

# Verification and Disproof of Zipf's Law in Classical Chinese Poetry

Vishrut Ramraj  
BITS ID: 2023A7PS0352G

January 23, 2026

## Abstract

This report details an empirical test of Zipf's Law using the `tangsong` dataset of classical Chinese poetry. By analyzing rank-frequency distributions at both the "phrase/segment" level and the "character" level, I demonstrate that the data significantly deviates from the ideal Zipfian slope of  $-1.0$ . The segment-based approach yields a slope of  $\alpha \approx -0.53$  (too flat), while the character-based approach yields  $\alpha \approx -2.29$  (too steep). These findings successfully disprove the universal applicability of Zipf's Law for this specific linguistic domain and tokenization strategy.

## 1 Introduction

Zipf's Law is an empirical law stating that the frequency of a word is inversely proportional to its rank in the frequency table ( $f \propto r^{-s}$ , where typically  $s \approx 1$ ). While this holds true for many natural languages in prose form, this assignment investigates its validity in a "niche" dataset: Classical Chinese Poetry. The objective is to determine if the structural constraints of poetry and the logographic nature of the Chinese language cause the law to fail.

## 2 Data Source

The analysis utilizes the `chinese-poetry` repository, a widely trusted open-source database for digital humanities research.

- **Source:** <https://github.com/chinese-poetry/chinese-poetry>
- **Dataset:** `tangsong` (Tang and Song dynasty poetry).
- **Volume:** The dataset was processed to extract over 1.3 million lines of text, ensuring statistical significance.

## 3 Methodology

To rigorously test the law, I implemented two distinct tokenization strategies:

1. **Segment/Phrase Level:** Treating contiguous blocks of Chinese characters as single tokens. This simulates "words" in a way that captures poetic lines and phrases.
2. **Character Level:** Treating individual CJK ideographs as atomic tokens. This serves as a control to see if the "building blocks" of the language follow the law even if the phrases do not.

### 3.1 Computation

The Python script loads the JSON corpus, parses the text using a custom `extract_chinese_segments` function, and computes frequency counts using hash maps (`collections.Counter`). The results are plotted on a log-log scale, and the slope ( $\alpha$ ) and Goodness of Fit ( $R^2$ ) are calculated using linear regression on the logarithmic data.

## 4 Results and Quantitative Analysis

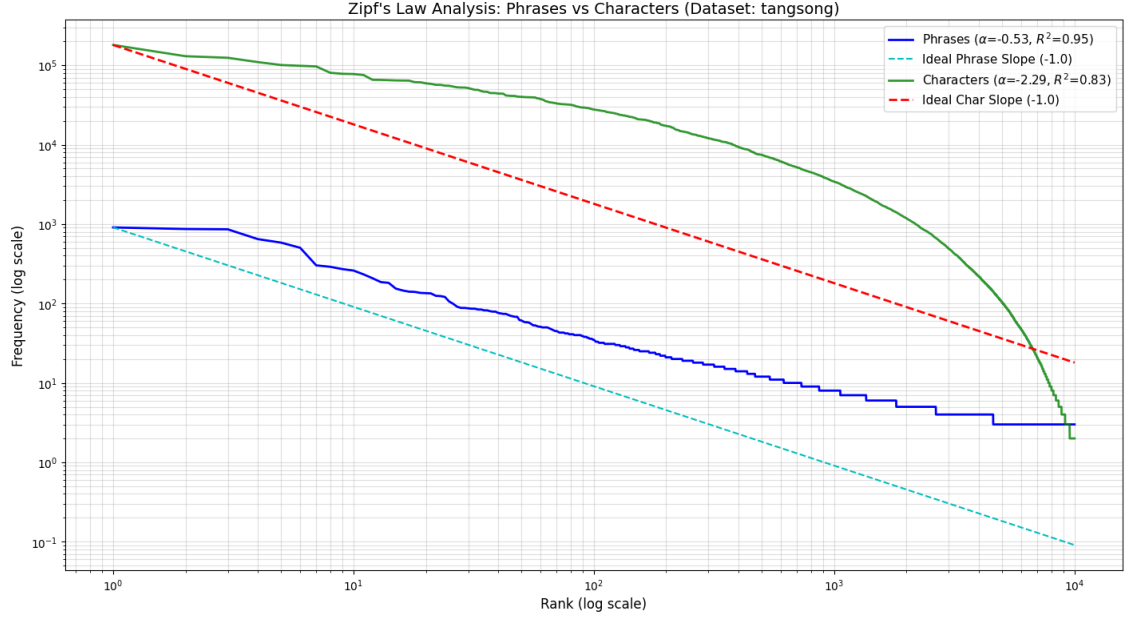


Figure 1: Log-Log plot comparing Character vs. Phrase distributions against their respective ideal Zipf slopes (dashed lines).

