Report of Deep Learning for Natural Langauge Processing

Shanghua Li
zy2403112@buaa.edu.com

Abstract

Entropy is a crucial measure of information uncertainty, widely used in natural language processing, data compression, and text analysis. This report explores methods for calculating the entropy of Chinese and English texts, covering entropy calculations at different granularities such as letters, words, and phrases, and comparing the results. Additionally, we introduce text preprocessing techniques, including stopword removal, stemming, tokenization, and n-gram entropy computation. Finally, we discuss the impact of traditional-to-simplified Chinese conversion on entropy calculation and compare entropy results between Chinese and English texts.

Introduction

This article mainly introduces the principles and calculation methods of information entropy in both Chinese and English and conducts the following four comparative experiments:

Calculation and comparison of the information entropy of English letters and words

Calculation of the information entropy of English word n-grams

Calculation and comparison of the information entropy of Chinese characters and words

Calculation of the information entropy of Chinese word n-grams

Finally, conclusions and analyses are presented.

Methodology

English Entropy Calculation Method

The basic formula for entropy calculation is:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

where $P(x_i)$ represents the probability of occurrence of the iith character or word, and nn is the total number of unique characters or words.

Letter vs. Word Entropy Calculation and Comparison

Letter Entropy

Letter entropy is computed based on the probability distribution of different letters (A-Z) in English texts. Typically, uppercase and lowercase letters are unified, and punctuation and spaces are removed.

Word Entropy

Word entropy is calculated based on word frequency distributions. Compared to letter entropy, word entropy is usually higher due to the larger variety of words.

Process Stopwords

Stopwords (such as "the," "is," and "of") appear frequently in text but contribute little to meaning. Removing stopwords lowers entropy because it reduces vocabulary diversity.

Stemming

Stemming normalizes different word forms, such as converting "running" and "run" into the same base form. This process lowers entropy by reducing the number of distinct words.

Tokenization

Tokenization refers to splitting text into its smallest linguistic units (letters or words). Different tokenization methods may affect entropy results, such as space-based segmentation versus more sophisticated approaches.

Word n-grams Entropy Calculation

n-grams refer to sequences of nn consecutive words or letters. Examples include:

1-gram (unigram): "hello".Calculated as

$$P(w) = \frac{\text{Frequency of } w}{\text{Total number of words in the corpus}}$$

For **N= 1**, it is jus the baf-of-worlds model and the probability of **“I love AI” = P(I)P(love)P(AI)**

2-gram (bigram): "hello world".Calculated as

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=2}^{n} P(w_i|w_{i-1}) \qquad P(w_i|w_{i-1}) = \frac{\text{Frequency of bigram } (w_{i-1}, w_i)}{\text{Frequency of word } w_{i-1}}$$

3-gram (trigram): "hello world today".Calculated as

$$P(w_1, w_2, w_3) = P(w_1) \times \prod_{i=2}^{3} P(w_i|w_{i-1}, w_{i-2})$$

As nn increases, the number of unique n-grams rises, leading to higher entropy. The calculation method is similar to word entropy but based on n-gram frequency distributions.Based on what we learned in the course we only count the three n-grams in the report.

Chinese Entropy Calculation Method

Since Chinese is a logographic language without explicit word boundaries, entropy calculation differs from English.

Character vs. Word Entropy Calculation and Comparison

Character Entropy

Chinese character entropy is computed similarly to English letter entropy, based on the probability distribution of unique characters.

Word Entropy

After segmentation, word entropy is typically higher than character entropy because the number of words exceeds that of individual characters.

Stopword Removal

Chinese stopwords (such as "的," "是," "在") are highly frequent. Removing them reduces entropy.

Traditional-to-Simplified Conversion

For processing traditional Chinese texts, conversion to simplified Chinese is necessary to ensure consistent statistical analysis and prevent duplicate calculations of synonymous words, which could otherwise inflate entropy.

Segmentation

Chinese word segmentation (tokenization) is crucial for entropy calculation. Common methods include:

Dictionary-based segmentation (e.g., Jieba)

Statistical and machine learning-based segmentation (e.g., HMM, deep learning)

Different segmentation techniques impact entropy results.The lexicon Jieba-based disambiguation is used here

Word n-grams Entropy Calculation

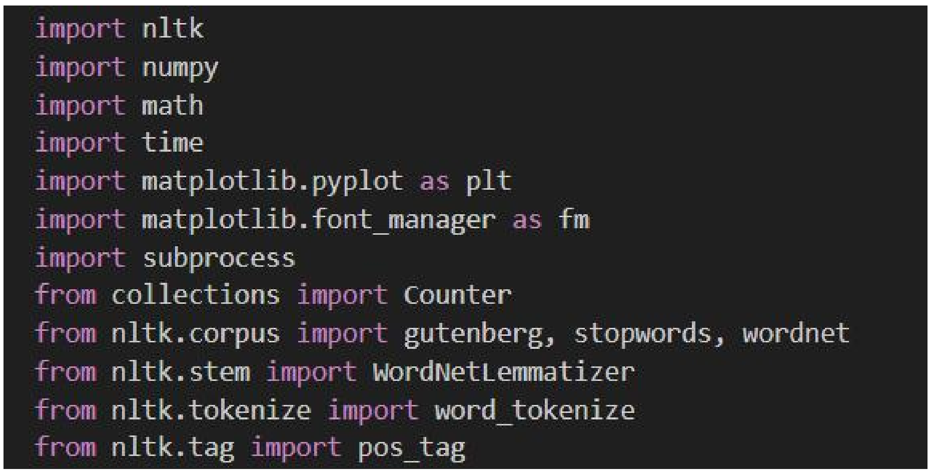
Similar to English, Chinese n-grams (e.g., 2-gram: "北京 大学") capture phrase relationships. Generally, as nn increases, entropy rises.

Experimental Studies

The equipment used in this experiment is the laboratory workstation, the system is ubuntu system, so it will be slightly different from the preparation in windows.

The following is a brief explanation of the process of calculating the information entropy of English letters and words, and other calculations will not be repeated.

Importing Required Libraries



```python
import nltk
import numpy
import math
import time
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
import subprocess
from collections import Counter
from nltk.corpus import gutenberg, stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
```

NLTK (Natural Language Toolkit): Used for text processing, including tokenization, part-of-speech tagging, and stopword removal.

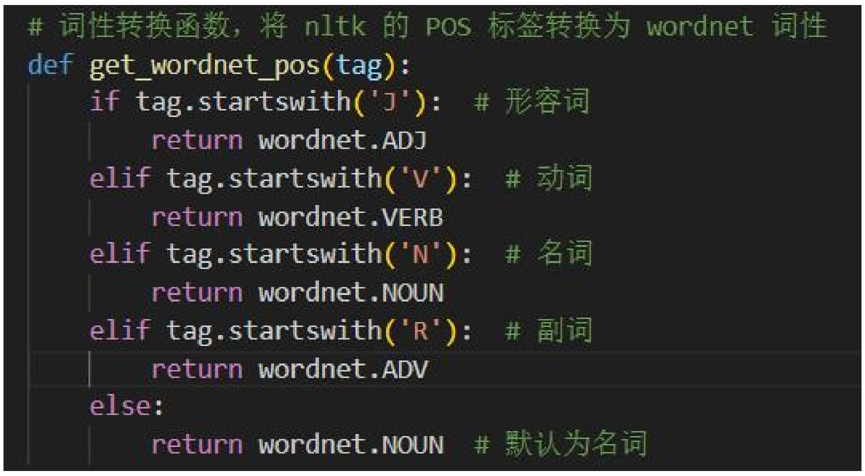
math: Provides mathematical functions (e.g., log2 for entropy calculation).

time: Measures execution time.

matplotlib.pyplot: Generates histograms of letter and word frequency distributions.

collections.Counter: Counts letter and word occurrences.

