NLP 第一次作业技术报告

关于中英文文本的爬取与概率和熵值计算

2025.10.17

尹超
中国科学院大学
University of Chinese Academy of Sciences
2025
2313AI
Template:https://github.com/hzkonor/bubble-template
GitHub 地址:https://github.com/CarterYin/NLP_UCAS_2025

目录

I. Chapter 1 Introduction

本次作业内容如下：

(A). 利用网络爬虫工具从互联网上收集尽量多的英语和汉语文本，完成下面的工作：

① 对收集的样本进行处理，如清洗乱码等。

② 设计并编程实现算法，计算在收集样本上英语字母和单词或汉字的概率和熵（如果有汉语自动分词工具，可以统计计算汉语词汇的概率和熵）。

③ 利用收集的英文文本验证齐夫定律(Zipf's law)。

④ 不断增加样本规模，重新计算上面②③的结果, 进行对比分析。

⑤ 完成一份技术报告，在报告中写明利用什么爬虫工具从哪些网站上收集的样本，如何进行的样本清洗，清洗后样本的规模，在不同样本规模下计算的结果等。实验分析有较大的伸缩空间。

II. Chapter 2 Data Collection and Cleaning

II.1. 爬虫类型和语料来源网站

本次实验中，我采取了定向（垂直）爬虫（Focused Web Crawler），从人民日报和 China Daily 两个网站上分别爬取了中文和英文文本数据。

II.2. 爬虫实现流程与细节

我编写了两个独立的爬虫脚本，分别命名为 rm.py（针对人民日报）和 cd.py（针对 China Daily）。

II.2.1. 爬虫实现流程

两份爬虫脚本都是面向具体电子报网站的定向采集工具，整体流程相近：通过日期驱动生成索引页 URL，逐级解析出文章列表，再抓取正文并累积到单一语料文件；在此基础上又引入 CorpusMonitor，按 2MB/5MB/10MB 阈值自动截取快照，便于规模控制和阶段分析。rm.py 依赖较固定的人民网上版面结构，层层定位到文章正文；cd.py 则针对 China Daily 的 HTML 差异做了更复杂的选择器和清洗逻辑（如多套候选节点、元信息剔除、重复段落去除），以适应版式变化。两者都使用 requests + BeautifulSoup 组合处理静态页面，并在请求层面设置通用的 UA/超时、在抓取循环中加延时，以降低被封风险，属于典型的模板化垂直爬虫实现。

II.2.2. 爬虫实现细节

细节处理方面，爬虫脚本均实现了断点续爬功能，利用本地文件记录已抓取日期，避免重复下载。文本清洗主要包括 HTML 标签剥离、空白字符归一化、非正文内容过滤、乱码去除等，确保语料纯净度。最终的语料文件均采用 UTF-8 编码保存，便于后续处理。

II.3. 爬取结果

爬取结果方面，人民日报从 20241201 爬取到 20250301，China Daily 则从 20241201 爬取到 20250401。分别成功获得了三份不同大小的语料文件，大小为 2MB/5MB/10MB，均为纯文本格式，编码为 UTF-8。检查显示，文本内容较为干净，无乱码，符合后续处理要求。

III. Chapter 3 Probability and Entropy Calculation

在获得清洗后的文本语料后，下一步是计算字母/汉字的概率分布及熵值。具体步骤如下：

III.1. 中文字符的概率与熵值计算

符号筛选与计数：针对中文语料，使用 `compute_rmrb.py` 中的 `iter_cjk_chars` 函数过滤 CJK 统一汉字范围，再用 `collections.Counter` 统计字符频次；针对英文语料，`compute_cd.py` 采用 `iter_english_letters` 过滤 ASCII 字母并按需小写化。

概率与熵计算：对每个字符取频次占比得到概率分布 `p_i`，随后使用香农熵公式 `H = -∑ p_i log₂ p_i` 计算信息熵，并记录字符总数、去重后字符数等指标。

结果展示：脚本支持通过 `--top` 参数输出高频符号明细，包含排名、字符、频次、概率、信息量（bits）与贡献度，便于直观对比。

III.1.1. 以 `data/rmrb_snapshot_2MB.txt` 为例

运行 `python compute_rmrb.py data\rmrb_snapshot_2MB.txt --top 10` 得到如下输出：

```
(nlp) PS E:\homework\nlp\hw1> python compute_entropy.py
data\rmrb_snapshot_2MB.txt --top 10
Corpus: data\rmrb_snapshot_2MB.txt
Total Chinese characters: 602680
Unique Chinese characters: 3571
Shannon entropy: 9.533166 bits

Top characters:
rank char     count    prob      information(bits)  contribution
   1  的      12996  0.021564        5.535253         0.119360
   2  国       7551  0.012529        6.318581         0.079166
   3  中       6743  0.011188        6.481858         0.072521
   4  一       5608  0.009305        6.747762         0.062789
   5  发       4736  0.007858        6.991579         0.054941
   6  新       4110  0.006820        7.196110         0.049074
   7  化       4006  0.006647        7.233086         0.048078
   8  人       3977  0.006599        7.243568         0.047799
   9  和       3966  0.006581        7.247564         0.047693
  10  业       3916  0.006498        7.265868         0.047211
```

