Report of Deep Learning for Natural Language Processing

曹思远

3203998114@qq.com

Abstract

本文档共分为两个部分，第一部分是通过提供的中文语料库来验证 Zipf's Law.；第二部分是通过阅读 Entropy Of English，分别计算中文(以词和字为单位)的平均信息熵。对于第一部分作业，是利用 Python 及其 jieba 库对作业提供的中文语料库进行精准分词，后通过图像显示来验证 Zipf's Law.定律；对于第二部分作业，同样是利用 Pathon 及其 jieba 库对作业提供的中文语料库进行精准分词，后依次通过 1-gram、2-gram 与 3-gram 统计语言模型针对上述中文语料库分别以汉字和词语两个方面计算中文的信息熵，其中针对 2-gram 与 3-gram 计算的是条件信息熵。

Introduction

Zipf's Law

Zipf's Law（齐夫定律）是由美国语言学家 George Zipf 提出的经验定律，描述了自然语言中单词使用频率与它们在频率排序表中的排名之间的关系。Zipf's Law 有以下特点：

1.在一个给定的自然语言语料库中，排名第二的单词的出现频率大约是排名第一的单词的频率的一半，排名第三的单词的频率大约是排名第一的单词的频率的三分之一，以此类推。换句话说，单词的使

用频率与其在频率排序表中的排名成反比关系。

2.这个定律适用于各种自然语言，包括英语、中文等。

3.Zipf's Law 的应用不仅局限于自然语言，还可以在其他领域如城市人口规模、公司规模等方面找到类似的规律。

4.尽管 Zipf's Law 是一个经验定律，但它对于自然语言处理、信息检索等领域具有重要意义，可以帮助我们理解和分析文本数据中单词的分布规律。

总的来说，Zipf's Law 提供了一种简洁而有力的描述自然语言中单词使用频率的规律，对于文本数据的分析和处理具有重要的指导意义。

信息熵

信息熵是信息论中的一个重要概念，用于衡量随机变量的不确定性或信息量。信息熵由克劳德·香农（Claude Shannon）在他的信息论中首次提出。

信息熵的定义如下：

对于一个离散随机变量 X，其取值集合为$\{ x_1, x_2, ..., x_n \}$，概率分布为 P(X)，信息熵 H(X)定义为：

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log p(x_i)$$

其中，$p(x_i)$ 是随机变量 X 取值为x_i的概率。

N-gram 语言模型

根据大数定理，当统计量足够大的时候，词、二元词组、三元词组出现的概率大致等于其出现的频率。

则有一元模型的信息熵计算公式为：

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

其中 P(x)可近似于每个词在语料库中的出现频率

二元模型的条件信息熵计算公式为：

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

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率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。

三元模型的条件信息熵计算公式为：

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

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率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

Methodology

M1:通过中文语料库来验证 Zipf's Law

1.文本的加载

```
def load_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()
    return text
```

2.中文语料的预处理

数据库中的语料是 txt 格式的，其中包含得有部分代码和无用的

符号，首先需要对数据进行一下预处理，其中包括删除隐藏符号（如换行符、分页符等）

```
def preprocess_text(text):
    # 去除标点符号和特殊字符
    text = re.sub( pattern: r'[^\w\s]',  repl: '', text)
    # 将文本转换为小写
    # text = text.lower()
    return text
```

3.利用 jieba 精准分词并计算词的频率

```
def calculate_word_frequency(text):
    # 中文分词
    words = jieba.lcut(text)
    # 计算词频
    word_freq = Counter(words)
    return word_freq
```

4.结果可视化

通过 matplotlib 将词频-排名可视化，均使用 log 化，方便验证其乘积为常数。仅取排名前 5000 的显示

```
def zipfs_law_plot(word_freq):
    # 根据词频排序
    sorted_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)
    # 提取前5000个词及其频率
    top_words = sorted_freq[:5000]
    ranks = range(1, len(top_words) + 1)
    freqs = [pair[1] for pair in top_words]
    # 对数化处理
    log_ranks = np.log(ranks)
    log_freqs = np.log(freqs)
    # 绘制曲线
    plt.figure(figsize=(10, 6))
    plt.plot( *args: log_ranks, log_freqs, marker='o')
    plt.xlabel('Log Rank')
    plt.ylabel('Log Frequency')
    plt.title('Zipf\'s Law')
    plt.grid(True)
    plt.show()
```

