机器学习与自然语言处理实验一

院（系）名称	自动化科学与电气工程学院
学生姓名	赵怡然
学生学号	ZY2103208
指导老师	秦曾昌

2022 年 04 月 07 日

一、 实验背景介绍

信息是个很抽象的概念。人们常常说信息很多，或者信息较少，但却很难说清楚信息到底有多少。比如一本五十万字的中文书到底有多少信息量。直到 1948 年，香农（Shannon）在他的《通信的数学原理》论文中借用了热力学中熵的概念提出了“信息熵”的概念，解决了对信息的量化度量问题。

自然语言是一种上下文相关的信息表达和传递的方式，让计算机处理自然语言，一个基本的问题就是为自然语言这种上下文相关的特性建立数学模型，即统计语言模型。而信息熵的一个重要应用领域是自然语言处理，信息反映了内容的随机性及不确定性和编码的情况，与内容本身无关。而随着随机变量的增加，信息熵又衍生出联合熵、条件熵等概念。

信息熵作为被用来衡量一个系统的信息含量的指标，进而进一步作为系统优化的目标或则参数选择的判据；是具有总要的研究意义的。下面实验就信息熵的相关定义，根据所提供的文本数据库计算了不同定义下的信息熵，为后续自然语言处理打下了基础。

二、 实验目的

根据所提供的文本数据库，计算文本中的以词作为最小单位的信息熵、条件熵等。

三、 信息熵定义与计算

3.1 信息熵的定义与性质

信息熵的定义公式为：

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

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。

香农从数学上严格证明了满足上述三个条件的随机变量不确定性度量函数具有唯一形式及信息熵的定义。

3.2 互信息

两个离散随机变量 X 和 Y 的互信息（Mutual Information）定义为：

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

$$\begin{aligned} \log(\frac{p(x,y)}{p(x)p(y)}) &= \log\left(p(x,y) - (\log p(x) + \log p(y)\right) \\ &= -\log p(x) - \log p(y) - (-\log p(x,y)) \end{aligned}$$

上式表明，x、y 两事件的互信息为两事件单独发生的信息量之和减去同时发生的信息量之后剩余的信息量大小。特别的当两事件完全独立时，互信息为 0。

3.3 联合熵

两个离散随机变量 X 和 Y 的联合熵（Joint Entropy）为：

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

表示了两事件同时发生时系统的不确定度。

3.4 条件熵

两个离散随机变量 X 和 Y 的条件熵（Conditional Entropy）为：

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

表示了随机变量 X 的条件下随机变量 Y 的不确定性。

四、 实验任务及结果分析

根据信息熵定义结合统计语言模型计算所提供的 16 部小说的文本文件的信息熵。采用 Python 实现（未使用相关计算库）。

4.1 统计语言模型与信息熵

假设 S 表示某个句子，句子是由特定顺序的词 w1、w2、…、wn 组成,n 为句子的长度。现在想知道 S 在文本中出现的可能性，即 P(S)。此时需要有个模型来估算，不妨把 P(S) 展开表示为 P(S)=P(w1,w2,...,wn)。利用条件概率的公式，S 这个序列出现的概率等于每一个词出现的条件概率相乘，于是 P(w1,w2,...,wn)可展开为：

$$P(w1,w2,\dots,w3)$$

$$= P(w1)P(w2|w1)P(w3|w1,w2)\dots P(wn|w1,w2,\dots,wn-1)$$

其中$P(w1)$表示第一个词出现的概率，$P(w2|w1)$表示已知第一个词的前提下，第二个词出现的概率。

显然，当句子长度过长时$P(wn|w1,w2,\dots,wn-1)$的可能性太多，

无法估算，俄国数学家马尔可夫假设任意一个词 wi 出现的概率只同它前面的词 wi-1 有关，这种假设成为马尔可夫假设。

根据词的个数可以分为一元模型、二元模型、三元模型等。

（1）一元模型：

$$P(S) = P(w1)P(w2)P(w3)\dots P(wn)$$

（2）二元模型：

$$P(S) = P(w1)P(w2|w1)P(w3|w2)\dots P(wn|wn-1)$$

（3）三元模型：

$$P(S) = P(w1)P(w2|w1)P(w3|w2,w1)\dots P(wn|wn-1,wn-2)$$

根据不同的模型，信息熵的计算公式有所不同：

（1）一元模型

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

（2）二元模型

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

（3）三元模型

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

4.2 实验步骤及结果

4.2.1 中文文本的预处理

第一步：数据库中文文本中，存在一些非中文字符例如缩进、回车以及各类标点符号；并且在信息熵的计算中每句话之间是不存在相关性的。因此，在读取文本文件时，应按照常用的每句结尾的标点符

号作为根据，将文本文件拆分为不同的句子。

```
endCharList = ['.', '。', '?', '？', ';', '；', '...']
```

按字符对文本进行按句拆分

第二步：将每句的字符串进行分词处理，这里使用了 Jieba 库（https://github.com/fxsjy/jieba）。得到按顺序排列的汉语单词，在通过中文字符判断去掉非中文的部分，得到了最终的想要的按句、按词、无标点符号的语料。