可以看到高频汉字集中在“的、国、中、一”等常用词，单个字符的信息量约 5.5～7.3 bits，整体熵值 9.53 bits 与课堂参考数据基本一致。

III.1.2. 当语料扩展到 5MB 与 10MB 快照时

```
(nlp) PS E:\homework\nlp\hw1> python compute_entropy.py
data\rmrb_snapshot_5MB.txt --top 10
Corpus: data\rmrb_snapshot_5MB.txt
Total Chinese characters: 1503549
Unique Chinese characters: 4307
Shannon entropy: 9.639168 bits

Top characters:
```

```
rank char     count    prob      information(bits)  contribution
   1  的      33439  0.022240        5.490696         0.122113
   2  国      16802  0.011175        6.483595         0.072453
   3  中      15766  0.010486        6.575411         0.068949
   4  一      13521  0.008993        6.797026         0.061124
   5  发      11182  0.007437        7.071050         0.052588
   6  人      10897  0.007248        7.108297         0.051518
   7  业       9574  0.006368        7.295034         0.046452
   8  和       9550  0.006352        7.298655         0.046358
   9  在       9487  0.006310        7.308204         0.046113
  10  年       9333  0.006207        7.331815         0.045511
```

```
(nlp) PS E:\homework\nlp\hw1> python compute_entropy.py
data\rmrb_snapshot_10MB.txt --top 10
Corpus: data\rmrb_snapshot_10MB.txt
Total Chinese characters: 3006557
Unique Chinese characters: 4735
Shannon entropy: 9.631801 bits

Top characters:
rank char     count    prob      information(bits)  contribution
   1  的      66909  0.022254        5.489768         0.122171
   2  国      32913  0.010947        6.513311         0.071302
   3  中      30465  0.010133        6.624816         0.067128
   4  一      26290  0.008744        6.837454         0.059788
   5  人      22374  0.007442        7.070145         0.052614
   6  发      21741  0.007231        7.111550         0.051425
   7  业      20417  0.006791        7.202198         0.048909
   8  和      19202  0.006387        7.290712         0.046564
   9  在      18814  0.006258        7.320162         0.045807
  10  年      18745  0.006235        7.325463         0.045672
```

`compute_rmrb.py` 输出的熵值分别为 9.639 bits 与 9.632 bits，去重字符数增至 4307 与 4735，但高频字仍围绕“的、国、中、一、人、发”等词汇，所占概率与贡献度变化不大。这说明在人民日报的报道语域中，单字符层面的分布在几兆规模下迅速收敛，捕捉到的多为功能词与核心名词。

III.2. 英文字母的概率与熵值计算

III.2.1. 纵观所有结果

```
(nlp) PS E:\homework\nlp\hw1> python compute_cd.py data\cd_snapshot_2MB.txt --
top 10
Corpus: data\cd_snapshot_2MB.txt
Total letters: 1706620
Unique letters: 26
Shannon entropy: 4.162821 bits

Top letters:
rank char    count    prob       information(bits) contribution
   1  e     200370  0.117408        3.090403         0.362837
   2  t     149595  0.087656        3.512008         0.307848
```

```
   3  i     144565  0.084708        3.561352         0.301676
   4  a     143569  0.084125        3.571326         0.300437
   5  n     141317  0.082805        3.594135         0.297613
   6  o     125017  0.073254        3.770946         0.276237
   7  s     111119  0.065111        3.940964         0.256598
   8  r     107151  0.062786        3.993425         0.250729
   9  h      73296  0.042948        4.541264         0.195038
  10  l      68587  0.040189        4.637063         0.186358
```

```
(nlp) PS E:\homework\nlp\hw1> python compute_cd.py data\cd_snapshot_5MB.txt --
top 10
Corpus: data\cd_snapshot_5MB.txt
Total letters: 4264958
Unique letters: 26
Shannon entropy: 4.162419 bits

Top letters:
rank char    count    prob      information(bits)  contribution
   1  e     500480  0.117347        3.091147         0.362737
   2  t     373307  0.087529        3.514097         0.307585
   3  i     361563  0.084775        3.560213         0.301818
   4  a     360869  0.084613        3.562984         0.301473
   5  n     352718  0.082701        3.595944         0.297390
   6  o     311461  0.073028        3.775408         0.275710
   7  s     279295  0.065486        3.932670         0.257535
   8  r     268068  0.062854        3.991861         0.250903
   9  h     182403  0.042768        4.547330         0.194479
  10  l     170040  0.039869        4.648585         0.185335
```

```
(nlp) PS E:\homework\nlp\hw1> python compute_cd.py data\cd_snapshot_10MB.txt
--top 10
Corpus: data\cd_snapshot_10MB.txt
Total letters: 8532582
Unique letters: 26
Shannon entropy: 4.162832 bits

Top letters:
rank char count prob information(bits) contribution
   1  e    1000965  0.117311        3.091591         0.362677
   2  t     746638  0.087504        3.514502         0.307534
   3  i     723501  0.084793        3.559915         0.301855
   4  a     721101  0.084511        3.564709         0.301259
   5  n     704407  0.082555        3.598501         0.297074
   6  o     619315  0.072582        3.784237         0.274669
   7  s     562327  0.065903        3.923501         0.258572
   8  r     537203  0.062959        3.989443         0.251171
   9  h     362041  0.042430        4.558757         0.193430
  10  l     341302  0.040000        4.643862         0.185754
```

III.2.2. 结果分析

对于英文语料，`compute_cd.py` 在不同快照下统计到的 26 个字母概率基本一致，熵值始终落在 4.162 4.163 bits 区间。由于脚本默认将大小写折叠为小写字符，分布主要反映了标准英文的基础字母频率，高频顺序呈现出 `e > t > i/a > n > o` 的经典“ETAOIN”形态。

在 2MB 样本中，总字母数约 170 万，`e` 单字母的概率达到 0.117，单独贡献 0.36 bits；中段的 `o` 、`s` 、`r` 探测到的概率持续落在 0.06 0.07 范围内，对整体熵的贡献维持在 0.25 0.28 bits；而尾部如 `h` 、`l` 的概率则在 0.04 左右，信息量上升到 4.5 bits 以上，但贡献度显著低于主频字母。随着规模扩大到 5MB 与 10MB，总字母数依次增至 4.26M 与 8.53M，各字母的概率、信息量与贡献度几乎未发生肉眼可察的变化，说明英语单字母分布在百万级样本中已经充分体现，后续数据更多是在降低统计噪声。

与中文语料相比，英文熵值更低（约 4.16 bits vs. 9.6 bits），这反映出 26 个字母构成的符号集更紧凑、平均概率差异更大；而中文汉字数量庞大且长尾效应明显，使得熵值更高。两种语料在扩大规模后都显示出稳定的单字符分布，但中文在扩展快照时去重字符仍持续增长，表明长期仍有大量低频汉字被逐步纳入，而英文字母集合在初始阶段就已完整覆盖。

综上，英语语料的单字符熵稳定性与经典语言学结论高度一致，也验证了脚本的处理逻辑。若想进一步探究版面、主题或时间段的差异，需要引入双字母或更高阶的 n 元模型，或结合词级统计与停用词过滤，才能捕捉到语义层面对分布的微妙变化。

III.3. 英文单词的概率与熵值计算

在英文单词层面，我对 `compute_cd.py` 进行了调整，命名新的脚本为 `compute_cd2.py` ，新增了对单词的提取与统计功能。具体做法是利用正则表达式匹配 ASCII 字母序列作为单词，并在统计时将所有撇号形式的所有格结尾（如 `China's` 、 `people’s` ）视为整词处理，避免孤立的 `s` 影响频率分布。