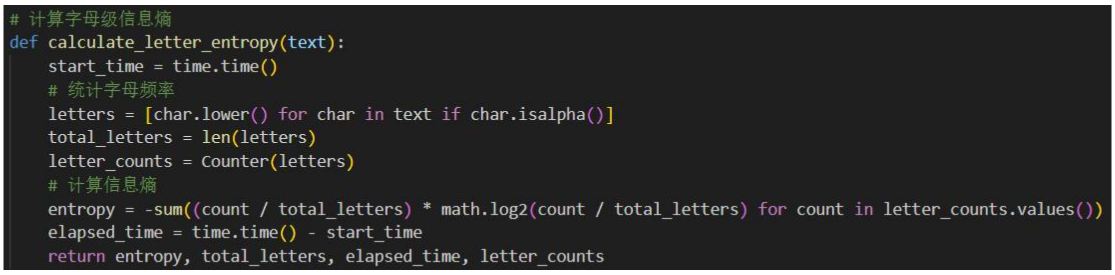
Function for Converting POS Tags



```python
# 词性转换函数，将 nltk 的 POS 标签转换为 wordnet 词性
def get_wordnet_pos(tag):
    if tag.startswith('J'):  # 形容词
        return wordnet.ADJ
    elif tag.startswith('V'):  # 动词
        return wordnet.VERB
    elif tag.startswith('N'):  # 名词
        return wordnet.NOUN
    elif tag.startswith('R'):  # 副词
        return wordnet.ADV
    else:
        return wordnet.NOUN  # 默认为名词
```

Since NLTK's lemmatize() function requires WordNet POS tags, this function converts NLTK POS tags (e.g., VB for verbs, NN for nouns) to WordNet-compatible formats.

Calculating Letter-Level Entropy



```python
# 计算字母级信息熵
def calculate_letter_entropy(text):
    start_time = time.time()
    # 统计字母频率
    letters = [char.lower() for char in text if char.isalpha()]
    total_letters = len(letters)
    letter_counts = Counter(letters)
    # 计算信息熵
    entropy = -sum((count / total_letters) * math.log2(count / total_letters) for count in letter_counts.values())
    elapsed_time = time.time() - start_time
    return entropy, total_letters, elapsed_time, letter_counts
```

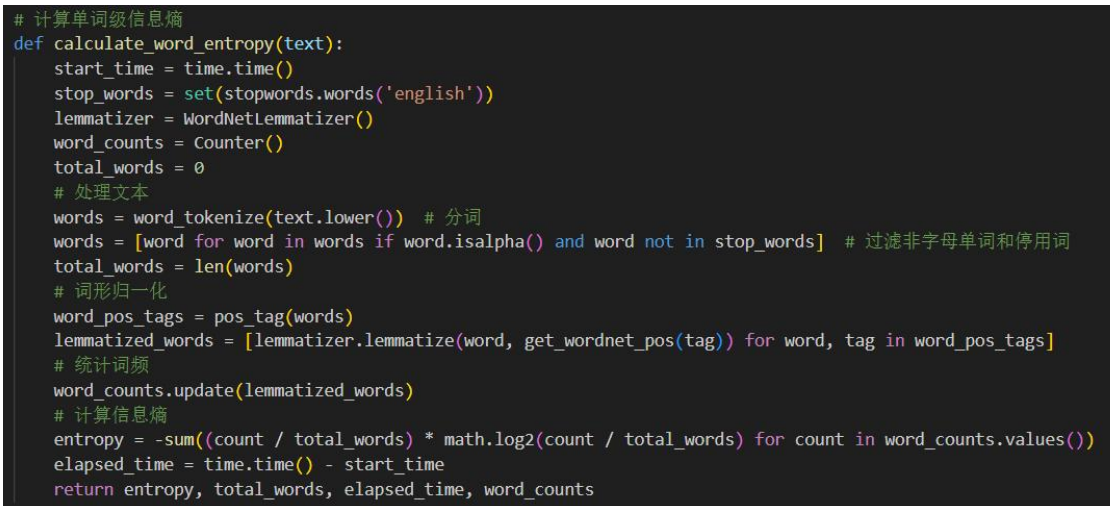
Extracts all letters (removing punctuation, digits, spaces, etc.).

Counts letter frequency using `Counter()`.

Computes entropy

Returns entropy, total letter count, computation time, and letter frequency data.

Calculating Word-Level Entropy



```python
# 计算单词级信息熵
def calculate_word_entropy(text):
    start_time = time.time()
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    word_counts = Counter()
    total_words = 0
    # 处理文本
    words = word_tokenize(text.lower())  # 分词
    words = [word for word in words if word.isalpha() and word not in stop_words]  # 过滤非字母单词和停用词
    total_words = len(words)
    # 词形归一化
    word_pos_tags = pos_tag(words)
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(tag)) for word, tag in word_pos_tags]
    # 统计词频
    word_counts.update(lemmatized_words)
    # 计算信息熵
    entropy = -sum((count / total_words) * math.log2(count / total_words) for count in word_counts.values())
    elapsed_time = time.time() - start_time
    return entropy, total_words, elapsed_time, word_counts
```

Retrieves stopwords using `stopwords.words('english')`.

Tokenizes words (`word_tokenize(text.lower())`).

Filters out non-alphabetic words and stopwords.

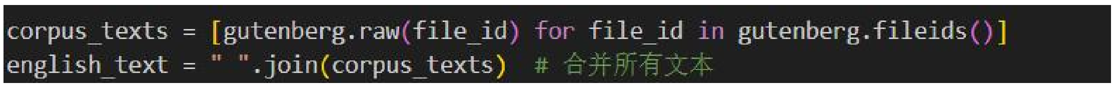
Lemmatization:

1. `pos_tag()` assigns POS tags.
2. `lemmatizer.lemmatize(word, get_wordnet_pos(tag))` converts words to their base forms (e.g., `running → run`).

Counts word occurrences and computes entropy.

Returns entropy, total word count, computation time, and word frequency data.

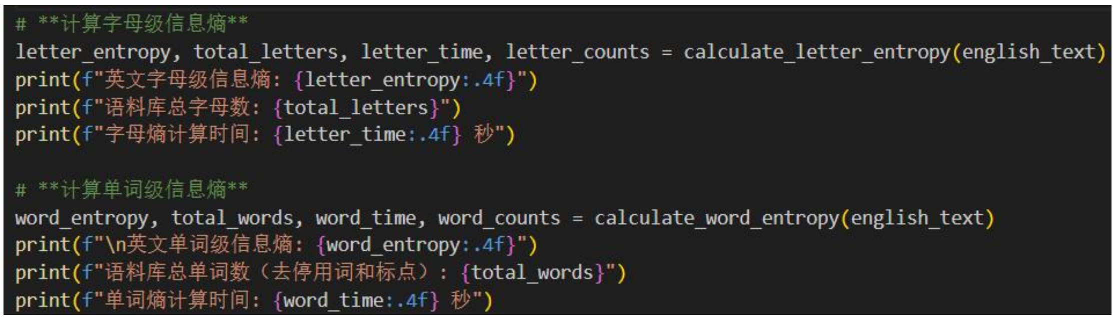
Reading the Corpus Text



```python
corpus_texts = [gutenberg.raw(file_id) for file_id in gutenberg.fileids()]
english_text = " ".join(corpus_texts)  # 合并所有文本
```

Extracts all text from the Gutenberg corpus and merges it into a single string.