4.2.2 词频统计

所采用的文本数据库，总中文字数为 7256740 个；分词个数为 4286240 词；平均词长为 1.693032 字。

4.2.3 词语关系统计

根据每句的分词结果，调用 NodeTree 类中的 update 函数，更新所统计的词语之间关系以及相关统计量。详细见 class NodeTree 代码部分。

4.2.4 计算信息熵

根据大数定理，词或二元词组或三元词组出现的概率大致等于其出现的频率。根据之前所提到的不同模型的信息熵计算公式：

（1）一元模型

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

（2）二元模型

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

（3）三元模型

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

4.2.5 实验结果

所使用的语料库的字数为 7256740 字，分词个数为 4286240 词，平均词长为 1.693032。

在一元模型中中文信息熵为 12.152782 比特/词；在二元模型中中文信息熵为 6.886784 比特/词；在三元模型中中文信息熵为 2.300682 比特/词。

```
(entropy_demo) PS C:\Users\Rannnnn\Desktop\entropy_demo> python Entropy.py
C:\Users\Rannnnn\Desktop\entropy_demo\database\三十三剑客图.txt
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Rannnnn\AppData\Local\Temp\jieba.cache
Loading model cost 0.639 seconds.
Prefix dict has been built successfully.
C:\Users\Rannnnn\Desktop\entropy_demo\database\书剑恩仇录.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\侠客行.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\倚天屠龙记.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\天龙八部.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\射雕英雄传.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\白马啸西风.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\碧血剑.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\神雕侠侣.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\笑傲江湖.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\越女剑.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\连城诀.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\雪山飞狐.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\飞狐外传.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\鸳鸯刀.txt
C:\Users\Rannnnn\Desktop\entropy_demo\database\鹿鼎记.txt
计算信息熵所用数据库字数总计7256740字分词个数为4286240词，平均词长为1.693032
1元模型信息熵: 12.152782比特/词
2元模型信息熵: 6.886784比特/词
3元模型信息熵: 2.300682比特/词
(entropy_demo) PS C:\Users\Rannnnn\Desktop\entropy_demo>
```

