, 30(1): 50-64.

[2] Cover T, King R. A convergent gambling estimate of the entropy of English[J]. IEEE Transactions on Information Theory, 1978, 24(4): 413-421.

附录

```
from collections import Counter
import os
import jieba
import math
basepath = 'C:\\Users\\HUAWEI\\Desktop\\深度学习与自然语言处理\\第一次作业'
with open(basepath+'\\jyxstxtqj_downcc.com\\inf.txt','r',encoding='ANSI') as f:
    booklist = f.read().split(',') #读目录
with open(basepath+"\\cn_stopwords.txt",'r',encoding = 'utf-8-sig') as f:
    stopwords = f.read().split() #读停用词
#stopwords = set(stopwords)

stopwords2 = [' ','\n','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','\u3000','目录']
```

```
def get_wf(words):
    twowords_fre = {}
    threewords_fre = {}
    for i in range(len(words)-1):
        twowords_fre[(words[i],words[i+1])] = twowords_fre.get((words[i],words[i+1]),0)+1
    for i in range(len(words)-2):
        threewords_fre[(words[i],words[i+1],words[i+2])] =
threewords_fre.get((words[i],words[i+1],words[i+2]),0)+1
    return twowords_fre,threewords_fre

for i in booklist:
    #data_txt = read_data(basepath+'\\jyxstxtqj_downcc.com\\'+i+'.txt')
    path = basepath+'\\jyxstxtqj_downcc.com\\'+i+'.txt'
    with open(path, 'r', encoding='ANSI') as f:
        data_txt = f.read()
    bookname = i
    for i in stopwords2:
        data_txt = data_txt.replace(i,'')#去除停用词二
    lettercount = Counter() #统计词频
    letter_num = 0
    letters = []

    entropy_oneletter = 0    #每个字的一元模型信息熵
    entropy_twoletters = 0     #每个字的二元模型信息熵
    entropy_threeletters = 0     #每个字的三元模型信息熵

    entropy_oneword = 0 #每个词的一元模型信息熵
    entropy_twowords = 0    #每个词的二元模型信息熵
    entropy_threewords = 0    #每个词的三元模型信息熵

    for letter in data_txt:
        if letter not in stopwords:
            letters.append(letter)
            lettercount[letter] += 1
            letter_num += 1
    for i in lettercount.most_common():
        px = i[1]/letter_num
        entropy_oneletter +=    (- px*math.log(px,2))
    #print(bookname,"的字的一元模型信息熵为",entropy_oneletter)

    two_letters_fre,thr_letters_fre = get_wf(letters)
    bigram_letter_len = sum([dic[1] for dic in two_letters_fre.items()])
    for bi_word in two_letters_fre.items():
        jp_xy = bi_word[1] / bigram_letter_len    #  计算联合概率 p(x,y)
```

```
        cp_xy = bi_word[1] / lettercount[bi_word[0][0]]  # 计算条件概率 p(x|y)
        entropy_twoletters +=( -jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵
# print(bookname,"的字的二元模型信息熵为",entropy_twoletters)

  trigram_letter_len = sum([dic[1] for dic in thr_letters_fre.items()])
  for tri_word in thr_letters_fre.items():
        jp_xy = tri_word[1] / trigram_letter_len  # 计算联合概率 p(x,y)
        cp_xy = tri_word[1] / two_letters_fre[tri_word[0][0:2]]  # 计算条件概率 p(x|y)
        entropy_threeletters +=( -jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵
  #print(bookname,"的字的三元模型信息熵为",entropy_threeletters)

  wordcount = Counter()
  word_num = 0
  words = []

  for word in jieba.cut(data_txt):

        if  word not in stopwords: #and len(word)>1 :
              words.append(word)
              wordcount[word] += 1
              word_num += 1
  for i in wordcount.most_common():
        px = i[1]/word_num
        entropy_oneword += (- px*math.log(px,2))
  #print(bookname,"的词的一元模型信息熵为",entropy_oneword)

  two_words_fre,thr_words_fre = get_wf(words)

  bigram_word_len = sum([dic[1] for dic in two_words_fre.items()])
  for bi_word in two_words_fre.items():
        jp_xy = bi_word[1] / bigram_word_len  # 计算联合概率 p(x,y)
