# 深度学习与自然语言处理第一次作业

**SY2106318 孙旭东**

## 1.作业内容

阅读文章《Entropy_of_English_PeterBrown》，参考此文章来计算中文(分别以词和字为单位)的平均信息熵。

## 2.相关知识

### 2.1信息熵

信息熵的概念最早由香农（1916-2001）于1948年借鉴热力学中的"热熵"的概念提出，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。其数学公式可以表示为：

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

论文中假设$X=\{...X_{-2},X_{-1},X_0,X_1,X_2,...\}$是基于有限字母表的平稳随机过程，$P$为$X$的概率分布，$E_P$为$P$的数学期望，则$X$的信息熵定义为

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

当对数底数为2时，信息熵的单位为bit，相关理论说明$H(P)$可以表示为

$$H(P)=\lim_{n\to\infty}-E_PlogP(X_0|X_{-1},X_{-2},...,X_{-n})=\lim_{n\to\infty}-\frac{1}{n}E_PlogP(X_1X_2...X_n)$$

如果$P$是遍历的，则$E_P=1$。我们无法精确获取$X$的概率分布，即无法获取精确的$P$，但可以通过足够长的随机样本来估计$P$，通过建立$P$的随机平稳过程模型$M$来估算$H(P)$的上界，与上述推理过程相同，我们可以得到以下公式：

$$H(P,M)=\lim_{n\to\infty}-E_PlogM(X_0|X_{-1},X_{-2},...,X_{-n})=\lim_{n\to\infty}-\frac{1}{n}E_MlogP(X_1X_2...X_n)$$

$H(P,M)$有较大的参考价值，因为其是$H(P)$的一个上界，即$H(P)\leq H(P,M)$，更加准确的模型能够产生更加精确的上界。从文本压缩的角度来理解信息熵，对于$X_1X_2...X_n$的任意编码方式，$l(X_1X_2...X_n)$为编码所需的比特数，均有

$$E_Pl(X_1X_2...X_n)\geq -E_PlogP(X_1X_2...X_n)$$

由上述分析知，$H(P)$是对从P中提取的长字符串进行编码所需的每个符号的平均位数的下限，每个符号编码时需要的位数越多，即熵越高，说明混乱程度越高，单个字符携带的信息量越大。

### 2.2统计语言模型（N-Gram）

统计语言模型是基于预先人为收集的大规模语料数据，以真实的人类语言为标准，预测文本序列在语料库中可能出现的概率，并以此概率去判断文本是否"合法"，是否能被人所理解。

给定一个句子（词语序列）：$S=W_1,W_2,...W_k$，它的概率可以表示为：

$$P(S)=P(W_1,W_2,...,W_k)=p(W_1)P(W_2|W_1)...P(W_k|W_1,W_2,...W_{k-1})=\prod_iP(W_i|W_1,...,W_{i-1})$$

但由于直接这样计算会导致参数空间过大，数据稀疏严重等问题，假设一个语料库中单词的数量为$|V|$个，一个句子由$n$个词组成，那么每个单词都可能有|V|个取值，那么由这些词组成的n元组合的数目为$|V|^n$ 种，也就是说，组合数会随着$n$的增加而呈现指数级别的增长，随着$n$的增加，语料数据库能够提供的数据是非常有限的，也就是说依据最大似然估计得到的概率将会是0，模型可能仅仅能够计算寥寥几个句子。所以可以引入N-Gram模型。在马尔可夫假设下，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，这样我们前面得到的条件概率的计算可以简化如下，

$$P(W_i|W_1,\dots,W_{i-1}) \approx P(W_i|W_{i-k}\dots W_{i-1})$$

当$k=0$时对应的模型为一元模型(unigram)，即$W_i$不与任何词相关，每个词都是相互独立的，

$$P(W_1,W_2,\dots,W_k)=\prod_i P(W_i)$$

当$k=1$时对应的模型为二元模型(bigram)，即$W_i$只与它前面的一个词相关，

$$P(W_1,W_2,\dots,W_k)=\prod_i P(W_i|W_{i-1})$$

当$k=2$时对应的模型为三元模型(trigram)，即$W_i$只与它前面的两个词相关，

$$P(W_1,W_2,\dots,W_k)=\prod_i P(W_i|W_{i-2,i-1})$$

# 3.实验过程

## 3.1数据准备

使用到的数据为金庸的16本武侠小说，首先对数据进行预处理，代码如下：

```
def get_single_corpus(file_path):
    """
    获取file_path文件对应的内容
    :return: file_path文件处理结果
    """
    corpus = ''
    # unuseful items filter
    r1 = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@, 。?★、…【】《》？“”‘’！[\\]^_`{|}~「」『』
 () ]+'
    with open(file_path, 'r', encoding='ANSI') as f:
        corpus = f.read()
        corpus = re.sub(r1, '', corpus)
        corpus = corpus.replace('\n', '')
        corpus = corpus.replace('\u3000', '')
        corpus = corpus.replace('本书来自免费小说下载站更多更新免费电子书请关注', '')
        f.close()
    return corpus
```

