Report of Deep Learning for Natural Langauge Processing

Xintong Zhang
zhangxintong0810@icloud.com

Abstract

This report's objective is to use the Chinese Wikipedia and the English Gutenberg as the corpus, and calculate the information entropy of Chinese and English respectively in the form of char and token, based on unigrams, bigrams and trigrams N-gram Model.

Introduction

Information Entropy

Information entropy is an extremely significant concept in information theory, which can be employed to measure the uncertainty or information quantity of a random variable. The concept of entropy was originally put forward by Claude Shannon in 1948 and is extensively utilized in domains such as communication, cryptography, data compression, signal processing, etc. In information theory, the definition of entropy is the average information content of a discrete random variable. It can be regarded as a measurement of the uncertainty of a random event. For a discrete random variable X, its information entropy is defined as

$$H(X)=-\sum ni\ P(xi)logP(xi)$$

Statistical Language Model

Suppose S represents a meaningful sentence, which is composed of a series of words or characters ω1, ω2, ω3, ..., ωn arranged in a specific sequence. Here, n denotes the number of words or characters that make up the sentence. Now we want to know the possibility of S appearing in the text, that is:

$$P(s) = P(\omega1 , \omega2 , \omega3 , . . . \omega n)$$

By applying the conditional probability formula, we can obtain:

$$P(\omega1 , \omega2 , \omega3 , . . . \omega n) = P(\omega1)P(\omega2 |\omega1). . . P(\omega n |\omega1 , \omega2 , . . . \omega n\text{-}1)$$

Let's take words as an example (the principle is the same as that of characters), and conduct theoretical analysis. Here, P(ω1) represents the probability of the first word ω1 appearing; P(ω2|ω1) is the probability of the second word appearing given that the first word has already occurred; and so on. P(ωn |ω1, ω2, ..., ωn-1) is the probability of the nth word appearing given that the first, second, third... (n-1)th words have already occurred. When calculating P(ω1), there is only one parameter; when calculating P(ω1|ω2), there are two parameters, and so on. In the case of long and numerous sentences, it is difficult to calculate. Therefore, Markov proposed an assumption: Assume that the probability of each word in the N-ary model is only related to the previous N-1 words. When N=2, it is a binary model; when N=3, it is a ternary model. An N-ary model means that the current word ωi depends on the previous N-1 words. The above N=1 means that it has no relation with the previous words; N=2 indicates that it is related to the previous word; N=3 indicates that it is related to the previous two words. The final results will vary when different models are selected.

When N=1, the mathematical expression of the unary model is as follows:

$$P(s)=P(w1)P(w2)P(w3)...P(wi)...P(wn)$$

When N = 2, the mathematical expression of the binary model is as follows.:

$$P(s)=P(\omega1)P(\omega2|\omega1)P(\omega3|\omega2)...P(\omega i|\omega i\text{-}1)...P(\omega n|\omega n\text{-}1)$$

When N = 3, the mathematical expression of the ternary model is as follows.:

$$P(s)=P(w1)P(w2|w1)P(w3|w1,w2)...P(wi|wi\text{-}2,wi\text{-}1)...P(wn|wn\text{-}2,wn\text{-}1)$$

The calculation of the information entropy of a computational language model

If the statistics are sufficient, the probabilities of the occurrence of words, words, bi-grams or trigrams are approximately equal to their frequencies of occurrence in the corpus.

Let's take words as an example (the principle is the same as that of characters), and conduct a theoretical analysis.

The formula for calculating the information entropy of a unigram model is as follows

$$H(X)=-\sum P(x)\log P(x)$$

Here, P(x) can be approximated as the frequency of each phrase appearing in the corpus.

The formula for calculating the information entropy of the binary model is as follows:

$$H(X|Y)=-\sum P(x,y)\log P(x|y)$$

Among them, the joint probability P(x,y) can be approximated as the frequency of each binary phrase appearing in the corpus, and the conditional probability P(x|y) can be approximated as the ratio of the frequency of each binary phrase appearing in the corpus to the frequency of binary phrases whose first word is the first word of the binary phrase x.