M2:基于 N-gram 模型的中文信息熵计算

1. 读取文件夹中的所有 txt 文件并整合在一起

```
def read_folder(folder_path):
    all_text = ''
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.txt'):
            file_path = os.path.join(folder_path, file_name)
            with open(file_path, 'r', encoding='utf-8') as file:
                text = file.read()
                all_text += text
    return all_text
```

2.数据库中的语料是 txt 格式的，其中包含得有部分代码和无用的符号，首先需要对数据进行一下预处理，其中包括删除隐藏符号（如换行符、分页符等）；删除文中的标点符号；删除空格。

```
def preprocess_text(text):
    # 删除隐藏符号（如换行符、分页符等）
    text = re.sub( pattern: r'\s+', repl: '', text)

    # 删除文中的标点符号
    text = re.sub( pattern: r'[^\u4e00-\u9fa5]', repl: '', text)

    # 删除空格
    text = text.replace( _old: ' ', _new: '')

    return text
```

3.利用 jieba 精准分词，生成字的库和词库

```
# 使用jieba分词
tokens_character = list(preprocessed_text)  # 以中文字为单位的tokens
tokens_word = jieba.lcut(preprocessed_text)  # 以词语为单位的tokens
```

4.1-gram 计算信息熵；2-gram、3-gram 计算条件信息熵

```python
def calculate_entropy(tokens):
    token_count = len(tokens)
    word_counts = Counter(tokens)
    entropy = 0
    for count in word_counts.values():
        probability = count / token_count
        entropy += -probability * math.log(probability, base: 2)
    return entropy
```

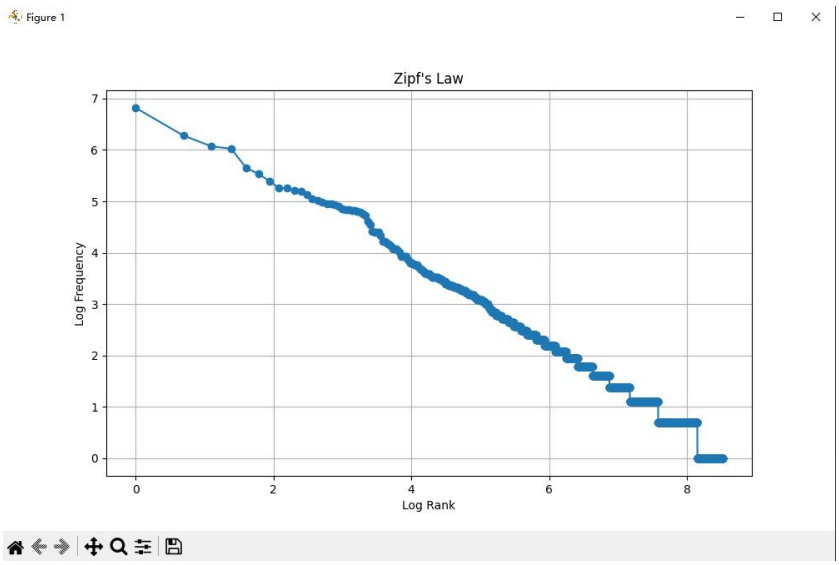
```python
2 usages
def calculate_conditional_entropy(tokens, n):
    conditional_counts = Counter(zip(*[tokens[i:] for i in range(n)]))
    token_count = len(tokens)
    conditional_entropy = 0
    for gram, count in conditional_counts.items():
        probability = count / token_count
        conditional_entropy += -probability * math.log(probability, base: 2)
    return conditional_entropy
```