在以ANSI编码格式读取文件内容后，删除文章内的所有非中文字符，以及和小说内容无关的片段，得到字符串形式的语料库，之后按照”分词“和”分字符“两种不同模式生成词频字典，在”分词“模式下，用jieba库中的cut函数对原始语料库进行处理，在”分字符“模式下，使用原始语料库，获取词频字典相关函数如下：

```
def get_tf(tf_dic, words):
    """
```

```
    获取一元词词频
    :return:一元词词频dic
    """
    for i in range(len(words)):
        tf_dic[words[i]] = tf_dic.get(words[i], 0) + 1


def get_bigram_tf(tf_dic, words):
    """
    获取二元词词频
    :return:二元词词频dic
    """
    for i in range(len(words)-1):
        tf_dic[(words[i], words[i+1])] = tf_dic.get((words[i], words[i+1]), 0) + 1


def get_trigram_tf(tf_dic, words):
    """
    获取二元词词频
    :return:二元词词频dic
    """
    for i in range(len(words)-2):
        tf_dic[((words[i], words[i+1]), words[i+2])] = tf_dic.get(((words[i], words[i+1]),
words[i+2]), 0) + 1
```

## 3.2信息熵计算

以三元信息熵的计算说明信息熵计算过程，输入参数为"二元词频字典"、"三元词频字典"和"语料库长度"，平均词长的计算原则为所有不同词的长度之和除以所有不同词的计数。信息熵的计算则对三元词频字典中每个键（key）计算联合概率密度和条件概率密度，然后进行熵值的累加。

```
def calculate_trigram_entropy(bigram_tf_dic, trigram_tf_dic, len_data):
    """
    计算三元词的信息熵
    :return:
    """
    begin = time.time()
    tri_words_num = sum([item[1] for item in trigram_tf_dic.items()])
    avg_word_len = sum(len(item[0][i]) for item in trigram_tf_dic.items() for i in
range(len(item[0])))/len(trigram_tf_dic)

    print("分词个数：{}".format(tri_words_num))
    print('不同词个数：{}'.format((len(trigram_tf_dic))))
    print("平均词长：{:.6f}".format(avg_word_len))

    entropy = 0
    for tri_item in trigram_tf_dic.items():
        jp = tri_item[1] / tri_words_num
        cp = tri_item[1] / bigram_tf_dic[tri_item[0][0]]
        entropy += -jp * math.log(cp, 2)
    print("基于分词的三元模型中文信息熵为：{:.6f} 比特/词".format(entropy))
```

```
    end = time.time()
    print("三元模型运行时间: {:.6f} s".format(end - begin))
    return ['trigram', len_data, len(trigram_tf_dic), round(avg_word_len, 6),
round(entropy, 6)]
```

其返回参数为'分词模型', '语料字数', '分词个数', '平均词长', '信息熵（比特/词）', 用于生成表格进行结果的展示。

## 3.3结果生成

为了方便的进行结果展示，除了在控制台打印结果外，还编写了markdown表格生成函数，代码如下：

```
def print_md(table_name, head, row_title, col_title, data):
    """
    :param table_name: 表名
    :param head: 表头
    :param row_title: 行名，编号，1，2，3.....
    :param col_title: 列名，词数，运行时间等
    :param data: {ndarray(H, W)}
    :return:字符串形式的markdown表格
    """
    element = " {} |"

    h, w = len(data), len(data[0])
    lines = ['#### {}'.format(table_name)]

    lines += ["| {} | {} |".format(head, ' | '.join(col_title))]

    # 分割线
    split = "{}:{}"
    line = "| {} |".format(split.format('-' * len(head), '-' * len(head)))
    for i in range(w):
        line = "{} {} |".format(line, split.format('-' * len(col_title[i]), '-' *
len(col_title[i])))
    lines += [line]

    # 数据部分
    for i in range(h):
        d = list(map(str, list(data[i])))
        lines += ["| {} | {} |".format(row_title[i], ' | '.join(d))]

    table = '\n'.join(lines)
    print(table)
    return table
```

输入表名、表头、行索引、列索引及数据后，打印并返回对应格式的markdown表格。

# 实验结果

### 金庸小说全集（16本）信息熵表（分词模式）

| # | 分词模型 | 语料字数 | 分词个数 | 平均词长 | 信息熵（比特/词） |
|---|---|---|---|---|---|
| 1 | unigram | 7282998 | 172676 | 1.696304 | 12.172905 |
| 2 | bigram | 7282998 | 1982472 | 3.87425 | 6.962872 |
| 3 | trigram | 7282998 | 3575450 | 3.749168 | 2.302969 |