III.3.1. 纵观所有结果

```
(nlp) PS E:\homework\nlp\hw1> python compute_cd2.py data\cd_snapshot_2MB.txt
--top 10
Corpus: data\cd_snapshot_2MB.txt
Total words: 321481
Unique words: 17145
Shannon entropy: 10.287671 bits

Top words:
rank word                 count    prob       information(bits) contribution
   1  the                 20490  0.063736        3.971742         0.253144
   2  and                 11410  0.035492        4.816363         0.170942
   3  of                   9785  0.030437        5.038018         0.153343
   4  to                   8979  0.027930        5.162035         0.144176
   5  in                   7703  0.023961        5.383169         0.128986
   6  a                    6249  0.019438        5.684964         0.110505
   7  china                3419  0.010635        6.555015         0.069714
   8  for                  3364  0.010464        6.578412         0.068837
   9  is                   2755  0.008570        6.866537         0.058844
  10  that                 2582  0.008032        6.960101         0.055901
```

```
(nlp) PS E:\homework\nlp\hw1> python compute_cd2.py data\cd_snapshot_5MB.txt
--top 10
Corpus: data\cd_snapshot_5MB.txt
Total words: 803795
Unique words: 26020
Shannon entropy: 10.380079 bits

Top words:
rank word                  count    prob      information(bits)  contribution
   1  the                  51976  0.064663        3.950910         0.255479
   2  and                  28260  0.035158        4.829994         0.169814
   3  of                   25020  0.031127        5.005674         0.155813
   4  to                   22073  0.027461        5.186473         0.142426
   5  in                   19488  0.024245        5.366170         0.130103
   6  a                    15591  0.019397        5.688042         0.110329
   7  for                   8270  0.010289        6.602796         0.067934
   8  china                 8029  0.009989        6.645463         0.066381
   9  is                    6809  0.008471        6.883241         0.058308
  10  that                  6341  0.007889        6.985973         0.055111
```

```
(nlp) PS E:\homework\nlp\hw1> python compute_cd2.py data\cd_snapshot_10MB.txt
--top 10
Corpus: data\cd_snapshot_10MB.txt
Total words: 1606038
Unique words: 34717
Shannon entropy: 10.425898 bits

Top words:
rank word                  count    prob      information(bits)  contribution
   1  the                 101812  0.063393        3.979527         0.252275
   2  and                  55965  0.034847        4.842837         0.168757
   3  of                   48236  0.030034        5.057252         0.151890
   4  to                   44129  0.027477        5.185635         0.142485
   5  in                   37712  0.023481        5.412339         0.127089
   6  a                    31622  0.019689        5.666434         0.111569
   7  for                  16283  0.010139        6.623996         0.067158
   8  china                15567  0.009693        6.688871         0.064834
   9  is                   13617  0.008479        6.881953         0.058350
  10  on                   12975  0.008079        6.951628         0.056161
```

III.3.2. 结果分析

在英文单词统计中，将所有撇号形式的所有格结尾（如 `China's` 、 `people's` ）视为整词处理后，孤立的 `s` 不再计入词频，因此 top-10 高频词更贴近常见功能词分布，整体熵值也随语料规模上升略有增加（约 10.29⟶10.43 bits），表明词级别的多样性与语料扩展保持正相关。

III.3.3. 齐夫定律验证

为了检验齐夫定律，我分别对 2MB、5MB、10MB 英文语料中排名前 200 的词频做了 `log10(rank)` 与 `log10(freq)` 的线性回归，得到的斜率接近理想值 −1，说明词频随排名呈现出近似的幂律衰减。

回归之外，直接检查头部词频也能看到齐夫定律的特征：以 10MB 样本为例，`rank × freq` 对于前十个词大致稳定在 0.07 0.12 区间；不同规模语料中 `the`、`and`、`of` 等功能词的概率几乎保持在 0.063、0.035、0.030 左右，随着语料容量扩大，绝对频次按比例增长而相对频率基本不变。这些现象共同表明，英文单词分布在大规模新闻语料上符合齐夫定律，且统计特性在几兆字节的样本量下已经充分收敛。

```
(nlp) PS E:\homework\nlp\hw1> python verify_zipf.py data\cd_snapshot_2MB.txt
--top 200 --show 10
Corpus: data\cd_snapshot_2MB.txt
Total words: 321481
Unique words: 17145
Top ranks used: 1-200
Linear fit (log10 rank vs log10 freq): slope=-0.934, intercept=-1.137,
mse=0.00066

rank    freq    rank*freq  log10(rank) log10(freq)
   1  0.063736  0.063736    0.000000   -1.195613
   2  0.035492  0.070984    0.301030   -1.449870
   3  0.030437  0.091312    0.477121   -1.516594
   4  0.027930  0.111720    0.602060   -1.553927
   5  0.023961  0.119805    0.698970   -1.620495
   6  0.019438  0.116629    0.778151   -1.711345
   7  0.010635  0.074446    0.845098   -1.973256
   8  0.010464  0.083713    0.903090   -1.980299
   9  0.008570  0.077127    0.954243   -2.067034
  10  0.008032  0.080316    1.000000   -2.095199
(nlp) PS E:\homework\nlp\hw1> python verify_zipf.py data\cd_snapshot_5MB.txt
--top 200 --show 10
Corpus: data\cd_snapshot_5MB.txt
Total words: 803795
Unique words: 26020
Top ranks used: 1-200
Linear fit (log10 rank vs log10 freq): slope=-0.935, intercept=-1.142,
mse=0.00071

rank    freq    rank*freq  log10(rank) log10(freq)
   1  0.064663  0.064663    0.000000   -1.189342
   2  0.035158  0.070316    0.301030   -1.453973
   3  0.031127  0.093382    0.477121   -1.506858
   4  0.027461  0.109844    0.602060   -1.561284
   5  0.024245  0.121225    0.698970   -1.615378
   6  0.019397  0.116380    0.778151   -1.712271
   7  0.010289  0.072021    0.845098   -1.987640
   8  0.009989  0.079911    0.903090   -2.000484
   9  0.008471  0.076240    0.954243   -2.072062
  10  0.007889  0.078888    1.000000   -2.102988
(nlp) PS E:\homework\nlp\hw1> python verify_zipf.py data\cd_snapshot_10MB.txt
--top 200 --show 10
Corpus: data\cd_snapshot_10MB.txt
Total words: 1606038
```

```
Unique words: 34717
Top ranks used: 1-200
Linear fit (log10 rank vs log10 freq): slope=-0.925, intercept=-1.155,
mse=0.00082
```

rank	freq	rank*freq	log10(rank)	log10(freq)
1	0.063393	0.063393	0.000000	-1.197957
2	0.034847	0.069693	0.301030	-1.457839
3	0.030034	0.090102	0.477121	-1.522385
4	0.027477	0.109908	0.602060	-1.561032
5	0.023481	0.117407	0.698970	-1.629276
6	0.019689	0.118137	0.778151	-1.705766
7	0.010139	0.070970	0.845098	-1.994021
8	0.009693	0.077542	0.903090	-2.013551
9	0.008479	0.076308	0.954243	-2.071674
10	0.008079	0.080789	1.000000	-2.092648