Computing and Printing Entropy

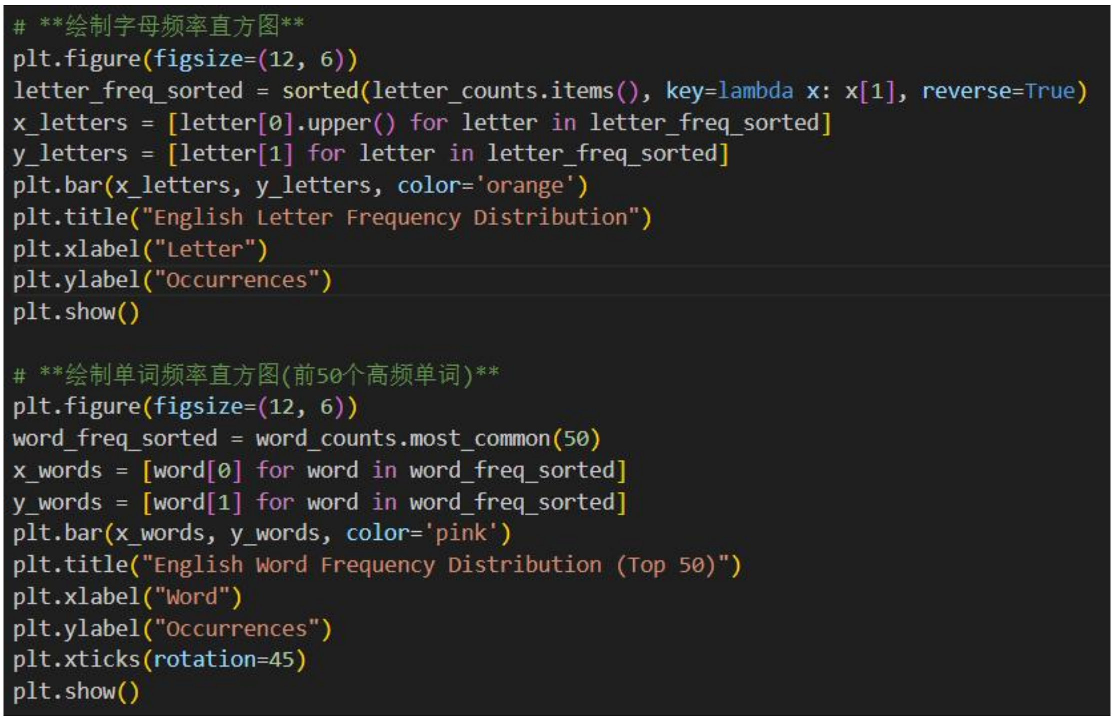


```python
# **计算字母级信息熵**
letter_entropy, total_letters, letter_time, letter_counts = calculate_letter_entropy(english_text)
print(f"英文字母级信息熵: {letter_entropy:.4f}")
print(f"语料库总字母数: {total_letters}")
print(f"字母熵计算时间: {letter_time:.4f} 秒")

# **计算单词级信息熵**
word_entropy, total_words, word_time, word_counts = calculate_word_entropy(english_text)
print(f"\n英文单词级信息熵: {word_entropy:.4f}")
print(f"语料库总单词数（去停用词和标点）: {total_words}")
print(f"单词熵计算时间: {word_time:.4f} 秒")
```

Calculates and prints letter entropy.

Calculates and prints word entropy (excluding stopwords and punctuation).

Plotting Frequency Histogram



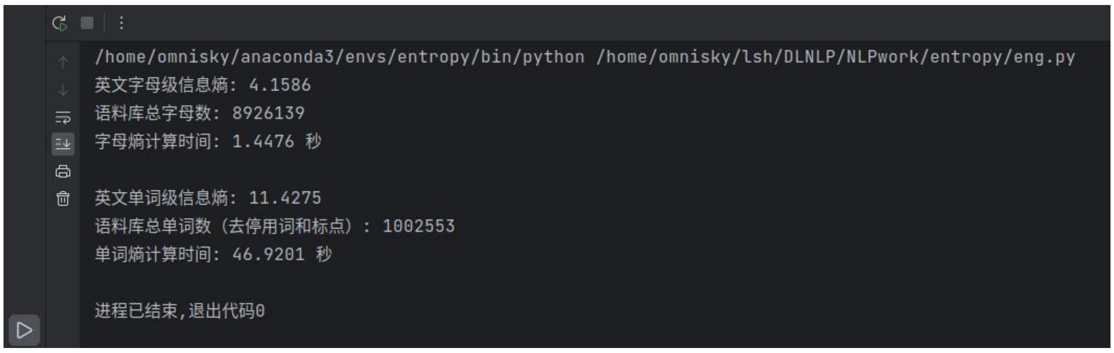
```python
# **绘制字母频率直方图**
plt.figure(figsize=(12, 6))
letter_freq_sorted = sorted(letter_counts.items(), key=lambda x: x[1], reverse=True)
x_letters = [letter[0].upper() for letter in letter_freq_sorted]
y_letters = [letter[1] for letter in letter_freq_sorted]
plt.bar(x_letters, y_letters, color='orange')
plt.title("English Letter Frequency Distribution")
plt.xlabel("Letter")
plt.ylabel("Occurrences")
plt.show()

# **绘制单词频率直方图(前50个高频单词)**
plt.figure(figsize=(12, 6))
word_freq_sorted = word_counts.most_common(50)
x_words = [word[0] for word in word_freq_sorted]
y_words = [word[1] for word in word_freq_sorted]
plt.bar(x_words, y_words, color='pink')
plt.title("English Word Frequency Distribution (Top 50)")
plt.xlabel("Word")
plt.ylabel("Occurrences")
plt.xticks(rotation=45)
plt.show()
```

Sorts letters by frequency and plots a bar chart.

Plots the top 50 most frequent words in a histogram and rotates x-axis labels for better readability.