## 金庸小说全集（16本）信息熵表（字符模式）

| # | 分词模型 | 语料字数 | 分词个数 | 平均词长 | 信息熵（比特/词） |
|---|---|---|---|---|---|
| 1 | unigram | 7282998 | 5778 | 1.0 | 9.534901 |
| 2 | bigram | 7282998 | 738777 | 2.0 | 6.734948 |
| 3 | trigram | 7282998 | 3223048 | 3.0 | 3.951713 |

## 金庸小说单本信息熵表（分词模式）

| # | 小说名 | 语料字数 | 一元分词个数 | 一元平均词长 | 一元模型信息熵（比特/词） | 二元分词个数 | 二元平均词长 | 二元模型信息熵（比特/词） | 三元分词个数 | 三元平均词长 | 三元模型信息熵（比特/词） |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 三十三剑客图 | 53399 | 11275 | 1.706584 | 11.681066 | 27817 | 3.50541 | 2.94602 | 30791 | 3.710272 | 0.272666 |
| 2 | 书剑恩仇录 | 435906 | 33902 | 1.719598 | 11.719617 | 173944 | 3.69462 | 5.02564 | 237891 | 3.735568 | 1.038883 |
| 3 | 侠客行 | 309887 | 23552 | 1.689936 | 11.188972 | 119942 | 3.65776 | 4.955795 | 169198 | 3.71028 | 1.12706 |
| 4 | 倚天屠龙记 | 818422 | 46720 | 1.723059 | 11.760183 | 295229 | 3.755288 | 5.518218 | 434036 | 3.746558 | 1.328639 |
| 5 | 天龙八部 | 1021603 | 54701 | 1.689835 | 11.736275 | 358425 | 3.73121 | 5.677541 | 542047 | 3.721688 | 1.480923 |
| 6 | 射雕英雄传 | 767899 | 49168 | 1.699441 | 11.834495 | 289349 | 3.702895 | 5.469081 | 417279 | 3.722929 | 1.266258 |
| 7 | 白马啸西风 | 57437 | 7167 | 1.633729 | 10.242509 | 25842 | 3.458401 | 3.991294 | 33355 | 3.64551 | 0.741034 |
| 8 | 碧血剑 | 416533 | 34049 | 1.714982 | 11.747485 | 169603 | 3.675837 | 4.996851 | 229135 | 3.730325 | 0.993505 |
| 9 | 神雕侠侣 | 827823 | 49026 | 1.659795 | 11.561237 | 302521 | 3.666545 | 5.671443 | 454349 | 3.687681 | 1.434044 |
| 10 | 笑傲江湖 | 824948 | 41587 | 1.709397 | 11.398787 | 277816 | 3.760651 | 5.618827 | 429486 | 3.738958 | 1.527482 |
| 11 | 越女剑 | 13663 | 2981 | 1.694531 | 10.072456 | 6996 | 3.455689 | 2.529249 | 7890 | 3.695691 | 0.328286 |
| 12 | 连城诀 | 194542 | 17719 | 1.659094 | 11.043223 | 81134 | 3.566631 | 4.691516 | 110400 | 3.672726 | 0.955575 |
| 13 | 雪山飞狐 | 113428 | 13243 | 1.685158 | 11.110252 | 50502 | 3.571918 | 4.110558 | 63759 | 3.696545 | 0.693631 |
| 14 | 飞狐外传 | 375225 | 28917 | 1.698327 | 11.528414 | 149762 | 3.661817 | 4.994856 | 206556 | 3.715399 | 1.053167 |
| 15 | 鸳鸯刀 | 29290 | 5143 | 1.705584 | 10.478126 | 14191 | 3.540342 | 3.085121 | 16662 | 3.709279 | 0.432056 |
| 16 | 鹿鼎记 | 1022993 | 49960 | 1.694323 | 11.447505 | 338794 | 3.740019 | 5.752031 | 531493 | 3.728335 | 1.619806 |

