Report of Deep Learning for Natural Langauge Processing

高志磊
gaozhilei@buaa.edu.cn

Abstract

本文根据文献[1]首先介绍了 N 元语言统计模型估计中文信息熵的方法，根据所给数据库分别以字和词为单位计算中文平均信息熵。

Introduction

信息是个很抽象的概念，人们常说信息很多，或者信息很少，但却很难说清楚信息到底有多少。比如一本书到底有多少信息量。信息论之父 Claude Elwood Shannon 第一次用数学语言阐明了概率和信息冗余度的关系。

Shannon 在 1948 年发表的论文“通信的数学理论（A Mathematical Theory of Communication）”[2]中指出，任何信息都存在冗余，冗余大小与信息中每个符号的出现概率或者说不确定性有关。Shannon 借鉴了热力学的概念，把信息中排除了冗余后的平均信息量称为“信息熵”，并给出了计算信息熵的数学表达式。

Methodology

M1: 信息熵

通常，一个信源发送出什么符号是不确定的，衡量它可以根据其出现的概率来度量。概率大，出现机会多，不确定性小；反之不确定性就大。

不确定性函数 f 是概率 P 的减函数；两个独立符号所产生的不确定性应等于各自不确定性之和，即 $f(P_1,P_2)=f(P_1)+f(P_2)$，这称为可加性。同时满足这两个条件的函数 f 是对数函数，即

$$f(P)=\log\frac{1}{p}=-\log p$$

在信源中，考虑的不是某一单个符号发生的不确定性，而是要考虑这个信源所有可能发

生情况的平均不确定性。若信源符号有 n 种取值：$U_1,\cdots,U_i,\cdots,U_n$，对应概率为：$P_1,\cdots,P_i,\cdots,P_n$，且各种符号的出现彼此独立。这时，信源的平均不确定性应当为单个符号不确定性 $-\log p_i$ 的统计平均值 (E)，可称为信息熵，即：

$$H(U)=E[-\log p_i]=-\sum_{i=1}^{n} p_i \log p_i$$

式中对数一般取 2 为底，单位为比特。

M2：统计语言模型

假定 S 表示某一个有意义的句子，由一连串特定顺序排列的词 $w_1,w_2,\cdots,w_n$ 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 $P(S)=P(w_1,w_2,\cdots,w_n)$。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 $P(w_1,w_2,\cdots,w_n)$ 可展开为：

$$P(w_1,w_2,\cdots,w_n)=P(w_1)P(w_2\mid w_1)P(w_3\mid w_1,w_2)\cdots P(w_n\mid w_1,w_2,\cdots,w_{n-1})$$

其中 $P(w_1)$ 表示第一个词 w_1 出现的概率；$P(w_2\mid w_1)$ 是在已知第一个词为 w_1 的前提下，第二个词 w_2 出现的概率，后面以此类推。

显然，当句子长度过长时，$P(w_n\mid w_1,w_2,\cdots,w_{n-1})$ 的可能性太多，无法估算。假设该句子具有马尔科夫性，即任意一个词 w_i 出现的概率只同它前面的词 w_{i-1} 有关，S 的概率变为：

$$P(S)=P(w_1)P(w_2\mid w_1)P(w_3\mid w_2)\cdots P(w_n\mid w_{n-1})$$

其对应的统计语言模型为二元模型。也可以假设一个词由前面 $N-1$ 个词决定，即 N 元模型。当 $N=1$ 时，每个词出现的概率与其他词无关，为一元模型，S 对应的概率变为：

$$P(S)=P(w_1)P(w_2)\cdots P(w_i)\cdots P(w_n)$$

M3：信息熵计算

如果统计量足够大，字、词、二元词组或三元词组出现的概率大致等于其出现的频率。由此可得，字和词的信息熵计算公式为：

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

其中，$P(x)$可近似等于每个字或词在语料库中出现的频率。

二元模型的信息熵计算公式为：

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

其中，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。

三元模型的信息熵计算公式为：

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

其中，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。

M4：程序实现

（一）文本预处理

所用数据库为金庸小说，读取文本后需要进行一些处理包括：

1）删除开头无用信息

2）删除中文停词及标点符号（利用 cn_stopwords.txt 文件）

3）删除空白符号，比如空格、换行符等。

程序如下：

```
def read_file(self, filename=""):
    # 如果未指定名称，则默认为类名
    if filename == "":
        filename = self.name
    target = "ChineseDataSet/" + filename + ".txt"
    with open(target, "r", encoding='gbk', errors='ignore') as f:
        self.data = f.read()
        self.data = self.data.replace(
            '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', '')
        f.close()
    # 分词
    for words in jieba.cut(self.data):
        if (words not in self.stop_word) and (not words.isspace()):
            self.split_word.append(words)
            self.split_word_len += 1
    # 统计字数
    for word in self.data:
        if (word not in self.stop_word) and (not word.isspace()):
        # if not word.isspace():
            self.word.append(word)
            self.word_len += 1
```

