Report of Deep Learning for Natural Langauge Processing

Yiwen Cui
wutcyw@163.com

Abstract

This article is based on the corpus of 16 novels by Jin Yong, verifying Zipf's Law, and utilizing unigram, bigram, and trigram models to calculate the average information entropy of Chinese characters and words respectively.

Introduction

The complexity of natural language has long been a significant focus of scholarly inquiry. In linguistics, George Zipf proposed an important empirical law known as Zipf's Law. This law explores the regularity between word frequency and word rank in natural language. In essence, Zipf's Law reveals a specific relationship between the frequency of words and their positions in a ranked sequence within any corpus: the higher the rank of a word, the higher its frequency, and the lower its rank, the lower its frequency, with an approximate power-law relationship between the two. This law is not only significant in the field of linguistics but also widely applied in information retrieval, text mining, and other areas. By studying Zipf's Law, we can gain deeper insights into the organizational structure and characteristics of natural language, thereby providing robust support for the development of text analysis and natural language processing technologies.

Information entropy is a fundamental concept in information theory, first introduced by Claude Shannon in the mid-20th century. It quantifies the uncertainty or randomness of a system or message. In essence, information entropy measures the average amount of information contained in each message symbol or event within a given information source. The higher the entropy, the greater the uncertainty or randomness, and vice versa.In the context of natural language processing, information entropy plays a crucial role in analyzing and understanding textual data. By calculating the entropy of a text corpus, researchers can gain insights into its complexity, diversity, and predictability. Specifically, in the realm of Chinese language, information entropy has been extensively applied to various tasks, including language modeling, text classification, and information retrieval.One notable application of information entropy in Chinese text analysis is the measurement of average entropy across different linguistic units, such as characters and words. By quantifying the entropy of Chinese characters and words, researchers can assess the predictability and information content of textual data. This, in turn, facilitates tasks such as machine translation, sentiment analysis, and document summarization in the Chinese language domain.This text uses 16 novel corpora, removes stop words, and proves Zipf's law. Then, it estimates the average information entropy of characters and words using unigram, bigram, and trigram models, respectively.

Methodology

Part1: Verification of Zipf's Law

Zipf's Law is an empirical law that describes the relationship between the frequency of a word and its rank in a natural language text. Spec ifically, it states that the frequency of a word is inversely proportional to its rank. In other words, the frequency of the second-ranked word is approximately half that of the first-ranked word, and the frequency of the third-ranked word is approximately one-third that of the first-ranked word, and so on. Detailed steps are as follows:

1. Select Text Dataset: Firstly, choose a large-scale text dataset that covers a wide range of topics to ensure its representativeness. These text data can come from books, articles, news, online texts, and more.
2. Tokenization: Process the selected text dataset by tokenizing it into sequences of words or phrases. For English text, spaces or punctuation marks can be used as separators. For Chinese text, specialized Chinese tokenization tools such as Jieba library are required.
3. Count Word Frequency: Calculate the frequency of occurrence for each word or phrase in the text dataset, which is the number of times each word appears. This can be achieved by iterating through the text dataset and counting the occurrences of each word.
4. Sorting: Sort the words or phrases based on their frequency, usually in descending order, so that words with higher frequencies have higher ranks.
5. Plot Frequency-Rank Graph: Plot the ranks of sorted words or phrases against their corresponding frequencies. The x-axis typically represents the ranks, while the y-axis represents the frequencies. To better visualize the data distribution, logarithmic scales are commonly used for both axes.
6. Observe Relationship: Observe the relationship between ranks and frequencies on the plotted frequency-rank graph. If the graph exhibits an approximate straight-line relationship, where higher-ranked words have higher frequencies, it suggests that the text dataset may follow Zipf's Law.
7. Further Analysis: If the observed trend in the graph aligns with Zipf's Law, further statistical analysis and validation can be conducted. For example, Spearman's rank correlation coefficient can be calculated to quantify the correlation between ranks and frequencies, or curve fitting analysis can be performed to assess the applicability of the law.