Experimental result

Letter vs. Word Entropy Calculation and Comparison



```
/home/omnisky/anaconda3/envs/entropy/bin/python /home/omnisky/lsh/DLNLP/NLPwork/entropy/eng.py
英文字母级信息熵: 4.1586
语料库总字母数: 8926139
字母熵计算时间: 1.4476 秒

英文单词级信息熵: 11.4275
语料库总单词数（去停用词和标点）: 1002553
单词熵计算时间: 46.9201 秒

进程已结束,退出代码0
```

Figure 1： The result of the program execution

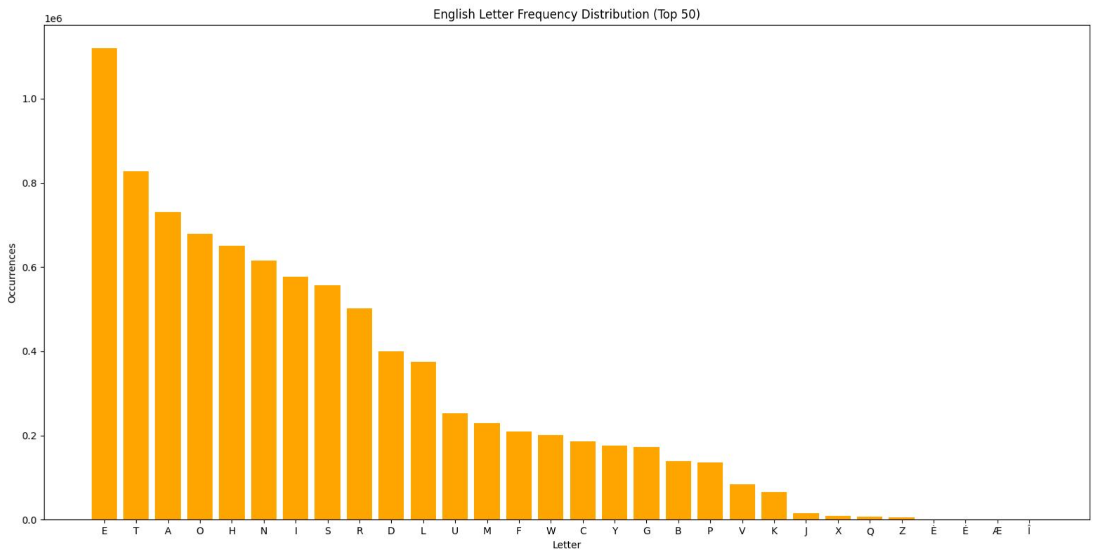


Figure 2： English Letter Frequency Distribution

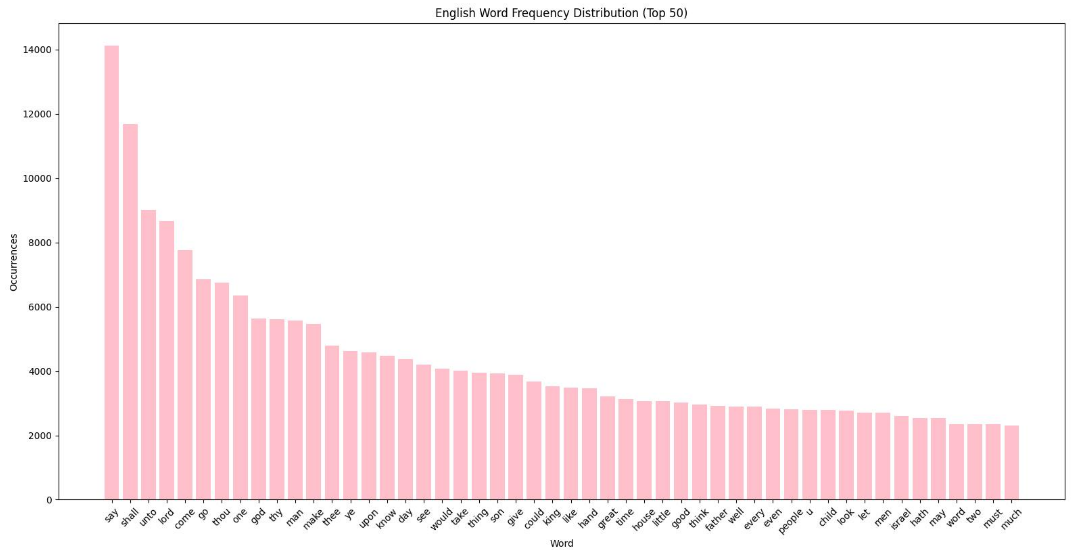
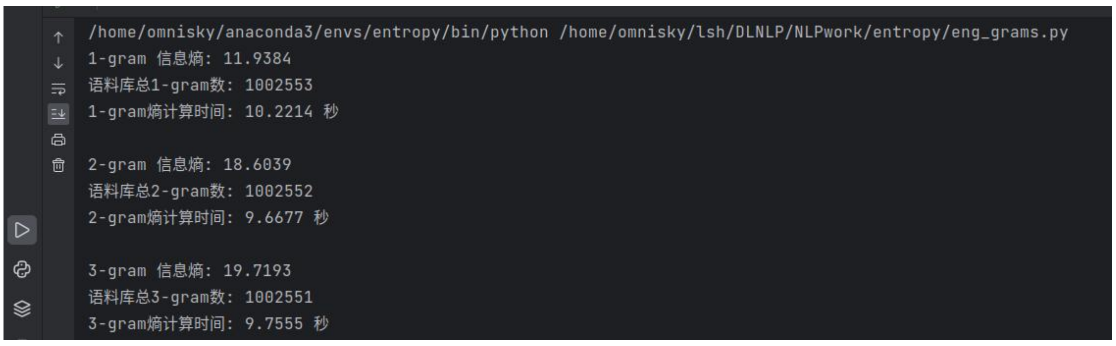


Figure 3： English Word Frequency Distribution (Top 50)

Word n-grams Entropy Calculation



```
/home/omnisky/anaconda3/envs/entropy/bin/python /home/omnisky/lsh/DLNLP/NLPwork/entropy/eng_grams.py
1-gram 信息熵: 11.9384
语料库总1-gram数: 1002553
1-gram熵计算时间: 10.2214 秒

2-gram 信息熵: 18.6039
语料库总2-gram数: 1002552
2-gram熵计算时间: 9.6677 秒

3-gram 信息熵: 19.7193
语料库总3-gram数: 1002551
3-gram熵计算时间: 9.7555 秒
```

Figure 4: The result of the program execution

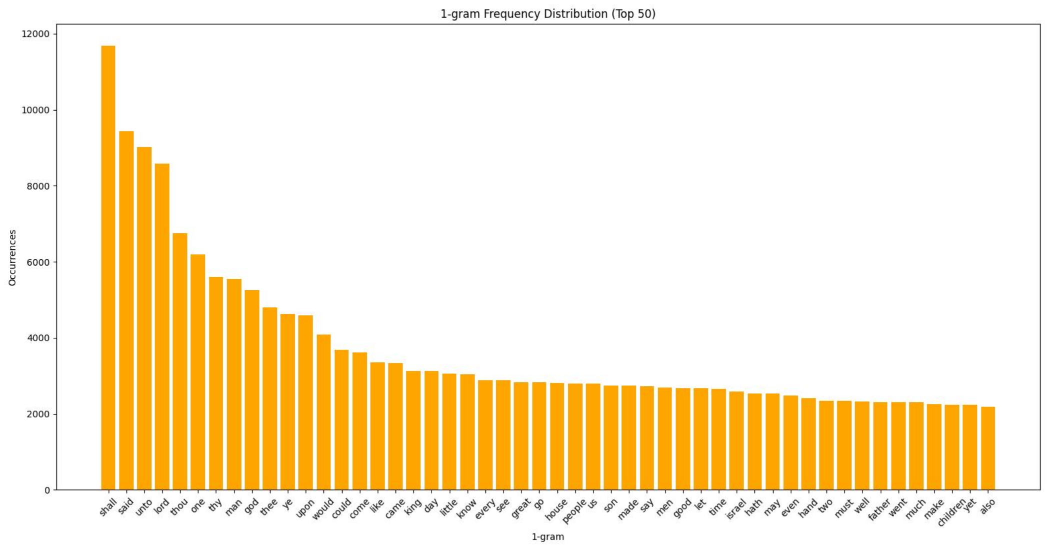


Figure 5: 1-gram Frequency Distribution (Top 50)

