Lecture 4 超基础复习笔记 - Zipf定律（齐夫定律）

零、先理解问题（什么是Zipf定律？）

一个神奇的现象

想象你统计一本书中所有单词的出现次数：

```
第1名（最常见）："the" 出现 10,000 次
第2名："of"  出现  5,000 次  （大约是第1名的一半）
第3名："and" 出现  3,333 次  （大约是第1名的1/3）
第4名："a"   出现  2,500 次  （大约是第1名的1/4）
...
```

发现规律了吗？

```
第N名的词出现次数 ≈ 第1名出现次数 / N
```

这就是 **Zipf定律**！

用数学公式表达

Zipf定律：

```
频率 ∝ 1 / 排名

或写成:
f(r) = C / r

其中:
- f(r) = 排名为r的词的频率
- r = 排名 (1, 2, 3, ...)
- C = 常数
- ∝ 表示"成正比"
```

一、为什么Zipf定律重要?

1. 语言的普遍规律

惊人的事实: - 英语符合Zipf定律 - 中文符合Zipf定律 - 其他几乎所有语言都符合!

不仅仅是语言: - 城市人口分布 - 公司规模分布 - 网站访问量分布

都遵循类似的幂律!

2. 实际应用

文本压缩:

```
常见词"the"出现10,000次
只需要用短编码 (如: 1bit)

罕见词"antidisestablishmentarianism"出现1次
用长编码也没关系
```

搜索引擎:

只需索引最常见的135个词
就能覆盖50%的文本内容

二、Brown语料库的例子（真实数据）

排名	单词	频率	占比
1	the	69,971	6.98%
2	of	36,411	3.63%
3	and	28,852	2.88%
10	in	21,341	2.13%
20	at	8,893	0.89%
100	way	1,460	0.15%

观察：

```
第1名频率 / 第2名频率 = 69,971 / 36,411 ≈ 1.92 ≈ 2
第1名频率 / 第3名频率 = 69,971 / 28,852 ≈ 2.42 ≈ 3
```

接近Zipf定律的预测！

三、实验任务（用Moby Dick验证）

步骤概览

1. 读取文本文件（Moby Dick小说）
2. 统计每个单词的出现次数
3. 按频率降序排序
4. 绘制三种图表验证Zipf定律

四、代码详细讲解（逐行解释）

第1部分：统计词频

```
import re

wordRE = re.compile('\w+')
wdcounts = {}

for filename in filenames:
    with open(filename) as infs:
        for line in infs:
            for wd in wordRE.findall(line.lower()):
                if wd not in wdcounts:
                    wdcounts[wd] = 0
                wdcounts[wd] += 1
```

逐行解释

1. 正则表达式

```
wordRE = re.compile('\w+')
```

`\w` 是什么？ - `\w` = word character（单词字符） - 匹配：字母、数字、下划线 - 等价于：`[A-Za-z0-9_]`

`+` 的作用： - 至少匹配1次，可以多次 - `\w+` = 一个或多个单词字符

例子：

```
text = "Hello, world! 123"
wordRE.findall(text)
# 结果: ['Hello', 'world', '123']
```

为什么不匹配逗号和感叹号？ - 因为它们不是`\w`（不是字母、数字、下划线）

---

2. 创建空字典

```
wdcounts = {}
```

- 用来存储：`{单词：出现次数}`

---

3. 读取文件

```
for filename in filenames:
    with open(filename) as infs:
        for line in infs:
```

执行流程：

```
外层循环：遍历每个文件
  内层循环：遍历文件的每一行
```

---

4. 提取单词并转小写

```
for wd in wordRE.findall(line.lower()):
```

`line.lower()` 的作用：

```
line = "The CAT"
line.lower()  # "the cat"
```

为什么要转小写？

```
不转小写：
  "The": 100次
  "the": 500次  ← 其实是同一个词！

转小写后：
  "the": 600次  ← 正确！
```