The formula for calculating the information entropy of the trigram model is as follows

$$H(X|Y,Z)=-\sum P(x,y,z)\log P(x|y,z)$$

Among them, the joint probability P(x,y,z) can be approximated as being equal to the frequency of each trigram appearing in the corpus, and the conditional probability P(x|y,z) can be approximated as being equal to the ratio of the frequency of each trigram appearing in the corpus to the frequency of trigrams whose first two words are the same as those of the trigram x,y,z.

Methodology

The following is an introduction to the programming design and its functions.

Step1:Loading and preprocessing of data

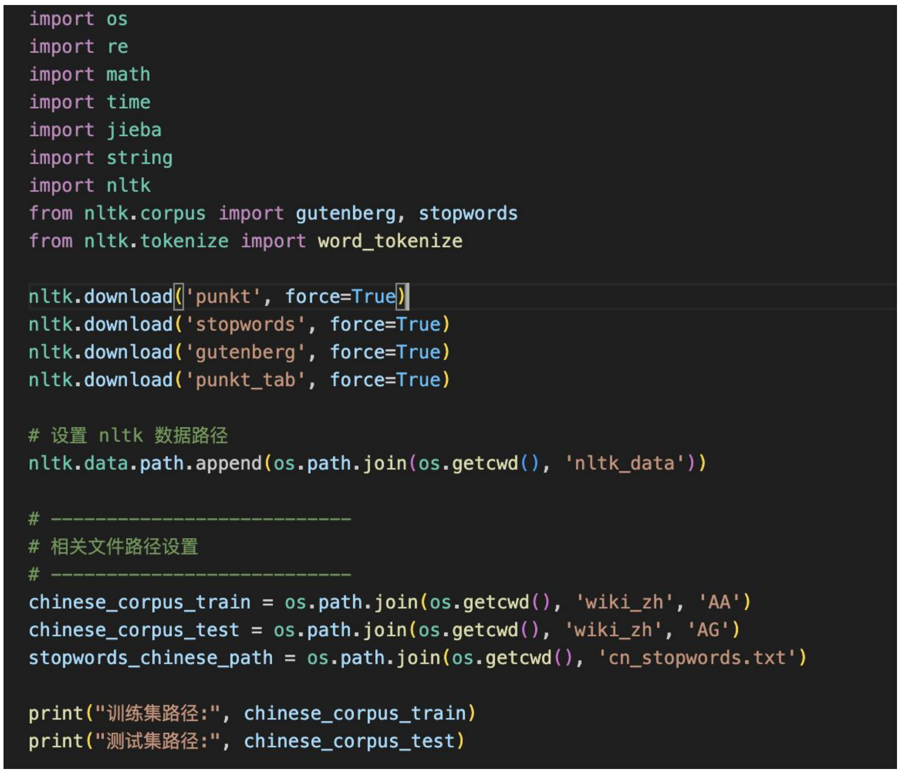
(1) Environment configuration:

Import necessary libraries (os, re, math, time, jieba, nltk, etc.).Download relevant resources of nltk and point the path of nltk data to the nltk_data folder in the current working directory.

(2) Setting of relevant paths

Set chinese_corpus_train and chinese_corpus_test to store the paths of Chinese corpora for training set and test set respectively.Set stopwords_chinese_path to store the path of the Chinese stopwordtable.

(3)