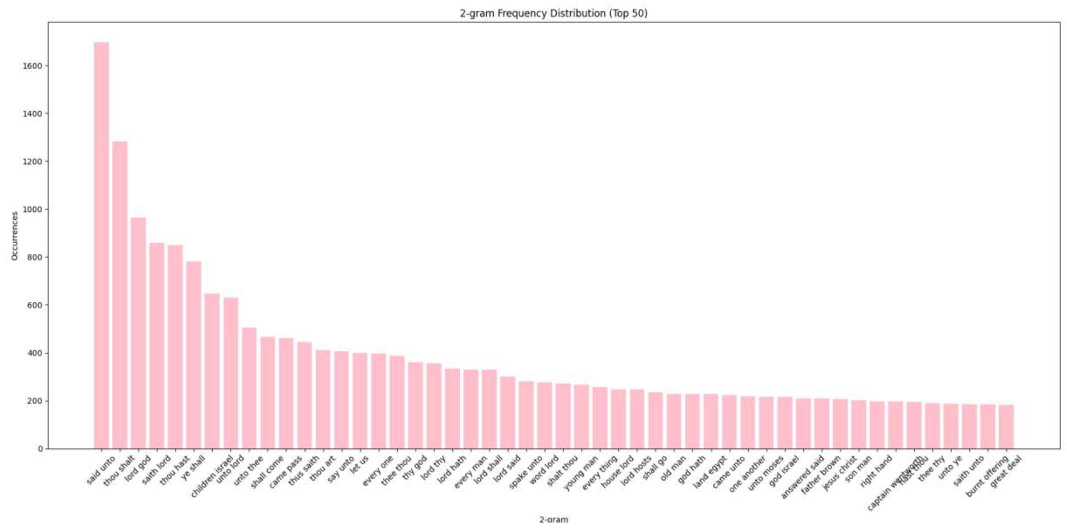


Figure 6: 2-gram Frequency Distribution (Top 50)

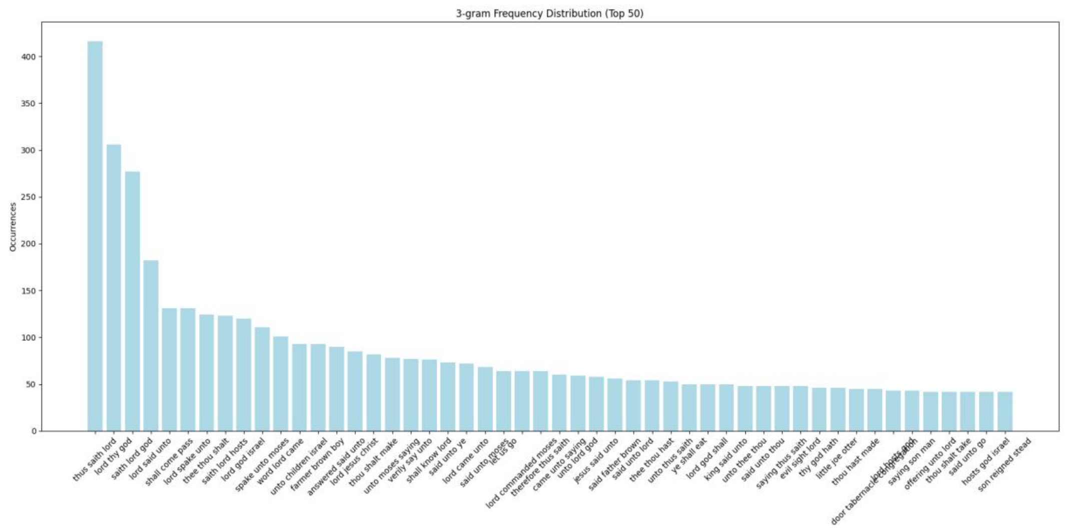
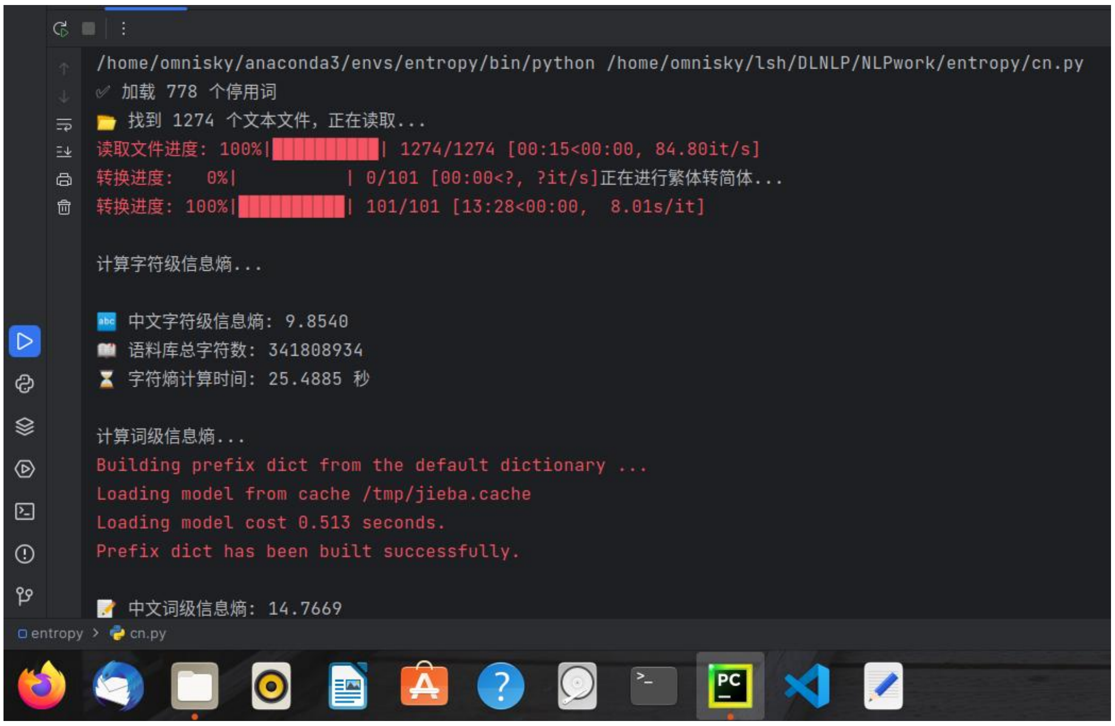