Part2: Calculating entropy

M1: Unigram Model

The unigram model, also referred to as the bag-of-words model, is a fundamental approach in natural language processing. It treats each word in a text as an independent entity, disregarding word order and syntactic relationships. The model calculates the probability of each word's occurrence based solely on its frequency in the corpus. The formula to compute the probability $P(\omega_i)$ of a word ω_i occurring in the text is:

$$P(\omega_i) = \frac{Count(\omega_i)}{Total\ number\ of\ words}$$

where $Count(\omega_i)$ represents the number of occurrences of word ω_i in the text, and the denominator is the total number of words in the corpus. This model serves as a foundational concept in various NLP tasks such as text classification, information retrieval, and language generation. However, it lacks context and semantic understanding due to its simplistic nature.

M2: Bigram Model

The bigram model, also known as the bi-gram model, is a statistical language model in natural language processing. Similar to the unigram model, the bigram model is based on adjacent pairs of words in the text.

In the bigram model, the probability of each word depends on the preceding word. Specifically, the bigram model calculates the probability of adjacent word pairs (bigrams), denoted as$P(\omega_i|\omega_{i-1})$, where ω_i is the current word and ω_{i-1} is the previous word. The formula for calculating the probability in the bigram model is:

$$P(\omega_i|\omega_{i-1}) = \frac{Count(\omega_i, \omega_{i-1})}{Count(\omega_{i-1})}$$

Here, $Count(\omega_i, \omega_{i-1})$ represents the number of occurrences of the word pair (ω_i, ω_{i-1}) in the text, and $Count(\omega_{i-1})$ represents the total number of occurrences of the word ω_{i-1}in the text.The bigram model takes into account the local context between words, capturing associations between adjacent words better than the unigram model. As a result, it performs better in some NLP tasks. However, the bigram model still has limitations, such as its inability to capture long-distance dependencies.

M3: Trigram Model

The trigram model, also known as the tri-gram model, is a statistical language model used in natural language processing. It extends the concept of the bigram model by considering sequences of three adjacent words in the text.

In the trigram model, the probability of each word depends on the preceding two words. Specifically, the trigram model calculates the probability of adjacent word triplets (trigrams), denoted as $P(\omega_i|\omega_{i-1}, \omega_{i-2})$, where ω_i is the current word, ω_{i-1} is the previous word, and ω_{i-2} is the word before that. The formula for calculating the probability in the trigram model is:

$$P(\omega_i|\omega_{i-1}, \omega_{i-2}) = \frac{Count(\omega_i, \omega_{i-1}, \omega_{i-2})}{Count(\omega_{i-1}, \omega_{i-2})}$$

Here, $Count(\omega_i, \omega_{i-1}, \omega_{i-2})$ represents the number of occurrences of the word triplet $(\omega_i, \omega_{i-1}, \omega_{i-2})$ in the text, and $Count(\omega_{i-1}, \omega_{i-2})$ represents the total number of occurrences of the word pair$(\omega_{i-1}, \omega_{i-2})$in the text.

The trigram model captures more context than the bigram model, making it potentially more accurate in capturing dependencies between words in a text. However, it also requires a larger amount of training data and may suffer from the data sparsity problem.

Experimental Studies

Part1: Verification of Zipf's Law

Data Preprocessing

Data preprocessing refers to the steps taken to clean and prepare data for analysis. These steps are crucial for ensuring that the data is in a suitable format and quality for further processing. Common data preprocessing steps include:

1. Stopwords Removal: Removing high-frequency words that do not carry significant meaning, such as "这一来", "②", etc. A predefined list of stopwords can be utilized, or customized based on specific requirements.

```
def load_stopwords(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        stopwords = [line.strip() for line in file]
    return stopwords
```

Figure 1 The implementation of stop words in Python

2. Punctuation Removal: Eliminating punctuation marks from the text to obtain cleaner word tokens.

```
def clean_text(text):
    # 去除标点符号和特殊字符
    text = re.sub(r'[^\u4e00-\u9fa5]', '', text)
    return text
```