---

5. 更新计数

```
if wd not in wdcounts:
    wdcounts[wd] = 0
wdcounts[wd] += 1
```

执行示例：

```
# 处理句子: "the cat and the dog"

# 词1: "the"
wd = "the"
wdcounts = {}
wd not in wdcounts  # True
wdcounts["the"] = 0
wdcounts["the"] += 1  # 变成1
# wdcounts = {"the": 1}

# 词2: "cat"
wd = "cat"
wdcounts["cat"] = 0
wdcounts["cat"] += 1
# wdcounts = {"the": 1, "cat": 1}

# 词3: "and"
wdcounts["and"] = 1
# wdcounts = {"the": 1, "cat": 1, "and": 1}

# 词4: "the" (第二次出现)
wd = "the"
wd not in wdcounts  # False (已经存在)
# 不执行 wdcounts["the"] = 0
wdcounts["the"] += 1  # 1变成2
# wdcounts = {"the": 2, "cat": 1, "and": 1}
```

第2部分：排序

```
words = sorted(wdcounts, reverse=True, key=lambda v:wdcounts[v])
freqs = [wdcounts[w] for w in words]
```

`sorted()` 详解

假设：

```
wdcounts = {
    'the': 1000,
    'cat': 50,
    'dog': 100,
    'a': 800
}
```

排序过程：

```
words = sorted(wdcounts,
               reverse=True,
               key=lambda v:wdcounts[v])
```

参数说明： - `wdcounts` - 要排序的字典（实际上排序的是键） - `reverse=True` - 降序（从大到小） - `key=lambda v:wdcounts[v]` - 按值（频率）排序

`lambda` 函数解释：

```
# lambda是匿名函数的简写

# 完整写法：
def get_freq(word):
    return wdcounts[word]

# lambda简写：
lambda v: wdcounts[v]
```

排序执行：

```
比较"the"和"cat":
  key("the") = wdcounts["the"] = 1000
  key("cat") = wdcounts["cat"] = 50
  1000 > 50，所以"the"排前面

所有比较完成后（降序）:
words = ['the', 'a', 'dog', 'cat']
         1000   800  100   50
```

列表推导式

```
freqs = [wdcounts[w] for w in words]
```

展开理解：

```
# 等价于:
freqs = []
for w in words:
    freqs.append(wdcounts[w])

# 结果:
# words = ['the', 'a', 'dog', 'cat']
# freqs = [1000, 800, 100, 50]
```

用途： - `words` - 单词列表（按频率排序） - `freqs` - 对应的频率列表

第3部分：打印统计信息

```
print('TYPES: ', len(words))
print('TOKENS:', sum(freqs))

topN = 200
for wd in words[:topN]:
    print(wd, ':', wdcounts[wd])
```

术语解释：

TYPES（类型数）= 不同单词的数量

```
len(words)  # 有多少个不同的单词

例如：
"the cat and the dog"
TYPES = 4   (the, cat, and, dog)
```

TOKENS（标记数）= 单词出现的总次数

```
sum(freqs)  # 所有频率相加

例如：
"the cat and the dog"
TOKENS = 5   (5个单词)
```

打印前200个词：

```
words[:topN]  # 列表切片，取前200个
```

五、绘图部分（重点！）

图1：频率 vs 排名

```
import pylab as p

ranks = range(1, len(freqs)+1)

p.figure()
p.plot(ranks, freqs)
p.title('freq vs rank')
```

`range()` 详解

```
ranks = range(1, len(freqs)+1)
```

假设 `len(freqs) = 10000`：

```
range(1, 10001)
# 生成: 1, 2, 3, ..., 9999, 10000
```

为什么从1开始？ - 排名从第1名开始（不是第0名）

为什么是 `len(freqs)+1`？

```
range(1, 5)  # 生成: 1, 2, 3, 4 (不包括5)

所以要包括len(freqs)，需要len(freqs)+1
```

`pylab` 绘图

基本绘图步骤：

```
import pylab as p

# 1. 创建新图
p.figure()

# 2. 绘制数据
p.plot(X, Y)

# 3. 添加标题
p.title('图表标题')

# 4. 显示图表
p.show()
```

`p.plot(ranks, freqs)` 的含义：

```
# 绘制点：
# (ranks[0], freqs[0])  → (1, 1000)
# (ranks[1], freqs[1])  → (2, 500)
# (ranks[2], freqs[2])  → (3, 333)
# ...
```

结果图形：

```
频率 ↑
     |
1000|*
     |
 500|  *
     |
 333|    *
     |      *
     |        ***
     |           ********
     |____________________→ 排名
        1  2  3  4  5...
```

特点：陡降曲线（少数词频率很高，大多数词频率很低）

图2：累积频率 vs 排名

```
cumulative = list(freqs)  # 复制列表

for i in range(len(cumulative) - 1):
    cumulative[i + 1] += cumulative[i]

p.figure()
p.plot(ranks, cumulative)
p.title('cumulative freq vs rank')
```

累积频率是什么?