IV. Chapter 4 Conclusion

通过本次实验，我构建了从数据采集到统计分析的完整流程：先针对人民日报与 China Daily 制作定向爬虫，结合 `CorpusMonitor` 在关键体积截取快照，再以 `compute_rmrb.py` 与 `compute_cd.py` 以及 `compute_cd2.py` 对语料进行单字符层面和单词层面的概率与熵值计算。整个实践强化了“高质量数据 + 精确统计”这一基本功：爬虫阶段的结构化解析、正文抽取与编码清洗直接决定了后续统计的可信度，而脚本化的熵计算则提供了量化语言冗余度与信息密度的手段。

实验结果显示，中文汉字的熵值稳定在约 9.6 bits，英文字母约 4.16 bits，英文单词约 10.38 bits，两者在 2MB/5MB/10MB 的规模扩展后仍保持高度一致的高频符号分布。这一现象印证了语言统计的“早期收敛”特性：即便语料继续累积，单字符概率的主干结构不会发生突变，新增数据更多是在补充长尾符号并降低采样噪声。另一方面，中文熵值显著高于英文字母，而英文单词熵值则更高，反映出不同语言在符号集大小与使用频率分布上的差异。

本次作业通过对英文单词层面的扩展，进一步验证了齐夫定律在大规模新闻语料中的适用性。无论是线性回归的斜率接近 −1，还是头部词频的 `rank × freq` 稳定性，都表明英文单词分布符合经典的幂律规律。这不仅加深了我对语言统计学基本原理的理解，也为后续更复杂的语言模型构建（如 n 元模型、主题模型）奠定了基础。

本次作业还暴露了一些值得深入的方向。例如，快照机制固然方便观察总体趋势，但若要比较不同板块（如财经、文化）或不同时间段的用字差异，还需在爬虫阶段记录更细粒度的元数据，并在统计脚本中支持分组或增量分析。此外，单字符熵只能捕捉零阶语言特征，要理解句法与语义层面，必须引入二元、三元乃至子词/词级模型，配合停用词处理和语言模型评估，才能揭示更深层的语言组织规律。

综上所述，这次实践不仅完成了作业要求，也为后续的深入研究奠定了基础：一方面掌握了构建垂直爬虫与清洗语料的工程方法，另一方面通过熵值分析认识到语言统计的稳定性与差异性。未来可以在此框架上继续扩展，如引入自动化监控、可视化仪表板或与云端数据管道结合，以支持更大规模、更细粒度的文本分析任务。