Figure 2 The implementation of punctuation removal in Python

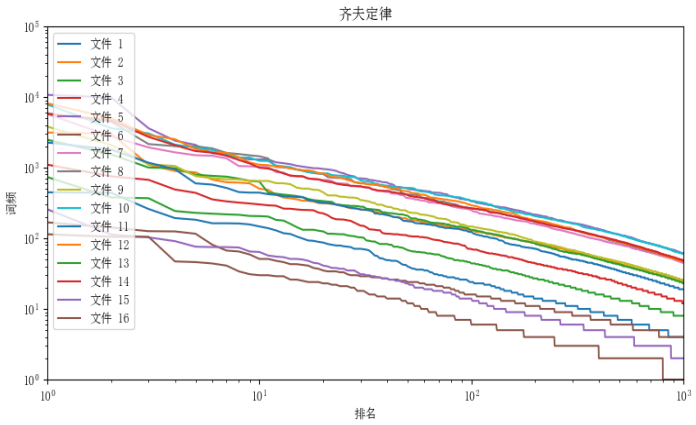
3. Word Segmentation: Dividing the text into individual words or tokens. In Chinese text, tools like jieba are commonly used for word segmentation.

```
def calculate_word_frequencies(text, stopwords):
    # 使用jieba进行分词
    words = jieba.lcut(text)
    # 去除停用词
    words = [word for word in words if word not in stopwords]
    word_counts = collections.Counter(words)
    return word_counts
```

Figure 3 The implementation of Verification processing of Zipf's Law jieba in Python

Verification Processing

The Python script first reads a Chinese text file '1.txt' for analysis, although it can be replaced with any text file of choice. Next, the script loads a stop words list, which typically includes common words with little semantic meaning in text analysis. The text is then cleaned by removing stop words, punctuation, and other unnecessary characters. The cleaned text is used to calculate word frequencies, i.e., the number of times each word appears in the text. Subsequently, the top N most frequent words are selected, with N set to 50 in this case. Finally, a chart is generated based on the frequencies and ranks of these words to validate whether they adhere to Zipf's Law. The x-axis of the chart represents the rank of words, while the y-axis represents their frequencies. Both axes are scaled logarithmically for clearer visualization of the relationship between rank and frequency. Figure 4 displays the validation curves of the 16 novels along with the correspondence between the novels and their file numbers.



白马啸西风	1. txt
碧血剑	2. txt
飞狐外传	3. txt
连城诀	4. txt
鹿鼎记	5. txt
三十三剑客图	6. txt
射雕英雄传	7. txt
神雕侠侣	8. txt
书剑恩仇录	9. txt
天龙八部	10. txt
侠客行	11. txt
笑傲江湖	12. txt
雪山飞狐	13. txt
倚天屠龙记	14. txt
鸳鸯刀	15. txt
越女剑	16. txt

Figure 4 The validation results of 16 novels

Part2: Calculating entropy

Part1: Verification of Zipf's Law

Firstly, two functions are defined, `calculate_entropy(text)` and `calculate_char_entropy(text)`, which are used to compute the word-level and character-level entropy of the text, respectively. Then, an empty DataFrame is created to store the results. Subsequently, a loop iterates over 16 files (assuming filenames range from 1.txt to 16.txt). If a file exists, its content is read, and the corresponding functions are called to compute the entropy. The calculated results are appended to the DataFrame. Finally, the results from the DataFrame are printed and saved as a CSV file. Figure 5 illustrates the character-level entropy and word-level entropy under the unigram model.

	文件名	词级别信息熵	字级别信息熵
0	1.txt	8.773	8.297
1	2.txt	10.363	9.029
2	3.txt	10.201	8.905
3	4.txt	9.706	8.726
4	5.txt	9.948	8.813
5	6.txt	10.289	9.195
6	7.txt	10.278	8.961
7	8.txt	10.275	8.943
8	9.txt	10.275	9.007
9	10.txt	10.237	8.943
10	11.txt	9.839	8.737
11	12.txt	10.047	8.810
12	13.txt	9.811	8.784
13	14.txt	10.333	8.970
14	15.txt	8.904	8.426
15	16.txt	8.596	8.232

白马啸西风	1. txt
碧血剑	2. txt
飞狐外传	3. txt
连城诀	4. txt
鹿鼎记	5. txt
三十三剑客图	6. txt
射雕英雄传	7. txt
神雕侠侣	8. txt
书剑恩仇录	9. txt
天龙八部	10. txt
侠客行	11. txt
笑傲江湖	12. txt
雪山飞狐	13. txt
倚天屠龙记	14. txt
鸳鸯刀	15. txt
越女剑	16. txt

Figure 5 The entropy under the unigram model.