原始频率：

```
排名:  1     2     3     4     5
频率: 1000  500   333   250   200
```

累积频率：

```
排名:  1     2      3       4       5
累积: 1000  1500   1833    2083    2283
       ↑     ↑      ↑       ↑       ↑
      1000  1000   1000    1000    1000
           + 500  + 500   + 500   + 500
                  + 333   + 333   + 333
                          + 250   + 250
                                  + 200
```

计算过程：

```
cumulative[0] = freqs[0] = 1000
cumulative[1] = freqs[1] + cumulative[0] = 500 + 1000 = 1500
cumulative[2] = freqs[2] + cumulative[1] = 333 + 1500 = 1833
cumulative[3] = freqs[3] + cumulative[2] = 250 + 1833 = 2083
```

代码详解

```
cumulative = list(freqs)  # 复制列表
```

为什么要复制?

```
# 错误做法:
cumulative = freqs  # 只是引用，不是复制!
cumulative[0] = 999
print(freqs[0])  # 也变成999了! ❌

# 正确做法:
cumulative = list(freqs)  # 创建新列表
cumulative[0] = 999
print(freqs[0])  # 还是原来的值✅
```

```
for i in range(len(cumulative) - 1):
    cumulative[i + 1] += cumulative[i]
```

为什么是 `len(cumulative) - 1`?

假设 `len(cumulative) = 5`:

```
range(4)  # 0, 1, 2, 3

i=0: cumulative[1] += cumulative[0]
i=1: cumulative[2] += cumulative[1]
i=2: cumulative[3] += cumulative[2]
i=3: cumulative[4] += cumulative[3]

如果用 range(5):
i=4: cumulative[5] += cumulative[4]  ❌ 越界!
```

执行示例:

```
cumulative = [1000, 500, 333, 250, 200]

# i=0:
cumulative[1] += cumulative[0]
# cumulative[1] = 500 + 1000 = 1500
# cumulative = [1000, 1500, 333, 250, 200]

# i=1:
cumulative[2] += cumulative[1]
# cumulative[2] = 333 + 1500 = 1833
# cumulative = [1000, 1500, 1833, 250, 200]

# i=2:
cumulative[3] += cumulative[2]
# cumulative[3] = 250 + 1833 = 2083
# cumulative = [1000, 1500, 1833, 2083, 200]

# i=3:
cumulative[4] += cumulative[3]
# cumulative[4] = 200 + 2083 = 2283
# cumulative = [1000, 1500, 1833, 2083, 2283]
```

累积图的意义:

```
累积频率 ↑
        |
  总词数 |_________________
        |           /
        |         /
        |       /
        |     /
        |   /
        |/
        |____________________→ 排名
```

前135个高频词就能覆盖50%的文本!

图3：log-log 图（验证Zipf定律）

```
logfreqs = [p.log(freq) for freq in freqs]
logranks = [p.log(rank) for rank in ranks]

p.figure()
p.plot(logranks, logfreqs)
p.title('log-freq vs log-rank')
p.savefig('log1.png')
```

为什么要取对数?