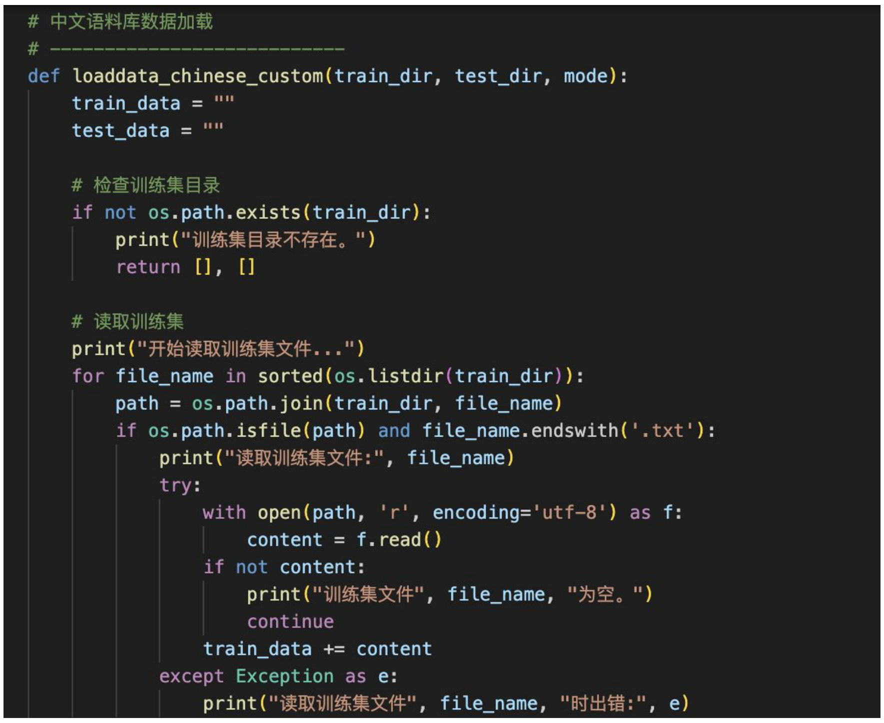


```python
import os
import re
import math
import time
import jieba
import string
import nltk
from nltk.corpus import gutenberg, stopwords
from nltk.tokenize import word_tokenize

nltk.download('punkt', force=True)
nltk.download('stopwords', force=True)
nltk.download('gutenberg', force=True)
nltk.download('punkt_tab', force=True)

# 设置 nltk 数据路径
nltk.data.path.append(os.path.join(os.getcwd(), 'nltk_data'))

# ---------------------------
# 相关文件路径设置
# ---------------------------
chinese_corpus_train = os.path.join(os.getcwd(), 'wiki_zh', 'AA')
chinese_corpus_test = os.path.join(os.getcwd(), 'wiki_zh', 'AG')
stopwords_chinese_path = os.path.join(os.getcwd(), 'cn_stopwords.txt')

print("训练集路径:", chinese_corpus_train)
print("测试集路径:", chinese_corpus_test)
```

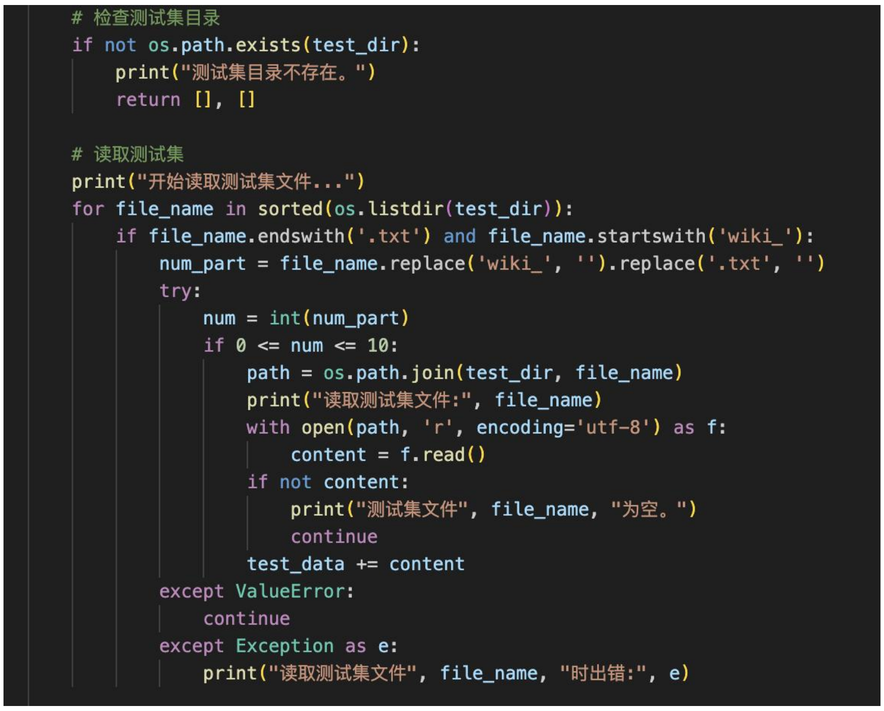
Step 2: Loading and preprocessing of Chinese and English data

For Chinese data, read the txt files of the training set and test set, merge all contents, retain only Chinese characters through the regular expression r'[^\u4e00-\u9fff]', use jieba.cut() to tokenize, load the stop word list, and filter out the stop words.