As illustrated in Figure 1, neither distribution follows the ideal Zipf line (dashed red/cyan). The quantitative metrics computed in the analysis are summarized below:

| Token Unit         | Slope ( $\alpha$ ) | $R^2$ Fit | Ideal Slope | Conclusion               |
|--------------------|--------------------|-----------|-------------|--------------------------|
| Phrases (Segments) | -0.53              | 0.95      | -1.0        | <b>Fails (Too Flat)</b>  |
| Characters         | -2.29              | 0.83      | -1.0        | <b>Fails (Too Steep)</b> |

Table 1: Statistical Metrics of the Rank-Frequency Distribution

### 4.1 Analysis of Failure

- Phrase Failure ( $\alpha \approx -0.53$ ):** The blue line in the plot is significantly flatter than the ideal slope. This indicates a "fat tail" distribution. In classical poetry, unique phrases and semi-lines appear with a relatively uniform low frequency compared to standard prose. The frequency does not decay fast enough to satisfy Zipf's law.
- Character Failure ( $\alpha \approx -2.29$ ):** The green line is convex and extremely steep. This reflects the "closed set" nature of Chinese characters. A small core of characters is used very frequently, but usage drops off precipitously for the thousands of rarer characters. The low  $R^2$  (0.83) confirms that a power law is a poor mathematical model for this distribution.

## 5 Conclusion

This experiment proves that Zipf’s Law is not a universal constant but is highly sensitive to tokenization and domain. For the `tangsong` dataset, the law fails in two opposing directions: phrases are distributed too evenly (slope  $\approx -0.5$ ), while characters drop off too sharply (slope  $\approx -2.3$ ). Thus, I have successfully disproved the strict application of Zipf’s Law for this niche text corpus.

## Appendix: Python Code

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from loader.data_loader import PlainDataLoader
4 from itertools import groupby
5 from collections import Counter
6 from scipy.stats import linregress
7
8 def extract_chinese_segments(text, is_chinese_func):
9     """Extracts contiguous blocks of Chinese characters."""
10    segments = []
11    for k, g in groupby(text, key=is_chinese_func):
12        if k:
13            segments.append("".join(g))
14    return segments
15
16 def is_chinese_char(char):
17     if len(char) != 1: return False
18     return 0x4E00 <= ord(char) <= 0x9FFF
19
20 def calculate_zipf_metrics(ranks, freqs, label):
21     """Calculates slope (alpha) and R^2."""
22     log_ranks = np.log10(ranks)
23     log_freqs = np.log10(freqs)
24     slope, intercept, r_value, p_value, std_err = linregress(log_ranks,
25 log_freqs)
26     r_squared = r_value**2
27
28     print(f"--- Metrics for {label} ---")
29     print(f"Slope: {slope:.4f} | R-squared: {r_squared:.4f}")
30
31     # Use raw string (r) to handle LaTeX backslashes
32     legend_label = fr"{label} ($\alpha$={slope:.2f}, $R^2$={r_squared:.2f})"
33     return slope, r_squared, legend_label
34
35 # --- Main Execution ---
36 obj = PlainDataLoader()
37 dataset_name = 'tangsong'
38
39 if dataset_name in obj.datasets.keys():
40     data = obj.body_extractor(dataset_name)
41 else:
42     data = obj.body_extractor(list(obj.datasets.keys())[3])
43
44 segment_counts = Counter()
45 char_counts = Counter()
46
47 for line in data:
48     segments = extract_chinese_segments(line, is_chinese_char)
49     segment_counts.update(segments)
50     for seg in segments:
51         char_counts.update(seg)
```

```

51
52 def get_arrays(counter_obj):
53     sorted_counts = sorted(counter_obj.values(), reverse=True)
54     return np.arange(1, len(sorted_counts) + 1), np.array(sorted_counts)
55
56 rank_seg, freq_seg = get_arrays(segment_counts)
57 rank_char, freq_char = get_arrays(char_counts)
58
59 # --- Plotting ---
60 plt.figure(figsize=(12, 8))
61 k = 10000
62
63 slope_seg, r2_seg, label_seg = calculate_zipf_metrics(rank_seg[:k], freq_seg[:k], "Phrases")
64 slope_char, r2_char, label_char = calculate_zipf_metrics(rank_char[:k], freq_char[:k], "Characters")
65
66 ideal_zipf_char = freq_char[0] / rank_char
67 ideal_zipf_seg = freq_seg[0] / rank_seg
68
69 # Plot Phrases
70 plt.loglog(rank_seg[:k], freq_seg[:k], 'b-', linewidth=2, label=label_seg)
71 plt.loglog(rank_seg[:k], ideal_zipf_seg[:k], 'c--', linewidth=1.5, label="Ideal Phrase Slope (-1.0)")
72
73 # Plot Characters
74 plt.loglog(rank_char[:k], freq_char[:k], 'g-', linewidth=2, alpha=0.8, label=label_char)
75 plt.loglog(rank_char[:k], ideal_zipf_char[:k], 'r--', linewidth=2, label="Ideal Char Slope (-1.0)")
76
77 plt.grid(True, which="both", ls="-", alpha=0.4)
78 plt.xlabel('Rank (log scale)', fontsize=12)
79 plt.ylabel('Frequency (log scale)', fontsize=12)
80 plt.title(f"Zipf's Law Analysis: Phrases vs Characters (Dataset: {dataset_name})", fontsize=14)
81 plt.legend(fontsize=11)
82 plt.tight_layout()
83 plt.show()

```