附录

源代码

```
1. from io import TextIOWrapper
2.
3. '''''Entropy Demo
```

```
  1. Calculate the information entropy of Chinese by using the database.
'''

import jieba
import math
import os

class ReadLine:
  '''''
  Class: Readline - 按 self.endCharList 中的所有 char 分割文本的 Generator
  '''
  MAX_SIZE = 100
  endCharList = ['.', '。', '?', '？', ';', '；', '...']

  def __init__(self, f: TextIOWrapper) -> None:
    self.buf = ''
    self.f = f
    return

  def __iter__(self):
    return self

  def __next__(self):
    pos, char = self.search()
    if pos >= 0 and char:
      line = self.buf[: pos]
      self.buf = self.buf[pos + len(char):]
      return line
    else:
      chunk = self.f.read(self.MAX_SIZE)
      if chunk:
        self.buf += chunk
        return self.__next__()
      else:
        self.f.close()
        raise StopIteration

  def search(self):
    dict = {}
    for char in self.endCharList:
      try:
        pos = self.buf.index(char)
```

```
        dict[char] = pos
      except:
        dict[char] = -1
    res = (-1, None)
    for char in dict:
      pos = dict[char]
      if pos > -1:
        res = (pos, char)
    return res

class NodeTree:
  def __init__(self, word: str = 'root', isRoot: bool = True, level: int = 0) -> None:
    self.nextNodeDict = {}
    self.nextCount = 0
    self.count = 1
    self.parentNode = None
    self.word = word
    self.level = level
    if isRoot:
      self.levelDict = {}
      self.levelCount = {}
    return

  def add(self, wordList: list[str], index: int = 0) -> None:
    if len(wordList) <= index:
      return
    word: str = wordList[index]
    nextNode: NodeTree = self.nextNodeDict.get(word)
    if nextNode == None:
      nextNode = NodeTree(word = word, isRoot = False, level = self.level + 1)
      nextNode.parentNode = self
      self.nextNodeDict[word] = nextNode
      nextNode.bindToRoot(nextNode)
    else:
      nextNode.count += 1
    self.nextCount += 1
    self.countUpdateToRoot(nextNode)
    nextNode.add(wordList, index + 1)
    return

  def bindToRoot(self, node) -> None:
    parentNode: NodeTree = self.parentNode
```

```
    bindNode: NodeTree = node
    if parentNode:
      parentNode.bindToRoot(node)
    else:
      nodeList: list = self.levelDict.get(bindNode.level)
      if nodeList != None:
        nodeList.append(bindNode)
      else:
        self.levelDict[bindNode.level] = [bindNode]
    return

  def countUpdateToRoot(self, node) -> None:
      parentNode: NodeTree = self.parentNode
      updateNode: NodeTree = node
      if parentNode:
        parentNode.countUpdateToRoot(node)
      else:
        prevCount = self.levelCount.get(updateNode.level)
        if prevCount != None:
          self.levelCount[updateNode.level] += 1
        else:
          self.levelCount[updateNode.level] = 1
        return

    def clear(self) -> None:
      self.nextNodeDict.clear()
      self.nextCount = 0
      return


  class Entropy:
    def __init__(self) -> None:
      self.nodeTree = NodeTree()
      self.charCount = 0
      self.wordCount = 0

    # 工具函数

    def getFilePathFromDB(self) -> list[str]:
      '''''
      Function: getFilePathFromDB - 返回.\database 中所有 txt 文件地址列
    表
      '''
      list = []
```

```
131.       dirname = os.path.join(os.path.dirname(os.path.abspath(__file_
    _)), 'database')
132.       for filename in os.listdir(dirname):
133.         list.append(os.path.join(dirname, filename))
134.       return list
135.
136.     def readFile(self, filePath: str) -> ReadLine:
137.       '''''
138.       Function: readFile - 读取 TXT
139.
140.       Parameters:
141.         filePath: str - txt 文件地址
142.
143.       Return:
144.         lines: Iterator - 返回文本文件的句子迭代器
145.       '''
146.       file = open(filePath, encoding='utf-8')
147.       lines = ReadLine(file)
148.       return lines
149.
150.     def isChinese(self, word: str) -> bool:
151.       '''''
152.       Function: isChinese - 判断是否所有字符均为中文
153.
154.       Parameters:
155.         word: str - 被判断的字符串
156.
157.       Return: bool - 所有字符均为汉字返回 True, 否则为 False
158.       '''
159.       for ch in word:
160.         if ('\u4e00' <= ch <= '\u9fff'):
161.           pass
162.         else:
163.           return False
164.       return True
165.
166.
167.     # 核心函数
168.     def divide(self, line: str):
169.       '''''
170.       Function: divide - 分词处理返回分词结果列表
171.
172.       Parameters:
173.         line: str - 输入字符串分词处理
```

```
174.
175.       Return:
176.         wordList: list[str] - 返回分词字符串列表
177.       '''
178.       wordList = []
179.       for word in jieba.cut(line, cut_all=False):
180.         if self.isChinese(word):
181.           wordList.append(word)
182.       return wordList
183.
184.     def updateDict(self, wordList: list[str], mode: int = 1) -> None
    :
185.       '''''
186.       Function: updateDict - 根据分词结果更新词语之间的关系 NodeTree, 并
    记录字数与分词个数
187.
188.       Parameters:
189.         wordList: list[str] - 分词结果
190.         mode: int - 以 mode 作为最高元，进行 NodeTree 更新
191.
192.       Return: None
193.       '''
194.       self.wordCount += len(wordList)
195.       charCount = 0
196.       for word in wordList:
197.         charCount += len(word)
198.       self.charCount += charCount
199.       for i in range(0, len(wordList)):
200.         list = wordList[i: i + mode]
201.         self.nodeTree.add(list)
202.       return
203.
204.     def clear(self) -> None:
205.       '''''
206.       Function: clear- 初始化 NodeTree, 汉字和分词个数
207.       Return: None
208.       '''
209.       self.nodeTree.clear()
210.       self.charCount = 0
211.       self.wordCount = 0
212.       return
213.
214.     def calculate(self, mode: int = 1) -> float:
215.       '''''
```

```
216.       Function: calculate - 根据词语之间的关系计算信息熵
217.
218.       Parameters:
219.         mode: int = 1 - 计算 N 元模型信息熵
220.
221.       Return:
222.         h: float - 信息熵
223.       '''
224.       nodeList: list[NodeTree] = self.nodeTree.levelDict.get(mode)
225.       if not nodeList:
226.         return
227.       h: float = 0
228.       for node in nodeList:
229.         parentNode: NodeTree = node.parentNode
230.         h -
    = node.count / self.nodeTree.levelCount[mode] * math.log2( node.count
     / parentNode.nextCount )
231.       return h
232.
233.     def run(self, mode: int = 1) -> None:
234.       '''''
235.       Function: run - 默认运行，统计.\database 中所有 txt 文件，并计算信
    息熵
236.
237.       Parameters:
238.         mode: int = 1 - 以 mode 作为最高元，计算所有模型的信息熵
239.                         eg. mode = 3，计算一元、二元、三元模型的信息熵
240.
241.       Return: - None
242.       '''
243.       self.clear()
244.       filePathList = self.getFilePathFromDB()
245.       for filePath in filePathList:
246.         print(filePath)
247.         lines = self.readFile(filePath)
248.         for line in lines:
249.           wordList = self.divide(line)
250.           self.updateDict(wordList, mode)
251.       print('计算信息熵所用数据库字数总计%d 字分词个数为%d 词，平均词长
    为%f' %(self.charCount, self.wordCount, self.charCount / self.wordCoun
    t))
252.       for i in range(0, mode):
253.         print('%d 元模型信息熵: %f 比特/词
    ' %(i+1, app.calculate(i + 1)))
```

```
254.       return
255.
256.   if __name__ == '__main__':
257.     app = Entropy()
258.       app.run(mode=3)
```