M2: Bigram Model

The main distinction between the code for the bigram model and the unigram model lies in how they handle the sequence of words or characters. In the bigram model, consecutive pairs of words or characters are considered as units for analysis, while in the unigram model, each word or character is treated independently. Consequently, the bigram model constructs bigrams by pairing consecutive elements and calculates entropy based on the probabilities of these pairs, whereas the

unigram model directly calculates entropy using individual elements without forming pairs. Figure 6 illustrates the character-level entropy and word-level entropy under the bigram model.

	文件名	二元词级别信息熵	二元字级别信息熵	白马啸西风	1.txt
0	1.txt	12.911	12.668	碧血剑	2.txt
1	2.txt	15.606	14.571	飞狐外传	3.txt
2	3.txt	15.364	14.315	连城诀	4.txt
3	4.txt	14.508	13.783	鹿鼎记	5.txt
4	5.txt	15.449	14.297	三十三剑客图	6.txt
5	6.txt	14.303	14.105	射雕英雄传	7.txt
6	7.txt	15.763	14.578	神雕侠侣	8.txt
7	8.txt	15.869	14.610	书剑恩仇录	9.txt
8	9.txt	15.503	14.436	天龙八部	10.txt
9	10.txt	15.783	14.570	侠客行	11.txt
10	11.txt	14.773	13.921	笑傲江湖	12.txt
11	12.txt	15.507	14.263	雪山飞狐	13.txt
12	13.txt	14.265	13.699	倚天屠龙记	14.txt
13	14.txt	15.831	14.568	鸳鸯刀	15.txt
14	15.txt	12.501	12.524	越女剑	16.txt
15	16.txt	11.756	11.878		

Figure 6 The entropy under the bigram model

M3: Trigram Model

The main difference between the trigram model and the unigram and bigram models in code lies in how they handle sequences of words or characters. In the trigram model, consecutive sequences of three words or characters are considered as units for analysis instead of individual words or characters. Therefore, before calculating entropy, it's necessary to construct trigrams consisting of three consecutive words or characters, rather than unigrams or bigrams. This leads to differences in the code implementation, including the choice of data structures and adjustments to the calculation process. Figure 7 illustrates the character-level entropy and word-level entropy under the trigram model.

	文件名	三元词级别信息熵	三元字级别信息熵	白马啸西风	1.txt
0	1.txt	14.420	14.585	碧血剑	2.txt
1	2.txt	17.388	17.154	飞狐外传	3.txt
2	3.txt	17.201	16.917	连城诀	4.txt
3	4.txt	16.186	16.127	鹿鼎记	5.txt
4	5.txt	17.839	17.318	三十三剑客图	6.txt
5	6.txt	15.113	15.491	射雕英雄传	7.txt
6	7.txt	17.936	17.513	神雕侠侣	8.txt
7	8.txt	18.069	17.607	书剑恩仇录	9.txt
8	9.txt	17.375	17.052	天龙八部	10.txt
9	10.txt	18.068	17.607	侠客行	11.txt
10	11.txt	16.608	16.428	笑傲江湖	12.txt
11	12.txt	17.798	17.219	雪山飞狐	13.txt
12	13.txt	15.617	15.703	倚天屠龙记	14.txt
13	14.txt	17.987	17.506	鸳鸯刀	15.txt
14	15.txt	13.634	13.986	越女剑	16.txt
15	16.txt	12.679	13.095		

Figure 7 The entropy under the trigram model

Conclusions

Under the unigram, bigram, and trigram models, the average character entropy and word entropy calculated from 16 Jin Yong novels are represented in the following table and Figure 8.