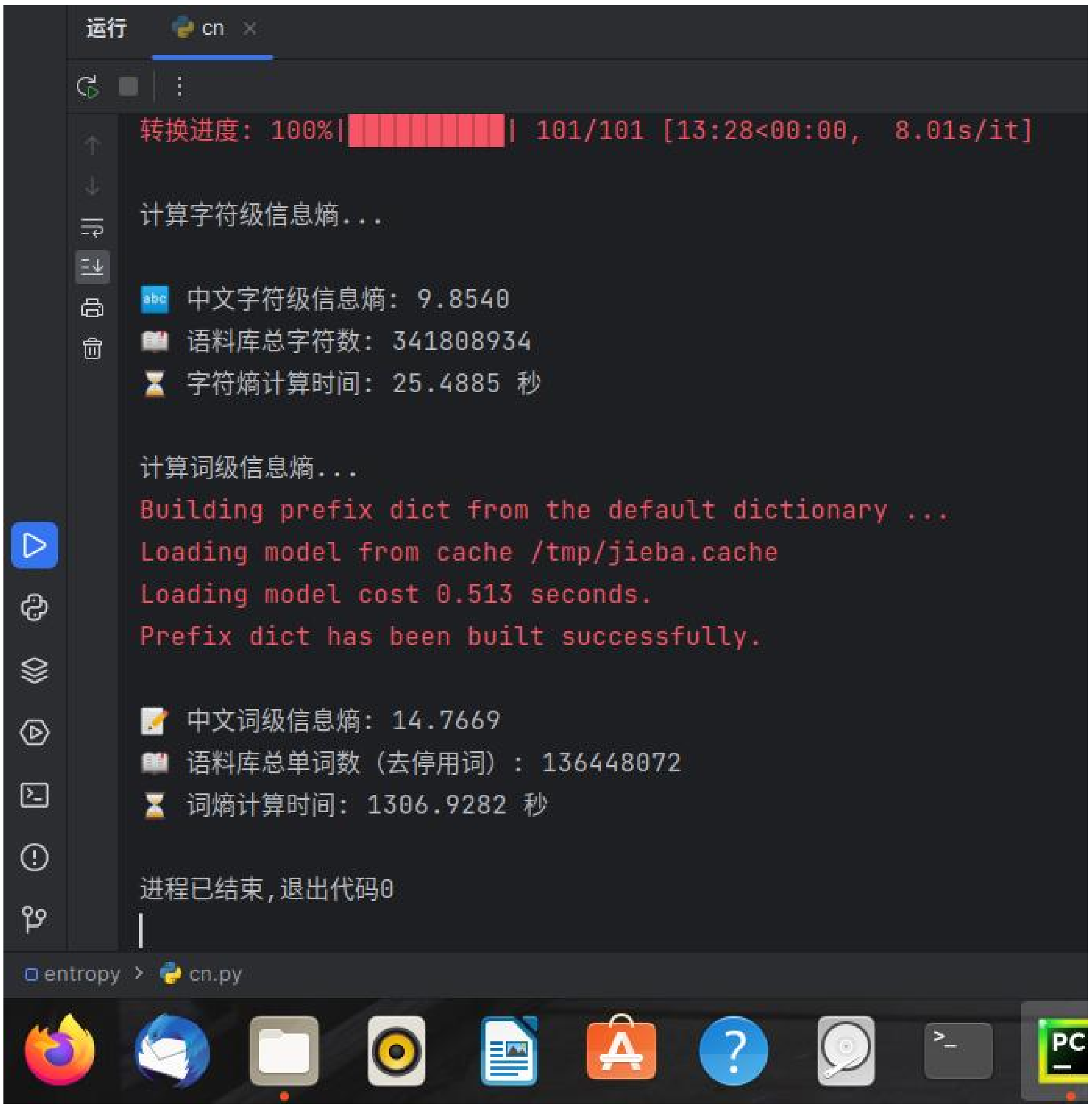
Figure 7: 3-gram Frequency Distribution (Top 50)

Character vs. Word Entropy Calculation and Comparison



```
/home/omnisky/anaconda3/envs/entropy/bin/python /home/omnisky/lsh/DLNLP/NLPwork/entropy/cn.py
✅ 加载 778 个停用词
📂 找到 1274 个文本文件，正在读取...
读取文件进度: 100%|██████████| 1274/1274 [00:15<00:00, 84.80it/s]
转换进度:   0%|          | 0/101 [00:00<?, ?it/s]正在进行繁体转简体...
转换进度: 100%|██████████| 101/101 [13:28<00:00,  8.01s/it]

计算字符级信息熵...

🔤 中文字符级信息熵: 9.8540
📖 语料库总字符数: 341808934
⏳ 字符熵计算时间: 25.4885 秒

计算词级信息熵...
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.513 seconds.
Prefix dict has been built successfully.

📝 中文词级信息熵: 14.7669
```

Figure 8: The result of the program execution



```
转换进度: 100%|██████████| 101/101 [13:28<00:00,  8.01s/it]

计算字符级信息熵...

🔤 中文字符级信息熵: 9.8540
📖 语料库总字符数: 341808934
⏳ 字符熵计算时间: 25.4885 秒

计算词级信息熵...
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.513 seconds.
Prefix dict has been built successfully.

📝 中文词级信息熵: 14.7669
📖 语料库总单词数（去停用词）: 136448072
⏳ 词熵计算时间: 1306.9282 秒

进程已结束,退出代码0
```

Figure 9: The result of the program execution

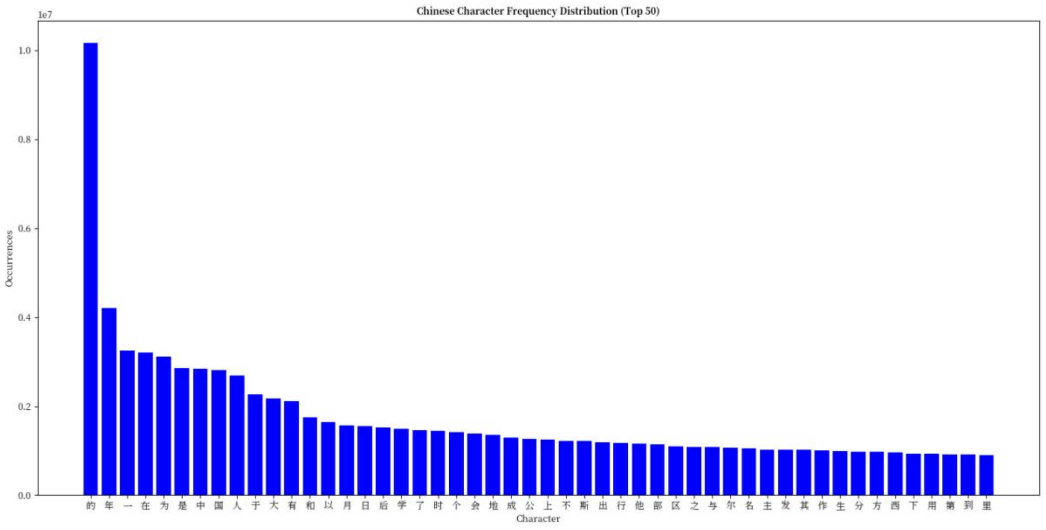


Figure 10: Chinese Character Frequency Distribution (Top 50)