V. Appendix: Code Listings

代码仓库地址：https://github.com/CarterYin/NLP_UCAS_2025

为完整性起见，以下列出所有代码文件内容。

V.1. rm.py

```
'''
代码名称：爬取人民日报数据为txt文件
编写日期：2025年1月1日
作者：github (caspiankexin) (CarterYin)
版本：第3版
可爬取的时间范围：2024年12月起
注意：此代码仅供交流学习，不得作为其他用途。
'''
```

```python
import requests
import bs4
import os
import datetime
import time
from typing import Optional, List

MB = 1024 * 1024
THRESHOLDS = [2 * MB, 5 * MB, 10 * MB]
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(SCRIPT_DIR, 'data')


class CorpusMonitor:
    """监控累计抓取体积并按阈值写入快照。"""

    def __init__(self, thresholds: Optional[List[int]] = None) -> None:
        self._thresholds = list(thresholds or THRESHOLDS)
        self._buffer = bytearray()
        self._total_bytes = 0

    def update(self, payload: bytes) -> None:
        self._buffer.extend(payload)
        self._total_bytes += len(payload)
        print(f"当前累计大小: {self._total_bytes / MB:.2f} MB")
        while self._thresholds and self._total_bytes >= self._thresholds[0]:
            threshold = self._thresholds.pop(0)
            self._save_snapshot(threshold)

    def _save_snapshot(self, threshold: int) -> None:
        os.makedirs(DATA_DIR, exist_ok=True)
        size_mb = threshold // MB
        filename = f"rmrb_snapshot_{size_mb}MB.txt"
        path = os.path.join(DATA_DIR, filename)
```

```
        with open(path, 'wb') as f:
            f.write(self._buffer[:threshold])
        print(f"快照已保存: {filename}")

def fetchUrl(url):
    '''
    功能：访问 url 的网页，获取网页内容并返回
    参数：目标网页的 url
    返回：目标网页的 html 内容
    '''

    headers = {
        'accept': 'text/html,application/xhtml+xml,application/
xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
        'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36',
    }

    r = requests.get(url,headers=headers)
    r.raise_for_status()
    r.encoding = r.apparent_encoding
    return r.text

def getPageList(year, month, day):
    '''
    功能：获取当天报纸的各版面的链接列表
    参数：年，月，日
    '''
    url = 'http://paper.people.com.cn/rmrb/pc/layout/' + year + month + '/' +
day + '/node_01.html'
    html = fetchUrl(url)
    bsobj = bs4.BeautifulSoup(html,'html.parser')
    temp = bsobj.find('div', attrs = {'id': 'pageList'})
    if temp:
        pageList = temp.ul.find_all('div', attrs = {'class': 'right_title-
name'})
    else:
        pageList = bsobj.find('div', attrs = {'class': 'swiper-
container'}).find_all('div', attrs = {'class': 'swiper-slide'})
    linkList = []

    for page in pageList:
        link = page.a["href"]
        url = 'http://paper.people.com.cn/rmrb/pc/layout/'  + year + month +
'/' + day + '/' + link
        linkList.append(url)

    return linkList

def getTitleList(year, month, day, pageUrl):
    '''
```

```
    功能：获取报纸某一版面的文章链接列表
    参数：年，月，日，该版面的链接
    '''
    html = fetchUrl(pageUrl)
    bsobj = bs4.BeautifulSoup(html,'html.parser')
    temp = bsobj.find('div', attrs = {'id': 'titleList'})
    if temp:
        titleList = temp.ul.find_all('li')
    else:
        titleList = bsobj.find('ul', attrs = {'class': 'news-
list'}).find_all('li')
    linkList = []

    for title in titleList:
        tempList = title.find_all('a')
        for temp in tempList:
            link = temp["href"]
            if 'content' in link:
                url = 'http://paper.people.com.cn/rmrb/pc/content/' + year +
month + '/' + day + '/' + link
                linkList.append(url)

    return linkList

def getContent(html):
    '''
    功能：解析 HTML 网页，获取新闻的文章内容
    参数：html 网页内容
    '''
    bsobj = bs4.BeautifulSoup(html,'html.parser')

    # 获取文章 标题
    title = bsobj.h3.text + '\n' + bsobj.h1.text + '\n' + bsobj.h2.text + '\n'
    #print(title)

    # 获取文章 内容
    pList = bsobj.find('div', attrs = {'id': 'ozoom'}).find_all('p')
    content = ''
    for p in pList:
        content += p.text + '\n'
    #print(content)

    # 返回结果 标题+内容
    resp = title + content
    return resp

def append_to_corpus(article_text, corpus_path, monitor:
Optional[CorpusMonitor] = None):
    '''
    功能：将文章内容追加到单一语料文件
    参数：文章内容字符串，语料文件路径
```

```
    '''
    corpus_dir = os.path.dirname(corpus_path)
    if corpus_dir and not os.path.exists(corpus_dir):
        os.makedirs(corpus_dir)

    with open(corpus_path, 'a', encoding='utf-8') as f:
        f.write(article_text)

    if monitor is not None:
        monitor.update(article_text.encode('utf-8'))

def download_rmrb(year, month, day, corpus_path, monitor:
Optional[CorpusMonitor] = None):
    '''
    功能：爬取《人民日报》网站 某年 某月 某日 的新闻内容，并追加保存到指定语料文件
    参数：年，月，日，语料文件路径
    '''
    pageList = getPageList(year, month, day)
    pageNo = 0
    for page in pageList:
        try:
            pageNo = pageNo + 1
            titleList = getTitleList(year, month, day, page)
            titleNo = 0
            for url in titleList:
                titleNo = titleNo + 1

                # 获取新闻文章内容
                html = fetchUrl(url)
                content = getContent(html)

#                 article_header = (
#                     f"### 日期：{year}-{month}-{day} 版面：
{str(pageNo).zfill(2)} 文章：{str(titleNo).zfill(2)}\n"
#                 )
#                 article_meta = f"来源链接：{url}\n"
                article_body = content.strip() + "\n\n"
                append_to_corpus(article_body, corpus_path, monitor)
        except Exception as e:
            print(f"日期 {year}-{month}-{day} 下的版面 {page} 出现错误：{e}")
            continue


def gen_dates(b_date, days):
    day = datetime.timedelta(days = 1)
    for i in range(days):
        yield b_date + day * i


def get_date_list(beginDate, endDate):
    """
```

```
    获取日期列表
    :param start: 开始日期
    :param end: 结束日期
    :return: 开始日期和结束日期之间的日期列表
    """

    start = datetime.datetime.strptime(beginDate, "%Y%m%d")
    end = datetime.datetime.strptime(endDate, "%Y%m%d")

    data = []
    for d in gen_dates(start, (end-start).days):
        data.append(d)

    return data


if __name__ == '__main__':
    '''
    主函数：程序入口
    '''
    # 输入起止日期，爬取之间的新闻
    print("欢迎使用人民日报爬虫，请输入以下信息：")
    beginDate = input('请输入开始日期:')
    endDate = input('请输入结束日期:')
    corpus_path = input("请输入语料文件的保存路径（例如 D:/data/
rmrb_corpus.txt）：").strip()
    if not corpus_path:
        raise ValueError("语料文件路径不能为空")
    corpus_path = os.path.abspath(corpus_path)
    corpus_dir = os.path.dirname(corpus_path) or '.'
    if not os.path.exists(corpus_dir):
        os.makedirs(corpus_dir)
    with open(corpus_path, 'w', encoding='utf-8') as f:
        f.write('')
    monitor = CorpusMonitor()
    data = get_date_list(beginDate, endDate)

    for d in data:
        year = str(d.year)
        month = str(d.month) if d.month >=10 else '0' + str(d.month)
        day = str(d.day) if d.day >=10 else '0' + str(d.day)
        download_rmrb(year, month, day, corpus_path, monitor)
        print("爬取完成：" + year + month + day)
        time.sleep(5)          # 怕被封 IP 爬一爬缓一缓，爬的少的话可以注释掉

    lastend = input("本月数据爬取完成!可以关闭软件了")
```