## 金庸小说单本信息熵表（字符模式）

| # | 小说名 | 语料字数 | 一元分词个数 | 一元平均词长 | 一元模型信息熵（比特/词） | 二元分词个数 | 二元平均词长 | 二元模型信息熵（比特/词） | 三元分词个数 | 三元平均词长 | 三元模型信息熵（比特/词） |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 三十三剑客图 | 53399 | 2765 | 1.0 | 9.67383 | 33571 | 2.0 | 4.833181 | 48807 | 3.0 | 0.98232 |
| 2 | 书剑恩仇录 | 435906 | 3938 | 1.0 | 9.472801 | 132520 | 2.0 | 5.77677 | 307362 | 3.0 | 2.389521 |
| 3 | 侠客行 | 309887 | 3228 | 1.0 | 9.154881 | 91480 | 2.0 | 5.58955 | 212851 | 3.0 | 2.368499 |
| 4 | 倚天屠龙记 | 818422 | 3936 | 1.0 | 9.395247 | 188663 | 2.0 | 6.020465 | 515464 | 3.0 | 2.804452 |
| 5 | 天龙八部 | 1021603 | 4253 | 1.0 | 9.407149 | 220114 | 2.0 | 6.124654 | 623797 | 3.0 | 2.948488 |
| 6 | 射雕英雄传 | 767899 | 4415 | 1.0 | 9.448709 | 191252 | 2.0 | 6.060441 | 503227 | 3.0 | 2.756466 |
| 7 | 白马啸西风 | 57437 | 2157 | 1.0 | 8.913669 | 25202 | 2.0 | 4.581347 | 45324 | 3.0 | 1.619834 |
| 8 | 碧血剑 | 416533 | 3954 | 1.0 | 9.45911 | 131965 | 2.0 | 5.864349 | 301439 | 3.0 | 2.349298 |
| 9 | 神雕侠侣 | 827823 | 3921 | 1.0 | 9.327852 | 189553 | 2.0 | 6.114244 | 531946 | 3.0 | 2.887035 |
| 10 | 笑傲江湖 | 824948 | 3819 | 1.0 | 9.209128 | 172468 | 2.0 | 5.896344 | 486098 | 3.0 | 2.869852 |
| 11 | 越女剑 | 13663 | 1420 | 1.0 | 8.82653 | 8568 | 2.0 | 3.641473 | 11978 | 3.0 | 0.910077 |
| 12 | 连城诀 | 194542 | 2999 | 1.0 | 9.173805 | 67150 | 2.0 | 5.397629 | 143673 | 3.0 | 2.156446 |
| 13 | 雪山飞狐 | 113428 | 2724 | 1.0 | 9.205488 | 47260 | 2.0 | 5.161323 | 89260 | 3.0 | 1.757219 |
| 14 | 飞狐外传 | 375225 | 3420 | 1.0 | 9.310561 | 112083 | 2.0 | 5.752678 | 262785 | 3.0 | 2.406082 |
| 15 | 鸳鸯刀 | 29290 | 1912 | 1.0 | 9.036094 | 16358 | 2.0 | 4.214057 | 24918 | 3.0 | 1.116929 |
| 16 | 鹿鼎记 | 1022993 | 4212 | 1.0 | 9.291481 | 210494 | 2.0 | 5.985501 | 597000 | 3.0 | 2.949062 |

# 结论

1. 中英文对比：本实验中计算出在分词模式下，一元词的信息熵为12.17bit，在字符模式下，单个字的信息熵为9.53bit，而论文中提出的模型计算出的英文单个词信息熵为1.75bit，这一点说明，无论是以字为单位还是词为单位，中文的信息熵都要比英文高，即中文混乱程度更高，包含的信息量也更多。
2. N-gram对比：在分词模式下，一元词的信息熵为12.17bit，二元词的信息熵为6.96bit，三元词的信息熵为2.30bit，可以看出随着N的增大，平均信息熵在下降，这是因为N取值越大，通过分词后得到的文本中词组的分布就越简单，N越大使得固定的词数量越多，固定的词能减少由字或者短词打乱文章的机会，使得文章变得更加有序，减少了由字组成词和组成句的不确定性，也即减少了文本的信息熵，符合实际认知。但在分字符模式

下，一元字符的信息熵为9.53bit，二元字符的信息熵为6.73bit，三元字符的信息熵为3.59bit，也符合上述规律。

3. 两种模式对比：分词模式在一元和二元下得到的信息熵比分字符模式更高，而在三元时得到的信息熵比分字符模式更低，由分词个数我们可以发现其背后的原因，在分词模式下，一元词的数目远大于一元字符的数目，这是由于不同字符组合成词的方式非常多样，这就导致了词这一单位信息熵的增大，而当N增大时，由于词与词之间常常有一些固定的组合搭配，会形成一些固定的意思，这就使得整体的信息熵降了下来，而对于字符，虽然也有同样的趋势，字符间的组合往往比词语的组合要更加随机，因此在三元时，分字符模式的信息熵要比分词模式高。

## 不足

在本实验中计算信息熵时，使用的训练集和测试集相同，即未满足论文中提到的语言模型M必须在没有测试样本相关知识的前提下构建，这可能导致对交叉熵产生影响，如果我们仅限于测试样本中出现的字符，而不是所有字符，交叉熵会显著降低，如果我们使用测试样本的实际词汇表，交叉熵还会降低。改进的措施是划分训练集和测试集，按照论文中提出的引入unknown_token，进行相应的处理。

## 参考文档


中文信息熵的计算_DiliDili_Q的博客

中文平均信息熵 | 穆殿下

中文平均信息熵的计算_zxycurry的博客