For English data, use the nltk.corpus.gutenberg corpus, remove a specific text (austen-emma.txt).Tokenization mode: Tokenization + Remove punctuation + Filter stop words. Character mode: Split by characters + Retain only alphabetic characters.

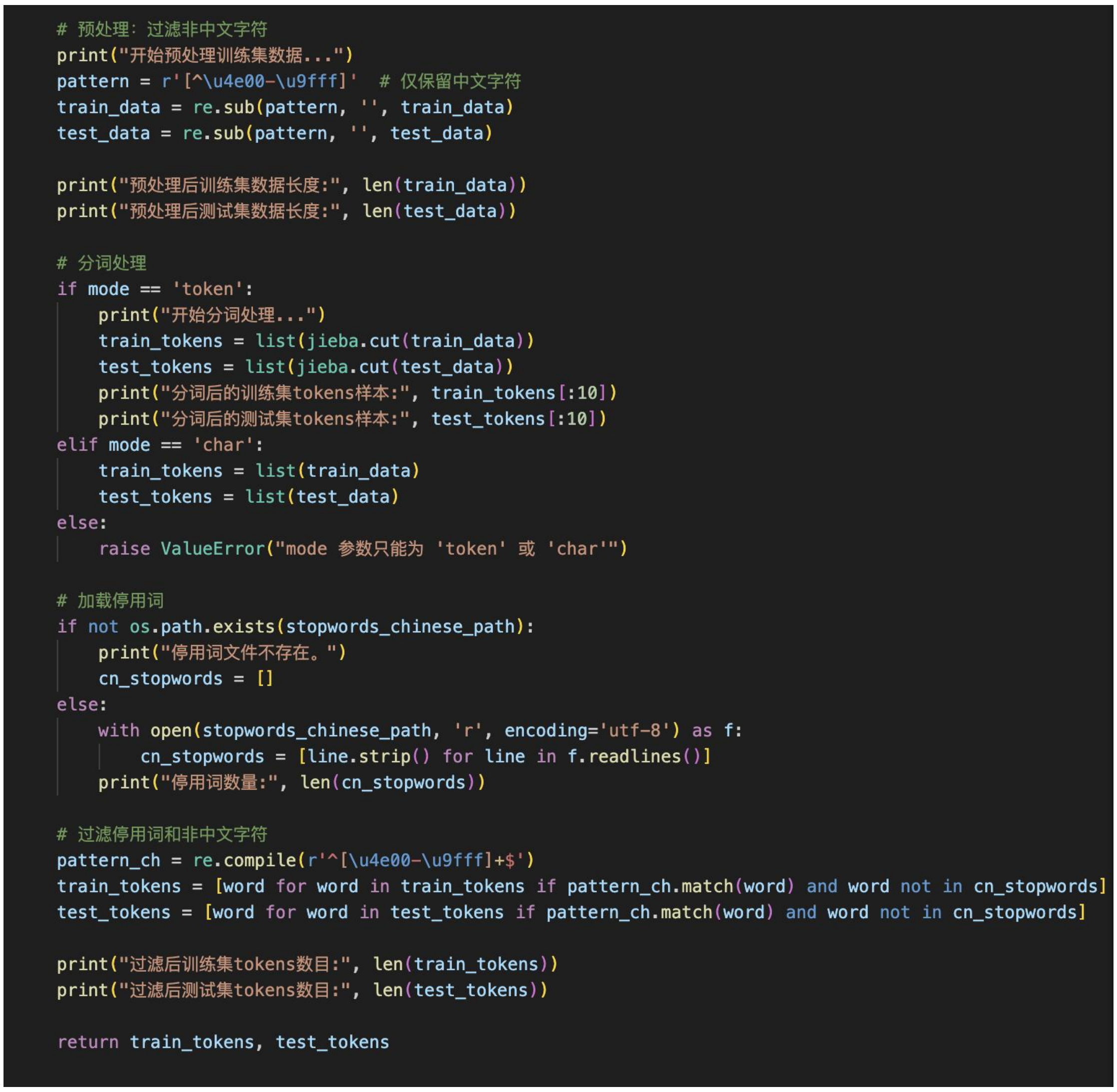


```python
# 中文语料库数据加载
# ---------------------------
def loaddata_chinese_custom(train_dir, test_dir, mode):
    train_data = ""
    test_data = ""

    # 检查训练集目录
    if not os.path.exists(train_dir):
        print("训练集目录不存在。")
        return [], []

    # 读取训练集
    print("开始读取训练集文件...")
    for file_name in sorted(os.listdir(train_dir)):
        path = os.path.join(train_dir, file_name)
        if os.path.isfile(path) and file_name.endswith('.txt'):
            print("读取训练集文件:", file_name)
            try:
                with open(path, 'r', encoding='utf-8') as f:
                    content = f.read()
                if not content:
                    print("训练集文件", file_name, "为空。")
                    continue
                train_data += content
            except Exception as e:
                print("读取训练集文件", file_name, "时出错:", e)
```