V.2. cd.py

```
'''
代码名称：爬取 China Daily 英文版数据为 txt 文件
```

```
编写日期：2025年1月1日
作者：github (CarterYin)
版本：第1版
可爬取的时间范围：2021年起（以站点可用为准）
注意：此代码仅供交流学习，不得作为其他用途。
'''

import datetime
import os
import re
import time
import urllib.parse
from collections import OrderedDict
from typing import List, Optional

import bs4
import requests

MB = 1024 * 1024
THRESHOLDS = [2 * MB, 5 * MB, 10 * MB]
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(SCRIPT_DIR, 'data')
BASE_DOMAIN = 'https://epaper.chinadaily.com.cn'
DEFAULT_EDITION = os.environ.get('CHINADAILY_EDITION', 'global').strip() or
'global'


class CorpusMonitor:
    """监控累计抓取体积并按阈值写入快照。"""

    def __init__(self, thresholds: Optional[List[int]] = None) -> None:
        self._thresholds = list(thresholds or THRESHOLDS)
        self._buffer = bytearray()
        self._total_bytes = 0

    def update(self, payload: bytes) -> None:
        self._buffer.extend(payload)
        self._total_bytes += len(payload)
        print(f"当前累计大小: {self._total_bytes / MB:.2f} MB")
        while self._thresholds and self._total_bytes >= self._thresholds[0]:
            threshold = self._thresholds.pop(0)
            self._save_snapshot(threshold)

    def _save_snapshot(self, threshold: int) -> None:
        os.makedirs(DATA_DIR, exist_ok=True)
        size_mb = threshold // MB
        filename = f"cd_snapshot_{size_mb}MB.txt"
        path = os.path.join(DATA_DIR, filename)
        with open(path, 'wb') as f:
            f.write(self._buffer[:threshold])
        print(f"快照已保存: {filename}")
```

```
def fetch_url(url: str) -> str:
    """访问指定 URL 并返回 HTML 文本。"""
    headers = {
        'accept': 'text/html,application/xhtml+xml,application/
xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
        'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36',
    }
    resp = requests.get(url, headers=headers, timeout=15)
    resp.raise_for_status()
    resp.encoding = resp.apparent_encoding or resp.encoding
    return resp.text


def build_index_url(year: str, month: str, day: str, edition: str) -> str:
    """构造某一天 China Daily 电子报索引页 URL。"""
    return f"{BASE_DOMAIN}/{edition}/{year}-{month}/{day}/index.html"


def extract_issue_links(index_html: str, year: str, month: str, day: str) ->
List[str]:
    """从索引页提取当日所有版块的文章链接。"""
    soup = bs4.BeautifulSoup(index_html, 'html.parser')
    issue_path = f"/a/{year}{month}/{day}/"
    links = OrderedDict()

    for anchor in soup.find_all('a', href=True):
        href = anchor['href'].strip()
        if not href:
            continue
        if href.startswith('//'):
            href = 'https:' + href
        elif href.startswith('/'):
            href = urllib.parse.urljoin(BASE_DOMAIN, href)
        elif not href.startswith('http'):
            href = urllib.parse.urljoin(BASE_DOMAIN, href)
        if issue_path not in href:
            continue
        if not href.endswith('.html'):
            continue
        links[href] = None

    return list(links.keys())


def _first_non_empty_text(soup: bs4.BeautifulSoup, selectors: List[str]) ->
Optional[str]:
    """按顺序返回第一个匹配选择器并含非空文本的节点内容。"""
    for selector in selectors:
```

```
        node = soup.select_one(selector)
        if node:
            text = node.get_text(' ', strip=True)
            if text:
                return text
    return None


def extract_article_content(html: str) -> str:
    """解析正文页面并返回标题及正文拼接文本。"""
    soup = bs4.BeautifulSoup(html, 'html.parser')

    article_root = None
    for selector in ['article', '#Content', '.content', '.text', '.lft',
'.main_l', '.left', '.news', '.news_l']:
        article_root = soup.select_one(selector)
        if article_root:
            break
    if article_root is None:
        article_root = soup.body or soup

    headline = _first_non_empty_text(article_root, [
        'h1.headline',
        'h1.title',
        'div.lft h1',
        '#Content h1',
        'article h1',
        'h1',
    ])
    sub_head = _first_non_empty_text(article_root, [
        'h2.subhead',
        'h2.subtitle',
        'div.lft h2',
        '#Content h2',
        'article h2',
        'h2',
    ])

    meta_text = None
    meta_candidates = article_root.select('div.info, div.meta, p.meta, p.info,
div.editor, div.source, p.source')
    for candidate in meta_candidates:
        candidate_text = candidate.get_text(' ', strip=True)
        if not candidate_text:
            continue
        if 'Updated' in candidate_text or candidate_text.startswith('By '):
            meta_text = candidate_text
            candidate.extract()
            break

    paragraphs: List[str] = []
    for block in article_root.find_all(['p', 'div']):
```

```
        if block.name == 'div' and block.find(['p', 'div']):
            continue
        text = block.get_text(' ', strip=True)
        if not text:
            continue
        if re.search(r'Copyright\s+1995', text):
            continue
        if 'Additional Links' in text or 'Search' == text:
            continue
        if text.upper().startswith('HOME') and 'Newspaper' in text:
            continue
        paragraphs.append(text)

    if not paragraphs:
        fallback_paragraphs = []
        for tag in soup.find_all('p'):
            text = tag.get_text(' ', strip=True)
            if len(text.split()) < 3:
                continue
            if re.search(r'Copyright\s+1995', text):
                continue
            fallback_paragraphs.append(text)
        paragraphs = fallback_paragraphs

    unique_paragraphs = list(OrderedDict((para, None) for para in
paragraphs).keys())

    header_parts = [part for part in [headline, sub_head, meta_text] if part]
    body_text = '\n'.join(unique_paragraphs)
    combined = '\n'.join(header_parts + ([body_text] if body_text else []))
    return combined.strip()


def append_to_corpus(article_text: str, corpus_path: str, monitor:
Optional[CorpusMonitor] = None) -> None:
    """将文章内容追加到单一语料文件。"""
    corpus_dir = os.path.dirname(corpus_path)
    if corpus_dir and not os.path.exists(corpus_dir):
        os.makedirs(corpus_dir)

    with open(corpus_path, 'a', encoding='utf-8') as f:
        f.write(article_text)

    if monitor is not None:
        monitor.update(article_text.encode('utf-8'))


def download_chinadaily(year: str, month: str, day: str, corpus_path: str,
monitor: Optional[CorpusMonitor] = None, edition: str = DEFAULT_EDITION) ->
None:
    """爬取 China Daily 网站指定日期的新闻并写入语料。"""
    index_url = build_index_url(year, month, day, edition)
```

```
    try:
        index_html = fetch_url(index_url)
    except Exception as exc:  # pragma: no cover - 网络异常提醒
        print(f"无法获取 {year}-{month}-{day} 的索引页 {index_url}: {exc}")
        return

    article_links = extract_issue_links(index_html, year, month, day)
    if not article_links:
        print(f"未在 {index_url} 找到文章链接。")
        return

    for seq, url in enumerate(article_links, 1):
        try:
            article_html = fetch_url(url)
            content = extract_article_content(article_html)
            if not content:
                print(f"跳过空白文章: {url}")
                continue
            article_body = content.strip() + "\n\n"
            append_to_corpus(article_body, corpus_path, monitor)
            print(f"成功抓取第 {seq:02d} 篇: {url}")
            time.sleep(1)
        except Exception as exc:  # pragma: no cover - 逐篇抓取问题提示
            print(f"文章 {url} 抓取失败: {exc}")
            continue


def gen_dates(b_date: datetime.datetime, days: int):
    day = datetime.timedelta(days=1)
    for i in range(days):
        yield b_date + day * i


def get_date_list(begin_date: str, end_date: str):
    """获取日期列表。"""
    start = datetime.datetime.strptime(begin_date, "%Y%m%d")
    end = datetime.datetime.strptime(end_date, "%Y%m%d")

    data = []
    for d in gen_dates(start, (end - start).days):
        data.append(d)

    return data


if __name__ == '__main__':
    print("欢迎使用 China Daily 英文版爬虫，请输入以下信息：")
    begin_date = input('请输入开始日期:')
    end_date = input('请输入结束日期:')
    corpus_path = input('请输入语料文件的保存路径（例如 D:/data/
chinadaily_corpus.txt): ').strip()
```

```
    if not corpus_path:
        raise ValueError('语料文件路径不能为空')
    corpus_path = os.path.abspath(corpus_path)
    corpus_dir = os.path.dirname(corpus_path) or '.'
    if not os.path.exists(corpus_dir):
        os.makedirs(corpus_dir)
    with open(corpus_path, 'w', encoding='utf-8') as f:
        f.write('')
    monitor = CorpusMonitor()
    date_list = get_date_list(begin_date, end_date)

    for current in date_list:
        year = str(current.year)
        month = str(current.month) if current.month >= 10 else '0' +
str(current.month)
        day = str(current.day) if current.day >= 10 else '0' +
str(current.day)
        download_chinadaily(year, month, day, corpus_path, monitor,
DEFAULT_EDITION)
        print("爬取完成: " + year + month + day)
        time.sleep(5)

    input('本段数据抓取完成!可以关闭软件了')
```