（二）得到词频表

一元模型：

```
def get_unigram_tf(self, word):
    # 得到单个词的词频表
    unigram_tf = {}
    for w in word:
        unigram_tf[w] = unigram_tf.get(w, 0) + 1
    return unigram_tf
```

二元模型：

```
def get_bigram_tf(self, word):
    # 得到二元词的词频表
    bigram_tf = {}
    for i in range(len(word) - 1):
        bigram_tf[(word[i], word[i + 1])] = bigram_tf.get(
            (word[i], word[i + 1]), 0) + 1
    return bigram_tf
```

三元模型：

```
def get_trigram_tf(self, word):
    # 得到三元词的词频表
    trigram_tf = {}
    for i in range(len(word) - 2):
        trigram_tf[(word[i], word[i + 1], word[i + 2])] = trigram_tf.get(
            (word[i], word[i + 1], word[i + 2]), 0) + 1
    return trigram_tf
```

（三）计算信息熵

一元模型：

```
def calc_entropy_unigram(self, word, is_ci=0):
    # 计算一元模型的信息熵
    word_tf = self.get_unigram_tf(word)
    word_len = sum([item[1] for item in word_tf.items()])
    entropy = sum(
        [-(word[1] / word_len) * math.log(word[1] / word_len, 2) for word in
         word_tf.items()])
    if is_ci:
        print("<{}>基于词的一元模型的中文信息熵为：{}比特/词".format(self.name, entropy))
    else:
        print("<{}>基于字的一元模型的中文信息熵为：{}比特/字".format(self.name, entropy))
    return entropy
```

二元模型：

```python
def calc_entropy_bigram(self, word, is_ci=0):
    # 计算二元模型的信息熵
    # 计算二元模型总词频
    word_tf = self.get_bigram_tf(word)
    last_word_tf = self.get_unigram_tf(word)
    bigram_len = sum([item[1] for item in word_tf.items()])
    entropy = []
    for bigram in word_tf.items():
        p_xy = bigram[1] / bigram_len  # 联合概率p(xy)
        p_x_y = bigram[1] / last_word_tf[bigram[0][0]]  # 条件概率p(x|y)
        entropy.append(-p_xy * math.log(p_x_y, 2))
    entropy = sum(entropy)
    if is_ci:
        print("<{}>基于词的二元模型的中文信息熵为：{}比特/词".format(self.name, entropy))
    else:
        print("<{}>基于字的二元模型的中文信息熵为：{}比特/字".format(self.name, entropy))
    return entropy
```

三元模型：

```python
def calc_entropy_trigram(self, word, is_ci):
    # 计算三元模型的信息熵
    # 计算三元模型总词频
    word_tf = self.get_trigram_tf(word)
    last_word_tf = self.get_bigram_tf(word)
    trigram_len = sum([item[1] for item in word_tf.items()])
    entropy = []
    for trigram in word_tf.items():
        p_xy = trigram[1] / trigram_len  # 联合概率p(xy)
        p_x_y = trigram[1] / last_word_tf[(trigram[0][0], trigram[0][1])]  # 条件概率
        entropy.append(-p_xy * math.log(p_x_y, 2))
    entropy = sum(entropy)
    if is_ci:
        print("<{}>基于词的三元模型的中文信息熵为：{}比特/词".format(self.name, entropy))
    else:
        print("<{}>基于字的三元模型的中文信息熵为：{}比特/字".format(self.name, entropy))
    return entropy
```

Experimental Studies

以金庸小说作为语料库，去除标点符号、停词等无用信息，计算信息熵如表 1 所示。

Table 1：语料库信息

	语料库名称	信息熵（比特/词）					
		一元字	二元字	三元字	一元词	二元词	三元词
1	白马啸西风	9.2209	4.0907	1.2126	11.1920	2.8818	0.3543
2	碧血剑	9.7550	5.6754	1.7955	12.8850	3.9621	0.4307
3	飞狐外传	9.6301	5.5691	1.8650	12.6259	4.0404	0.4609
4	连城诀	9.5152	5.0902	1.6390	12.2066	3.5890	0.3685
5	鹿鼎记	9.6583	6.0200	2.4097	12.6390	4.9929	0.8344
6	三十三剑客图	10.0067	4.2845	0.6507	12.5335	1.8096	0.0912
7	射雕英雄传	9.7411	5.9700	2.1995	13.0359	4.6006	0.5341