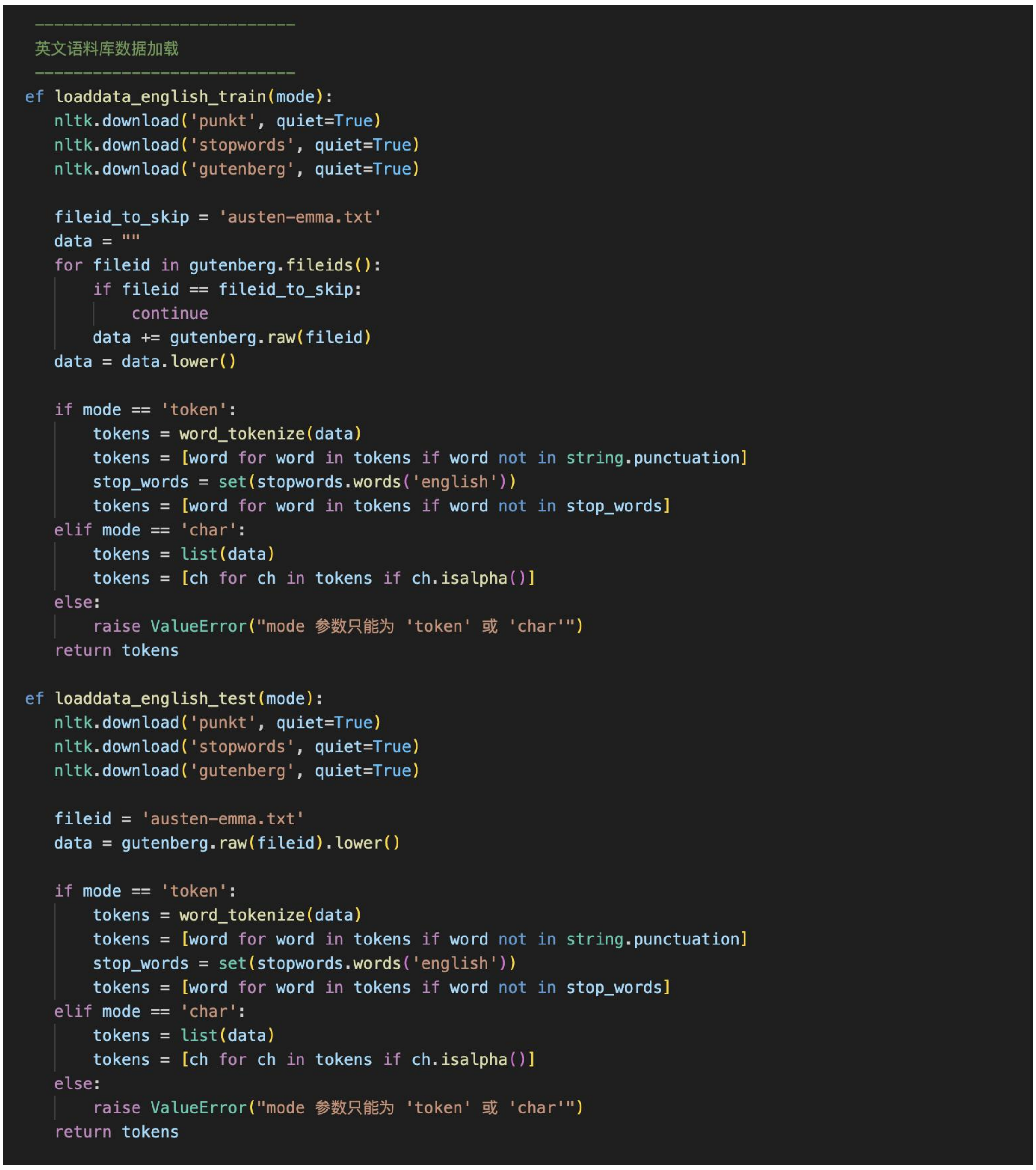
```python
    # 检查测试集目录
    if not os.path.exists(test_dir):
        print("测试集目录不存在。")
        return [], []

    # 读取测试集
    print("开始读取测试集文件...")
    for file_name in sorted(os.listdir(test_dir)):
        if file_name.endswith('.txt') and file_name.startswith('wiki_'):
            num_part = file_name.replace('wiki_', '').replace('.txt', '')
            try:
                num = int(num_part)
                if 0 <= num <= 10:
                    path = os.path.join(test_dir, file_name)
                    print("读取测试集文件:", file_name)
                    with open(path, 'r', encoding='utf-8') as f:
                        content = f.read()
                    if not content:
                        print("测试集文件", file_name, "为空。")
                        continue
                    test_data += content
            except ValueError:
                continue
            except Exception as e:
                print("读取测试集文件", file_name, "时出错:", e)
```