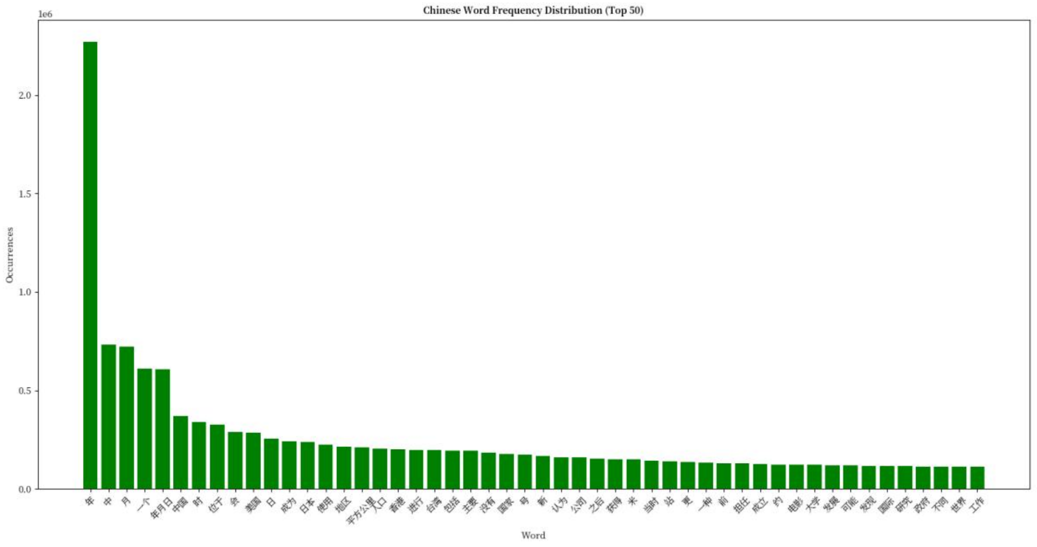
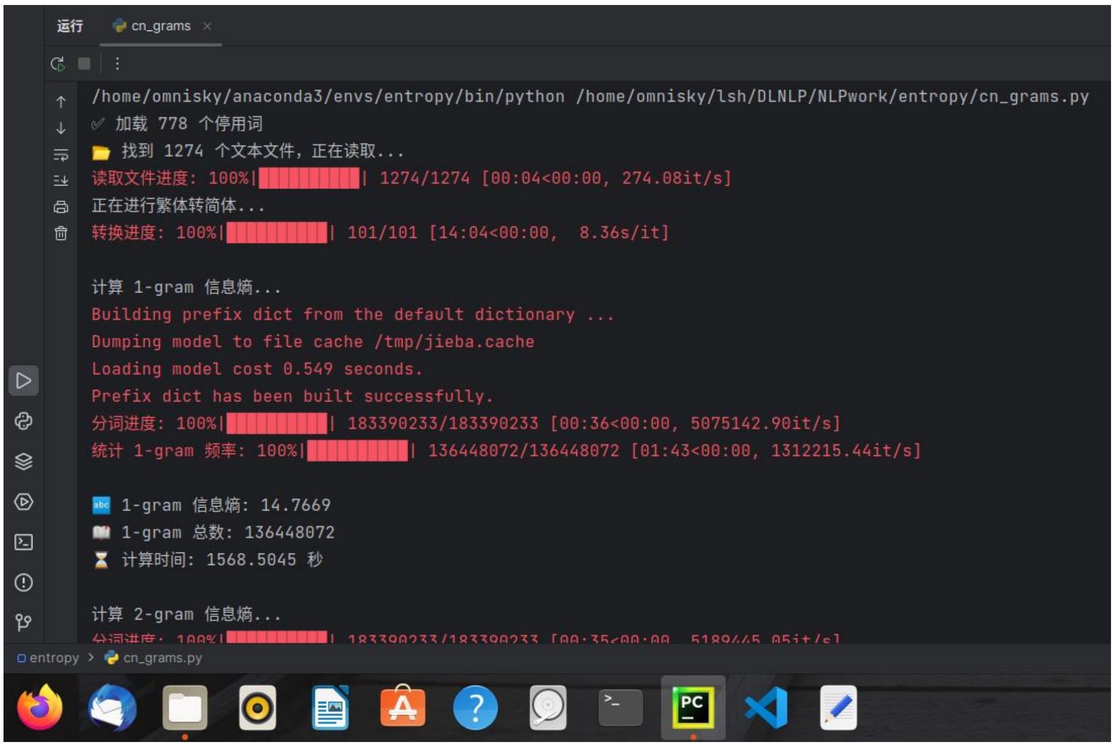


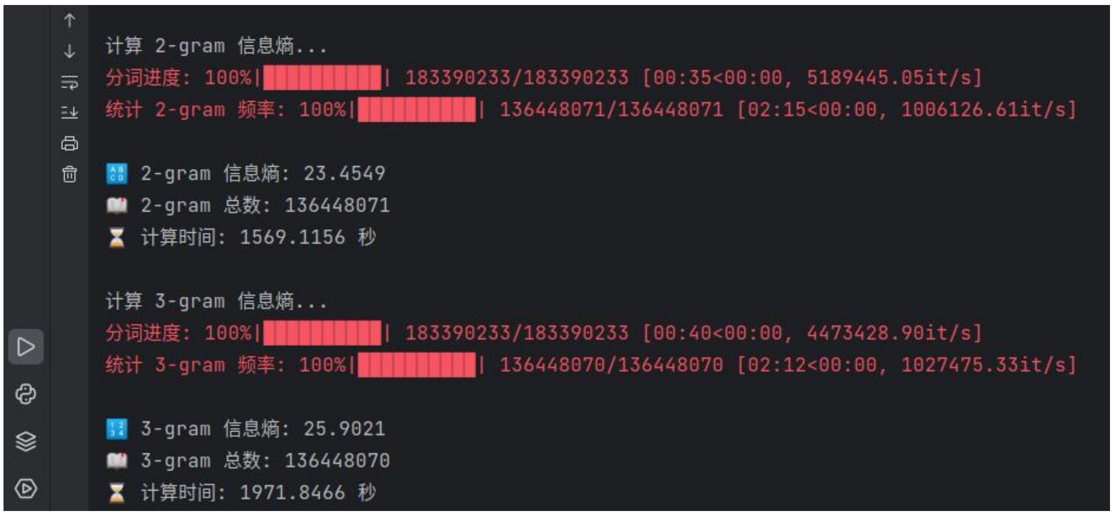
Figure 11: Chinese Word Frequency Distribution (Top 50)

Word n-grams Entropy Calculation



```
/home/omnisky/anaconda3/envs/entropy/bin/python /home/omnisky/lsh/DLNLP/NLPwork/entropy/cn_grams.py
✅ 加载 778 个停用词
📂 找到 1274 个文本文件，正在读取...
读取文件进度: 100%|██████████| 1274/1274 [00:04<00:00, 274.08it/s]
正在进行繁体转简体...
转换进度: 100%|██████████| 101/101 [14:04<00:00,  8.36s/it]

计算 1-gram 信息熵...
Building prefix dict from the default dictionary ...
Dumping model to file cache /tmp/jieba.cache
Loading model cost 0.549 seconds.
Prefix dict has been built successfully.
分词进度: 100%|██████████| 183390233/183390233 [00:36<00:00, 5075142.90it/s]
统计 1-gram 频率: 100%|██████████| 136448072/136448072 [01:43<00:00, 1312215.44it/s]

🔤 1-gram 信息熵: 14.7669
📖 1-gram 总数: 136448072
⏳ 计算时间: 1568.5045 秒

计算 2-gram 信息熵...
分词进度: 100%|██████████| 183390233/183390233 [00:35<00:00, 5189445.05it/s]
```

Figure 12: The result of the program execution



```
计算 2-gram 信息熵...
分词进度: 100%|██████████| 183390233/183390233 [00:35<00:00, 5189445.05it/s]
统计 2-gram 频率: 100%|██████████| 136448071/136448071 [02:15<00:00, 1006126.61it/s]

🔠 2-gram 信息熵: 23.4549
📖 2-gram 总数: 136448071
⌛ 计算时间: 1569.1156 秒

计算 3-gram 信息熵...
分词进度: 100%|██████████| 183390233/183390233 [00:40<00:00, 4473428.90it/s]
统计 3-gram 频率: 100%|██████████| 136448070/136448070 [02:12<00:00, 1027475.33it/s]

🔢 3-gram 信息熵: 25.9021
📖 3-gram 总数: 136448070
⌛ 计算时间: 1971.8466 秒
```