Zipf定律数学形式：

```
频率 = C / 排名
freq = C / rank
```

两边取对数：

```
log(freq) = log(C / rank)
          = log(C) - log(rank)
```

这是一条直线！

```
y = -x + log(C)
↑   ↑    ↑
log(freq) 斜率=-1  截距
```

如果Zipf定律成立： - log-log图应该是**一条直线** - 斜率约为 **-1**

代码详解

```
logfreqs = [p.log(freq) for freq in freqs]
```

`p.log()` **是什么？** - pylab的对数函数 - 默认是自然对数（ln，底数e）

列表推导式展开：

```
logfreqs = []
for freq in freqs:
    logfreqs.append(p.log(freq))

# 例子：
freqs = [1000, 500, 250, 125]
logfreqs = [log(1000), log(500), log(250), log(125)]
         ≈ [6.91, 6.21, 5.52, 4.83]
```

`p.savefig()` - 保存图片

```
p.savefig('log1.png')
```

作用：将图表保存为PNG图片文件

为什么要保存？ - 可以放到报告里 - 可以分享给别人 - 不用每次重新运行代码

六、完整执行示例（手工模拟）

输入数据（简化版）

```
文本: "the cat and the dog and the bird"
```

步骤1：统计词频

```
wdcounts = {
    'the': 3,
    'cat': 1,
    'and': 2,
    'dog': 1,
    'bird': 1
}
```

步骤2：排序

```
# 按频率降序
words = ['the', 'and', 'cat', 'dog', 'bird']
freqs = [3, 2, 1, 1, 1]
```

步骤3：计算排名

```
ranks = [1, 2, 3, 4, 5]
```

步骤4：绘制freq vs rank

```
频率 ↑
   3 |*
   2 |  *
   1 |    * * *
     |___________→ 排名
       1 2 3 4 5
```

步骤5：计算累积频率

```
cumulative = [3, 5, 6, 7, 8]
# 计算过程：
# cumulative[0] = 3
# cumulative[1] = 3 + 2 = 5
# cumulative[2] = 5 + 1 = 6
# cumulative[3] = 6 + 1 = 7
# cumulative[4] = 7 + 1 = 8
```

步骤6：计算log值

```
import math

logranks = [log(1), log(2), log(3), log(4), log(5)]
         = [0, 0.69, 1.10, 1.39, 1.61]

logfreqs = [log(3), log(2), log(1), log(1), log(1)]
         = [1.10, 0.69, 0, 0, 0]
```

步骤7：绘制log-log图

```
log(freq) ↑
      1.1 |*
      0.7 |  *
        0 |    * * *
          |__________→ log(rank)
            0 0.7 1.1 1.4 1.6
```

七、为什么是直线？（数学证明）

Zipf定律

```
频率 = C / 排名

用符号表示：
f = C / r
```

取对数

```
log(f) = log(C / r)
       = log(C) - log(r)
```

整理成直线方程

```
y = mx + b 的形式

log(f) = -log(r) + log(C)
  ↑       ↑  ↑      ↑
  y    斜率m x    截距b

所以:
- y = log(频率)
- x = log(排名)
- 斜率 m = -1
- 截距 b = log(C)
```

这就是一条斜率为-1的直线！

八、实际结果（Moby Dick）

统计数据

```
TYPES（不同单词数）: 约 17,000 个
TOKENS（总单词数）: 约 215,000 个

前20个高频词:
1. the: 14,431次  (6.7%)
2. of: 6,609次    (3.1%)
3. and: 6,430次   (3.0%)
4. a: 4,736次     (2.2%)
5. to: 4,625次    (2.1%)
...
```

log-log图结果

观察： - 大部分点接近一条直线 - 斜率约为 -1 - **证实了Zipf定律！**

偏差： - 最高频的几个词略高于直线（超常用词） - 最低频的词偏离较大（样本量小，统计误差）

九、Zipf定律的意义

1. 语言的经济性

> 少数词承担大部分工作：
> - 前100个词 → 覆盖50%的文本
> - 前1000个词 → 覆盖80%的文本

为什么？ - 高效沟通 - 减少认知负担

2. 长尾效应

```
频率 ↑
     |*
     | *
     |  *
     |    ********* ← "长尾"（大量低频词）
     |_____________________→ 排名
```

影响： - 搜索引擎需要索引大量低频词 - 机器学习需要处理稀有词

3. 幂律的普遍性

其他符合Zipf定律的现象： - 城市人口：纽约人口 ≈ 2×洛杉矶人口 - 公司规模：最大公司 ≈ 2×第二大公司 - 网站流量：Google访问量 » 其他网站