中文平均信息熵						
	一元模型信息熵		二元模型信息熵		三元模型信息熵	
	词级别	字级别	词级别	字级别	词级别	字级别
白马啸西风	8.773	8.297	12.911	12.668	14.42	14.585
碧血剑	10.363	9.029	15.606	14.571	17.388	17.154
飞狐外传	10.201	8.905	15.364	14.315	17.201	16.917
连城诀	9.706	8.726	14.508	13.783	16.186	16.127
鹿鼎记	9.948	8.813	15.449	14.297	17.839	17.318
三十三剑客图	10.289	8.961	14.303	14.105	15.113	15.491
射雕英雄传	10.278	8.961	15.763	14.578	17.936	17.513
神雕侠侣	10.275	8.943	15.869	14.61	18.069	17.607
书剑恩仇录	10.275	9.007	15.503	14.436	17.375	17.052
天龙八部	10.237	8.943	15.783	14.57	18.068	17.607
侠客行	9.839	8.737	14.773	13.921	16.608	16.428
笑傲江湖	10.047	8.81	15.507	14.263	17.798	17.219
雪山飞狐	9.811	8.784	14.265	13.699	15.617	15.703
倚天屠龙记	10.333	8.97	15.831	14.568	17.987	17.506
鸳鸯刀	8.904	8.426	12.5	12.524	13.634	13.986
越女剑	8.596	8.232	11.756	11.878	12.679	13.095

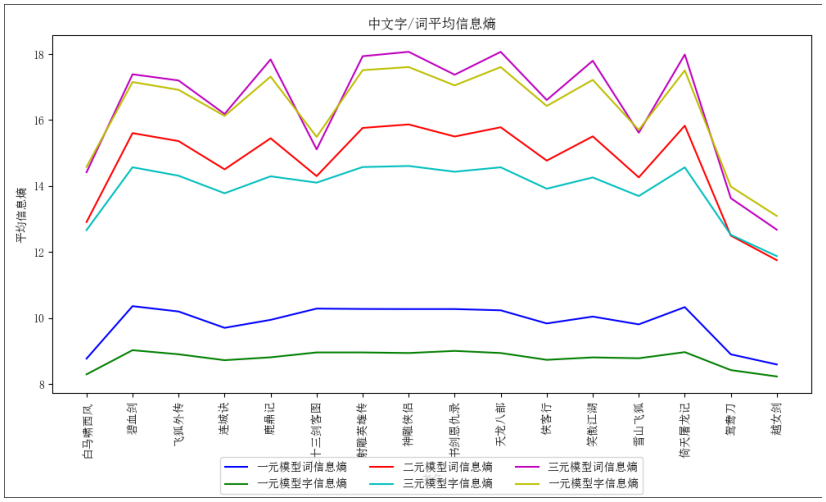


Figure 8 The average character/word entropy summary of the 16 novels.

Regardless of whether it's using a unigram, bigram, or trigram language model, the trends of entropy variation for characters and words in each work are consistent. This might suggest that Jin Yong's linguistic style is similar across his works. Additionally, we observe that the entropy of a unigram language model is greater than that of a bigram language model, and the entropy of a bigram language model is greater than that of a trigram language model. This indicates that as the sequence length increases, the expression becomes more precise. It's worth noting that in a unigram language model, the entropy of words is greater than that of characters, while in bigram and trigram language models, the entropy of words is smaller than that of characters.

The assignment has provided me with valuable insights into unigram, bigram, and trigram language models in natural language processing. Through practical exercises, I learned how to use Python to calculate the entropy of text data and analyze the linguistic characteristics of Jin Yong's novels. Additionally, I gained experience in utilizing commonly used Python libraries such as jieba, collections, and pandas for text processing, frequency counting, and language model construction. These practical exercises have enhanced my familiarity with the Python programming language and

the basic principles and methods of natural language processing, providing me with valuable experience for both learning and work.

References

[1] Brown P F, Della Pietra S A, Della Pietra J, et al. An estimate of an upper bound for the entropy of English[J]. Computational Linguistics, 1992, 18(1): 31-40.