        cp_xy = bi_word[1] / wordcount[bi_word[0][0]]  # 计算条件概率 p(x|y)
        entropy_twowords +=( -jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵
  #print(bookname,"的词的二元模型信息熵为",entropy_twowords)

  trigram_word_len = sum([dic[1] for dic in thr_words_fre.items()])
  for tri_word in thr_words_fre.items():
        jp_xy = tri_word[1] / trigram_word_len  # 计算联合概率 p(x,y)
        cp_xy = tri_word[1] / two_words_fre[tri_word[0][0:2]]  # 计算条件概率 p(x|y)
        entropy_threewords +=( -jp_xy * math.log(cp_xy, 2))  # 计算三元模型的信息熵
  #print(bookname,"的词的三元模型信息熵为",entropy_threewords)

  print(bookname,entropy_oneletter,entropy_twoletters,entropy_threeletters)
  #print(bookname,entropy_oneword,entropy_twowords,entropy_threewords)
```

```
# 统计所有文章的整合
data_txt = ''
for i in booklist:
    #data_txt = read_data(basepath+'\\jyxstxtqj_downcc.com\\'+i+'.txt')
    path = basepath+'\\jyxstxtqj_downcc.com\\'+i+'.txt'
    with open(path, 'r', encoding='ANSI') as f:
        data_txt += f.read()

for i in stopwords2:
    data_txt = data_txt.replace(i,'')#去除停用词二
lettercount = Counter() #统计词频
letter_num = 0
letters = []

entropy_oneletter = 0   #每个字的一元模型信息熵
entropy_twoletters = 0    #每个字的二元模型信息熵
entropy_threeletters = 0    #每个字的三元模型信息熵

entropy_oneword = 0 #每个词的一元模型信息熵
entropy_twowords = 0   #每个词的二元模型信息熵
entropy_threewords = 0   #每个词的三元模型信息熵

for letter in data_txt:
    if letter not in stopwords:
        letters.append(letter)
        lettercount[letter] += 1
        letter_num += 1
for i in lettercount.most_common():
    px = i[1]/letter_num
    entropy_oneletter +=   (- px*math.log(px,2))
print("总的字的一元模型信息熵为",entropy_oneletter)

two_letters_fre,thr_letters_fre = get_wf(letters)

bigram_letter_len = sum([dic[1] for dic in two_letters_fre.items()])
for bi_word in two_letters_fre.items():
    jp_xy = bi_word[1] / bigram_letter_len   # 计算联合概率 p(x,y)
    cp_xy = bi_word[1] / lettercount[bi_word[0][0]]   # 计算条件概率 p(x|y)
    entropy_twoletters +=( -jp_xy * math.log(cp_xy, 2))   # 计算二元模型的信息熵
print("总的字的二元模型信息熵为",entropy_twoletters)

trigram_letter_len = sum([dic[1] for dic in thr_letters_fre.items()])
for tri_word in thr_letters_fre.items():
```

```
    jp_xy = tri_word[1] / trigram_letter_len    # 计算联合概率 p(x,y)
    cp_xy = tri_word[1] / two_letters_fre[tri_word[0][0:2]]    # 计算条件概率 p(x|y)
    entropy_threeletters +=( -jp_xy * math.log(cp_xy, 2))    # 计算三元模型的信息熵
print("总的字的三元模型信息熵为",entropy_threeletters)

wordcount = Counter()
word_num = 0
words = []

for word in jieba.cut(data_txt):

    if    word not in stopwords: #and len(word)>1 :
          words.append(word)
          wordcount[word] += 1
          word_num += 1
for i in wordcount.most_common():
    px = i[1]/word_num
    entropy_oneword += (- px*math.log(px,2))
print("总的词的一元模型信息熵为",entropy_oneword)

two_words_fre,thr_words_fre = get_wf(words)

bigram_word_len = sum([dic[1] for dic in two_words_fre.items()])
for bi_word in two_words_fre.items():
    jp_xy = bi_word[1] / bigram_word_len    # 计算联合概率 p(x,y)
    cp_xy = bi_word[1] / wordcount[bi_word[0][0]]    # 计算条件概率 p(x|y)
    entropy_twowords +=( -jp_xy * math.log(cp_xy, 2))    # 计算二元模型的信息熵
print("总的词的二元模型信息熵为",entropy_twowords)

trigram_word_len = sum([dic[1] for dic in thr_words_fre.items()])
for tri_word in thr_words_fre.items():
    jp_xy = tri_word[1] / trigram_word_len    # 计算联合概率 p(x,y)
    cp_xy = tri_word[1] / two_words_fre[tri_word[0][0:2]]    # 计算条件概率 p(x|y)
    entropy_threewords +=( -jp_xy * math.log(cp_xy, 2))    # 计算三元模型的信息熵
print("总的词的三元模型信息熵为",entropy_threewords)
```