V.3. compute_rmrb.py

```
"""Compute single-character probabilities and entropy for Chinese text
corpora."""

from __future__ import annotations

import argparse
import math
from collections import Counter
from pathlib import Path
from typing import Iterable

CJK_RANGE_START = 0x4E00
CJK_RANGE_END = 0x9FFF


def iter_cjk_chars(text: str) -> Iterable[str]:
  for char in text:
    code_point = ord(char)
    if CJK_RANGE_START <= code_point <= CJK_RANGE_END:
      yield char


def analyze_characters(text: str) -> tuple[Counter[str], int, float]:
  counts = Counter(iter_cjk_chars(text))
  total = sum(counts.values())
  if not total:
```

```
    raise ValueError("input corpus does not contain any CJK Unified
Ideographs")
  probabilities = (count / total for count in counts.values())
  entropy = -sum(p * math.log2(p) for p in probabilities)
  return counts, total, entropy


def format_top_characters(counts: Counter[str], total: int, top_k: int) ->
str:
  lines = []
  header = "rank char count prob information(bits) contribution"
  lines.append(header)
  for rank, (char, count) in enumerate(counts.most_common(top_k), 1):
    probability = count / total
    information_bits = -math.log2(probability)
    contribution = probability * information_bits
    lines.append(
      f"{rank:>4}  {char}   {count:>8}  {probability:>0.6f}"
      f"      {information_bits:>10.6f}       {contribution:>10.6f}"
    )
  return "\n".join(lines)


def main() -> None:
  parser = argparse.ArgumentParser(
    description="Compute Chinese character probabilities and Shannon entropy."
  )
  parser.add_argument(
    "corpus",
    type=Path,
    help="Path to the UTF-8 encoded corpus file to analyze.",
  )
  parser.add_argument(
    "--top",
    type=int,
    default=20,
    help="Show the top-K most frequent characters (default: 20).",
  )
  args = parser.parse_args()

  text = args.corpus.read_text(encoding="utf-8", errors="ignore")
  counts, total, entropy = analyze_characters(text)
  unique = len(counts)

  print(f"Corpus: {args.corpus}")
  print(f"Total Chinese characters: {total}")
  print(f"Unique Chinese characters: {unique}")
  print(f"Shannon entropy: {entropy:.6f} bits")

  if args.top > 0:
    print("\nTop characters:")
    print(format_top_characters(counts, total, args.top))
```

```
if __name__ == "__main__":
  main()
```

V.4. compute_cd.py

```
"""Compute single-character probabilities and entropy for English text
corpora."""

from __future__ import annotations

import argparse
import math
from collections import Counter
from pathlib import Path
from typing import Iterable


def iter_english_letters(text: str, case_sensitive: bool) -> Iterable[str]:
  """Yield ASCII alphabetic characters, normalizing case when requested."""
  for char in text:
    if char.isascii() and char.isalpha():
      yield char if case_sensitive else char.lower()


def analyze_characters(text: str, case_sensitive: bool) -> tuple[Counter[str],
int, float]:
  counts = Counter(iter_english_letters(text, case_sensitive))
  total = sum(counts.values())
  if not total:
    raise ValueError("input corpus does not contain any ASCII alphabetic
characters")
  probabilities = (count / total for count in counts.values())
  entropy = -sum(p * math.log2(p) for p in probabilities)
  return counts, total, entropy


def format_top_characters(counts: Counter[str], total: int, top_k: int) ->
str:
  lines = []
  header = "rank char count prob information(bits) contribution"
  lines.append(header)
  for rank, (char, count) in enumerate(counts.most_common(top_k), 1):
    probability = count / total
    information_bits = -math.log2(probability)
    contribution = probability * information_bits
    lines.append(
      f"{rank:>4}  {char}   {count:>8}  {probability:>0.6f}"
      f"      {information_bits:>10.6f}       {contribution:>10.6f}"
    )
  return "\n".join(lines)
```

```
def main() -> None:
  parser = argparse.ArgumentParser(
    description="Compute English letter probabilities and Shannon entropy."
  )
  parser.add_argument(
    "corpus",
    type=Path,
    help="Path to the UTF-8 encoded corpus file to analyze.",
  )
  parser.add_argument(
    "--top",
    type=int,
    default=20,
    help="Show the top-K most frequent letters (default: 20).",
  )
  parser.add_argument(
    "--case-sensitive",
    action="store_true",
    help="Treat uppercase and lowercase letters as distinct symbols.",
  )
  args = parser.parse_args()

  text = args.corpus.read_text(encoding="utf-8", errors="ignore")
  counts, total, entropy = analyze_characters(text, args.case_sensitive)
  unique = len(counts)

  print(f"Corpus: {args.corpus}")
  print(f"Total letters: {total}")
  print(f"Unique letters: {unique}")
  print(f"Shannon entropy: {entropy:.6f} bits")

  if args.top > 0:
    print("\nTop letters:")
    print(format_top_characters(counts, total, args.top))


if __name__ == "__main__":
  main()
```