共同特征： - "富者愈富" - 马太效应 - 幂律分布

十、纸笔考试重点

必须掌握

1. ✅ **Zipf定律的定义**

2. 频率 ∝ 1/排名
3. 第N名频率 ≈ 第1名频率 / N
4. ✅ **log-log图的意义**
5. 为什么取对数
6. 为什么是直线
7. 斜率应该是多少
8. ✅ **累积频率的计算**
9. 前N个词的频率总和
10. 代码实现方式
11. ✅ **词频统计**
12. 用字典存储
13. 排序方法
14. TYPES vs TOKENS

可能的考题

题型1：计算验证

题目：给定词频数据，判断是否符合Zipf定律

```
排名   单词    频率
1      the     1000
2      of      500
3      and     333
4      a       250
```

问：是否符合Zipf定律?

▶ **答案**

题型2：累积频率计算

题目：

> 频率列表：`[100, 50, 30, 20, 10]`
>
> 计算累积频率列表

▶ **答案**

题型3：log变换

题目：为什么Zipf定律在log-log图上是直线？

▶ **答案**

十一、Python绘图基础

基本绘图

```
import pylab as p

# 数据
X = [1, 2, 3, 4, 5]
Y = [2, 4, 6, 8, 10]

# 绘图
p.figure()          # 创建新图
p.plot(X, Y)        # 画点和线
p.title('My Plot')  # 标题
p.xlabel('X轴')     # X轴标签
p.ylabel('Y轴')     # Y轴标签
p.show()            # 显示
```

多个图表

方法1：多个窗口

```
p.figure()  # 图1
p.plot(X, Y1)

p.figure()  # 图2（新窗口）
p.plot(X, Y2)

p.show()
```

方法2：子图

```
p.subplot(211)  # 2行1列，第1个
p.plot(X, Y1)

p.subplot(212)  # 2行1列，第2个
p.plot(X, Y2)

p.show()
```

格式控制

```
p.plot(X, Y, 'ro-')
# 'r' = red (红色)
# 'o' = circle (圆圈标记)
# '-' = solid line (实线)

其他选项:
'b' = blue
'g' = green
'*' = asterisk
'x' = cross
'--' = dashed line
```

十二、记忆口诀

Zipf定律

```
频率反比排名,
第一最多常见,
第二减半出现,
第N除以N倍。
```

log-log图

```
取对数变直线,
斜率负一验证,
偏离说明误差,
大体符合定律。
```

累积计算

> 从前往后加，
> 每项加前项，
> 总和在增长，
> 最后是总数。

十三、常见错误

❌ 错误1：混淆TYPES和TOKENS

```
# 错误理解
text = "the cat the dog"
TYPES = 4  ❌  # 错！有重复

# 正确理解
TYPES = 3  ✅  # 不同的词：the, cat, dog
TOKENS = 4 ✅  # 总共4个词
```

❌ 错误2：排名从0开始

```
# 错误
ranks = range(0, len(freqs))  ❌
# 排名应该从1开始！

# 正确
ranks = range(1, len(freqs)+1)  ✅
```

❌ 错误3：累积计算方向错误

```
# 错误：从后往前加
for i in range(len(cumulative)-1, 0, -1):
    cumulative[i-1] += cumulative[i]  ❌

# 正确：从前往后加
for i in range(len(cumulative) - 1):
    cumulative[i+1] += cumulative[i]  ✅
```

十四、扩展思考

问题1：为什么会有Zipf定律?

可能的解释：

1. **最小努力原则**
2. 人们倾向用最少的词表达最多的意思
3. 常用词复用率高
4. **优先连接模型**
5. 新文本倾向使用已有的常用词
6. "富者愈富"效应
7. **信息论角度**
8. 平衡表达效率和理解成本
9. 最优编码策略

问题2：哪些情况不符合Zipf定律?

特殊文本： - 诗歌（刻意用罕见词） - 技术文档（大量专业术语） - 随机生成的文本

祝你考试顺利！ 🎉

记住： - Zipf定律：频率 $\propto$ 1/排名 - log-log图是验证工具 - 累积频率看覆盖率 - 理解比记公式重要！