Figure 13: The result of the program execution

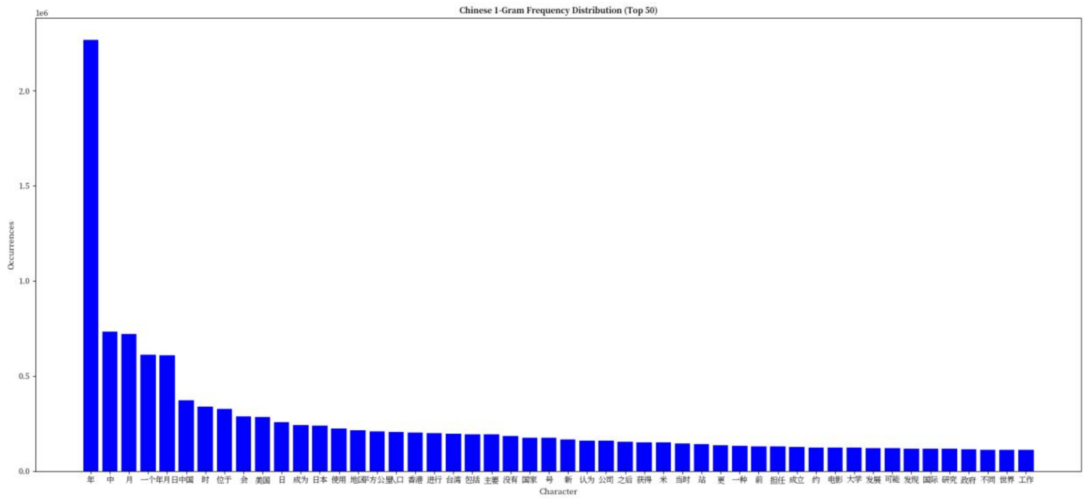


Figure 14: Chinese 1-Gram Frequency Distribution (Top 50)

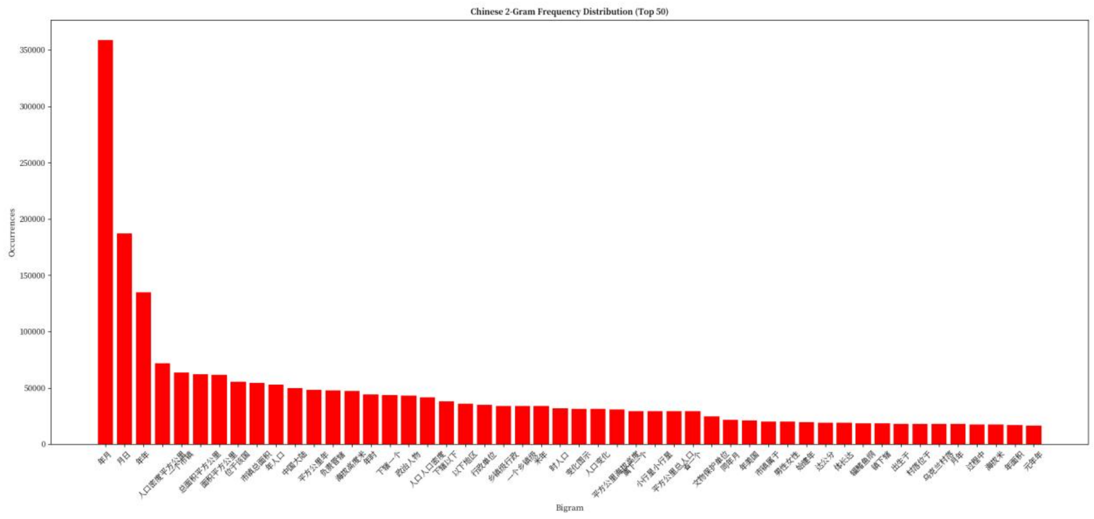


Figure 15: Chinese 2-Gram Frequency Distribution (Top 50)

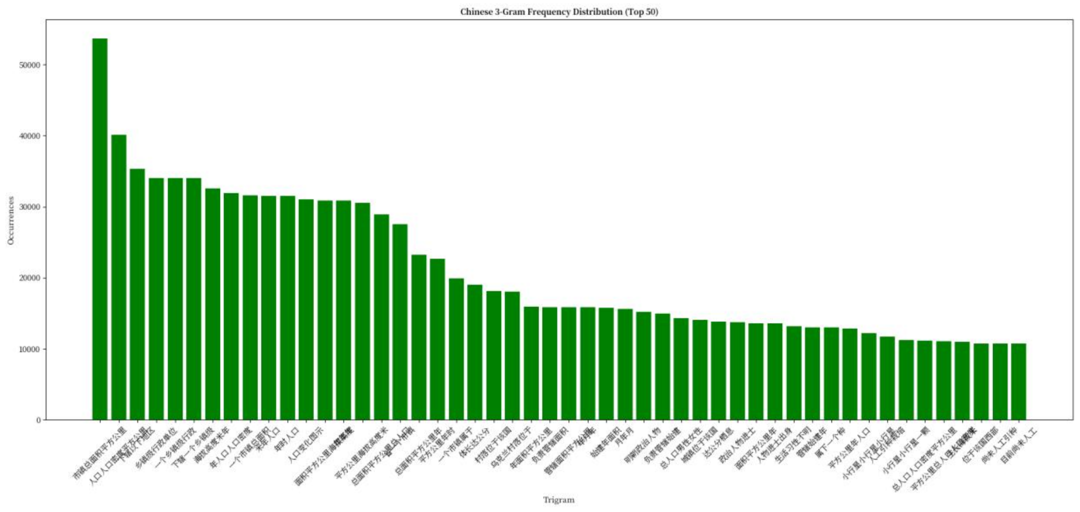


Figure 16: Chinese 3-Gram Frequency Distribution (Top 50)

Conclusions

This report explores the calculation methods of information entropy in Chinese and English, including letters, words, n-grams, and text preprocessing techniques such as stopword removal and word segmentation. It quantitatively compares information entropy at the character and word levels in both languages. The study finds that, in both Chinese and English, word-level entropy is higher than character-level entropy, reflecting the increased complexity of word combinations. The use of n-grams further increases entropy, while word segmentation significantly impacts entropy calculation in Chinese, and stopword removal generally reduces entropy. Additionally, Chinese exhibits higher character-level entropy due to its rich morphemic character set, whereas English demonstrates more uniform word-level entropy. The findings provide valuable insights for NLP tasks such as language modeling, text compression, and information retrieval. Furthermore, future research can extend to the study of information entropy differences across different languages, aiming to optimize preprocessing methods and improve the accuracy of entropy calculation.

References

[1] Zenchang Qin(2025), NLP1_NaturalLanguage.pdf.

[2] Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. 1992. An estimate of an upper bound for the entropy of English. Comput. Linguist. 18, 1 (March 1992), 31–40.

[3] ChatGPT:https://chatgpt.com

[4] Jieba Chinese Text Segmentation: https://github.com/fxsjy/jieba