V.5. compute_cd2.py

```
"""Compute English word probabilities and entropy for text corpora."""

from __future__ import annotations

import argparse
import math
import re
from collections import Counter
from pathlib import Path
from typing import Iterable
```

```
WORD_PATTERN = re.compile(r"[A-Za-z]+")


def iter_english_words(text: str, case_sensitive: bool) -> Iterable[str]:
  """Yield ASCII alphabetic word tokens, normalizing case unless requested."""
  for match in WORD_PATTERN.finditer(text):
    word = match.group(0)
    start = match.start()
    if start > 0 and text[start - 1] in {"'", "’"} and word.lower() == "s":
      continue
    yield word if case_sensitive else word.lower()


def analyze_words(text: str, case_sensitive: bool) -> tuple[Counter[str], int,
float]:
  counts = Counter(iter_english_words(text, case_sensitive))
  total = sum(counts.values())
  if not total:
    raise ValueError("input corpus does not contain any ASCII alphabetic
words")
  probabilities = (count / total for count in counts.values())
  entropy = -sum(p * math.log2(p) for p in probabilities)
  return counts, total, entropy


def format_top_words(counts: Counter[str], total: int, top_k: int) -> str:
  lines = []
  header = "rank word                count prob information(bits)
contribution"
  lines.append(header)
  for rank, (word, count) in enumerate(counts.most_common(top_k), 1):
    probability = count / total
    information_bits = -math.log2(probability)
    contribution = probability * information_bits
    lines.append(
      f"{rank:>4}  {word:<18} {count:>8}  {probability:>0.6f}"
      f"      {information_bits:>10.6f}       {contribution:>10.6f}"
    )
  return "\n".join(lines)


def main() -> None:
  parser = argparse.ArgumentParser(
    description="Compute English word probabilities and Shannon entropy."
  )
  parser.add_argument(
    "corpus",
    type=Path,
    help="Path to the UTF-8 encoded corpus file to analyze.",
  )
  parser.add_argument(
    "--top",
```

```
    type=int,
    default=20,
    help="Show the top-K most frequent words (default: 20).",
  )
  parser.add_argument(
    "--case-sensitive",
    action="store_true",
    help="Treat uppercase and lowercase words as distinct tokens.",
  )
  args = parser.parse_args()

  text = args.corpus.read_text(encoding="utf-8", errors="ignore")
  counts, total, entropy = analyze_words(text, args.case_sensitive)
  unique = len(counts)

  print(f"Corpus: {args.corpus}")
  print(f"Total words: {total}")
  print(f"Unique words: {unique}")
  print(f"Shannon entropy: {entropy:.6f} bits")

  if args.top > 0:
    print("\nTop words:")
    print(format_top_words(counts, total, args.top))


if __name__ == "__main__":
  main()
```

V.6. verify_zipf.py

```
"""Command-line tool to verify Zipf's law on English word frequencies."""

from __future__ import annotations

import argparse
import math
from pathlib import Path
from typing import Iterable

from compute_cd2 import analyze_words


def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="Fit a Zipf regression on English word frequencies.",
    )
    parser.add_argument(
        "corpus",
        type=Path,
        help="Path to the UTF-8 encoded corpus file to analyze.",
    )
    parser.add_argument(
        "--top",
```

```
        type=int,
        default=200,
        help="Number of top-ranked words to include in the regression
(default: 200).",
    )
    parser.add_argument(
        "--case-sensitive",
        action="store_true",
        help="Treat uppercase and lowercase words as distinct tokens.",
    )
    parser.add_argument(
        "--show",
        type=int,
        default=10,
        help="Show the first K rows of the rank-frequency summary (default:
10).",
    )
    return parser.parse_args()


def load_frequencies(path: Path, case_sensitive: bool, top_n: int) ->
tuple[list[tuple[int, float]], int, int]:
    text = path.read_text(encoding="utf-8", errors="ignore")
    counts, total, _ = analyze_words(text, case_sensitive)
    pairs = [
        (rank + 1, count / total)
        for rank, (_, count) in enumerate(counts.most_common(top_n))
    ]
    return pairs, total, len(counts)


def linear_regression(pairs: Iterable[tuple[int, float]]) -> tuple[float,
float, float]:
    log_ranks = [math.log10(rank) for rank, _ in pairs]
    log_freqs = [math.log10(freq) for _, freq in pairs]
    n = len(log_ranks)
    mean_x = sum(log_ranks) / n
    mean_y = sum(log_freqs) / n
    numerator = sum((x - mean_x) * (y - mean_y) for x, y in zip(log_ranks,
log_freqs))
    denominator = sum((x - mean_x) ** 2 for x in log_ranks)
    slope = numerator / denominator
    intercept = mean_y - slope * mean_x
    mse = sum(
        (y - (slope * x + intercept)) ** 2
        for x, y in zip(log_ranks, log_freqs)
    ) / n
    return slope, intercept, mse


def main() -> None:
    args = parse_args()
```

```
    pairs, total, unique = load_frequencies(args.corpus, args.case_sensitive,
args.top)
    if not pairs:
        raise ValueError("no words were found in the corpus")

    slope, intercept, mse = linear_regression(pairs)

    print(f"Corpus: {args.corpus}")
    print(f"Total words: {total}")
    print(f"Unique words: {unique}")
    print(f"Top ranks used: 1-{len(pairs)}")
    print(
        "Linear fit (log10 rank vs log10 freq): "
        f"slope={slope:.3f}, intercept={intercept:.3f}, mse={mse:.5f}"
    )
    print()
    print("rank freq rank*freq log10(rank) log10(freq)")
    sample_pairs = pairs[: max(args.show, 0)]
    for rank, freq in sample_pairs:
        log_rank = math.log10(rank)
        log_freq = math.log10(freq)
        print(
            f"{rank:>4}  {freq:>0.6f}  {rank * freq:>0.6f}  "
            f"{log_rank:>10.6f}  {log_freq:>10.6f}"
        )


if __name__ == "__main__":
    main()
```