8	神雕侠侣	9.6628	6.0025	2.2828	12.7604	4.6874	0.6265
9	书剑恩仇录	9.7448	5.6043	1.8654	12.7148	4.1461	0.4986
10	天龙八部	9.7807	6.1155	2.3522	13.0172	4.8399	0.6636
11	侠客行	9.4349	5.3800	1.8197	12.2875	3.9929	0.5127
12	笑傲江湖	9.5158	5.8565	2.3614	12.5238	4.8388	0.7955
13	雪山飞狐	9.5010	4.8015	1.3032	12.0564	3.0659	0.2906
14	倚天屠龙记	9.7066	5.9830	2.2760	12.8937	4.6851	0.6426
15	鸳鸯刀	9.2108	3.6565	0.8961	11.1362	2.1465	0.2323
16	越女剑	8.7824	3.1092	0.8421	10.5021	1.7351	0.2331
17	total	9.9490	7.0229	3.4951	13.5835	6.5218	1.1713

绘制各语料库信息熵柱状图如下：

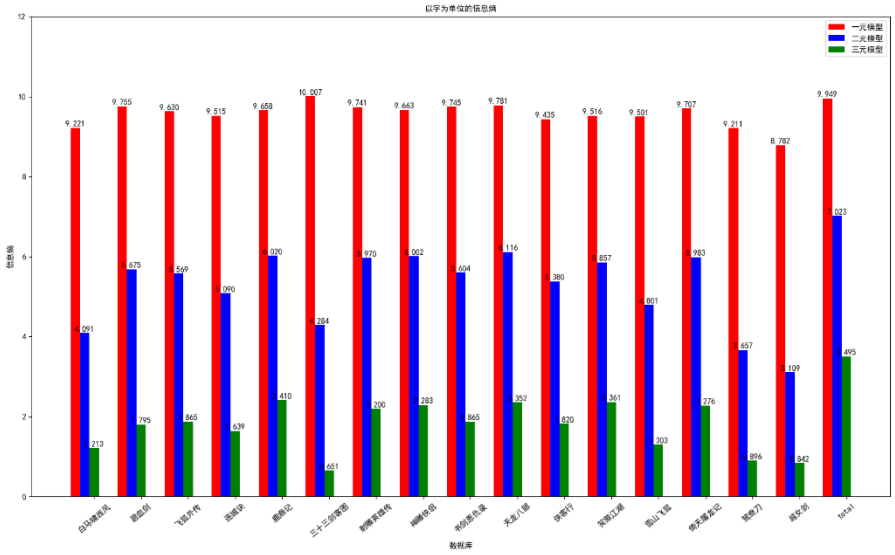

Figure 1：各语料库信息熵（以字为单位）

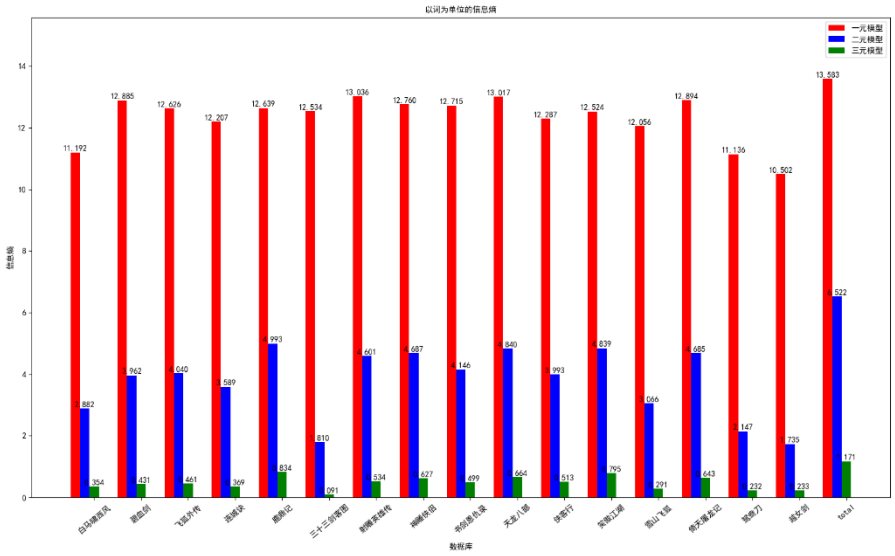

Figure 2：各语料库信息熵（以词为单位）

Conclusions

对比一元模型、二元模型、三元模型可以看到，N 取值越大，即考虑前后文关系的长度越大，文本的信息熵越小，这是因为 N 越大，组成该词组的词越多，其冗余度也就越小，使用的特定场景越小，出现在文章中的不确定性越小。

References

[1] Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. 1992. An estimate of an upper bound for the entropy of English. Comput. Linguist. 18, 1 (March 1992), 31–40.
[2] C. E. Shannon, "A mathematical theory of communication," in The Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, July 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.