5.数据显示

输出以字和词为单元的排名频率前 10 的；并且输出相应的信息熵。

Experimental Studies

M1:通过中文语料库来验证 Zipf's Law



可以从词频-排名的对数坐标下，看到近似成一条直线，两者是成反比的

M2:基于 N-gram 模型的中文信息熵计算

字的 1-gram 信息熵：

```
Entropy for all files (Character unit): 9.52733866871124

Top 10 most frequent words in 1-gram model (Character unit):
一: 139397
不: 134150
的: 121671
是: 112708
了: 111927
道: 111057
人: 84305
他: 73575
这: 68993
我: 67000
```

字的 2-gram 条件信息熵：

```
Conditional Entropy for 2-gram model (Character unit): 16.253979398606898

Top 10 most frequent words in 2-gram model (Character unit):
('说', '道'): 13571
('了', '一'): 12267
('一', '个'): 10615
('道', '你'): 10420
('自', '己'): 10356
('小', '宝'): 9961
('韦', '小'): 9869
('也', '不'): 9336
('道', '我'): 8593
('笑', '道'): 8159
```

字的 3-gram 条件信息熵：

```
Conditional Entropy for 3-gram model (Character unit): 20.20531510209588

Top 10 most frequent words in 3-gram model (Character unit):
('韦', '小', '宝'): 9833
('令', '狐', '冲'): 5890
('张', '无', '忌'): 4667
('的', '一', '声'): 3508
('袁', '承', '志'): 3037
('小', '宝', '道'): 2421
('小', '龙', '女'): 2134
('陈', '家', '洛'): 2115
('石', '破', '天'): 1818
('不', '由', '得'): 1812
```

词的 1-gram 信息熵：

```
Entropy for all files (Word unit): 12.179017650340606

Top 10 most frequent words in 1-gram model (Word unit):
的: 115582
了: 104504
他: 64754
是: 64295
道: 58557
我: 57504
你: 56676
在: 43675
也: 32602
这: 32235
Conditional Entropy for 2-gram model (Word unit): 19.129359760026244
```

词的 2-gram 条件信息熵：

```
Conditional Entropy for 2-gram model (Word unit): 19.129359760026244

Top 10 most frequent words in 2-gram model (Word unit):
('道', '你'): 5825
('叫', '道'): 5033
('道', '我'): 5012
('笑', '道'): 4266
('听', '得'): 4218
('都', '是'): 3923
('了', '他'): 3784
('他', '的'): 3509
('也', '是'): 3212
('的', '一声'): 3127
```

词的 3-gram 条件信息熵：

```
Conditional Entropy for 3-gram model (Word unit): 21.42895520302674

Top 10 most frequent words in 3-gram model (Word unit):
('只', '听', '得'): 1615
('忽', '听', '得'): 1138
('站', '起身', '来'): 733
('哼', '了', '一声'): 581
('笑', '道', '你'): 576
('吃', '了', '一惊'): 539
('啊', '的', '一声'): 525
('点', '了', '点头'): 505
('说', '到', '这里'): 476
('了', '他', '的'): 461
```

对比 1-Gram、2-Gram、3-Gram 三种语言模型得到的结果可以看出，在考虑条件信息熵后，2-Gram、3-Gram 的熵值在增大，可能是因为存在一些两个词或三个词的固定搭配，这样在算相应的条件概率

时值可能会增大。

Conclusion

综上所述，本作业利用 Python 及其提 jieba 库对提供的中文语料库进行了分词与词频统计，验证了中文下的 Zipf's Law 定理；其次基于 1-gram、2-gram 和 3-gram 统计语言模型在上述语料库中以字与词两个方面计算了中文的信息熵，并分析两个实验结果。

Referances

1.An Estimate of an Upper Bound for the Entropy of English

2. 深度学习与自然语言处理第一次作业——中文平均信息熵的计算_中文信源的独立熵要算标点符号吗-CSDN 博客

3. 深度学习与自然语言处理作业第一次作业——中文平均信息熵计算_自然语言平均每字符信息熵-CSDN 博客