Preprocessing of Chinese data:



```python
# 预处理：过滤非中文字符
print("开始预处理训练集数据...")
pattern = r'[^\u4e00-\u9fff]'  # 仅保留中文字符
train_data = re.sub(pattern, '', train_data)
test_data = re.sub(pattern, '', test_data)

print("预处理后训练集数据长度:", len(train_data))
print("预处理后测试集数据长度:", len(test_data))

# 分词处理
if mode == 'token':
    print("开始分词处理...")
    train_tokens = list(jieba.cut(train_data))
    test_tokens = list(jieba.cut(test_data))
    print("分词后的训练集tokens样本:", train_tokens[:10])
    print("分词后的测试集tokens样本:", test_tokens[:10])
elif mode == 'char':
    train_tokens = list(train_data)
    test_tokens = list(test_data)
else:
    raise ValueError("mode 参数只能为 'token' 或 'char'")

# 加载停用词
if not os.path.exists(stopwords_chinese_path):
    print("停用词文件不存在。")
    cn_stopwords = []
else:
    with open(stopwords_chinese_path, 'r', encoding='utf-8') as f:
        cn_stopwords = [line.strip() for line in f.readlines()]
    print("停用词数量:", len(cn_stopwords))

# 过滤停用词和非中文字符
pattern_ch = re.compile(r'^[\u4e00-\u9fff]+$')
train_tokens = [word for word in train_tokens if pattern_ch.match(word) and word not in cn_stopwords]
test_tokens = [word for word in test_tokens if pattern_ch.match(word) and word not in cn_stopwords]

print("过滤后训练集tokens数目:", len(train_tokens))
print("过滤后测试集tokens数目:", len(test_tokens))

return train_tokens, test_tokens
```

Loading and preprocessing of English data:



```
 ---------------------------
 英文语料库数据加载
 ---------------------------
ef loaddata_english_train(mode):
   nltk.download('punkt', quiet=True)
   nltk.download('stopwords', quiet=True)
   nltk.download('gutenberg', quiet=True)

   fileid_to_skip = 'austen-emma.txt'
   data = ""
   for fileid in gutenberg.fileids():
       if fileid == fileid_to_skip:
           continue
       data += gutenberg.raw(fileid)
   data = data.lower()

   if mode == 'token':
       tokens = word_tokenize(data)
       tokens = [word for word in tokens if word not in string.punctuation]
       stop_words = set(stopwords.words('english'))
       tokens = [word for word in tokens if word not in stop_words]
   elif mode == 'char':
       tokens = list(data)
       tokens = [ch for ch in tokens if ch.isalpha()]
   else:
       raise ValueError("mode 参数只能为 'token' 或 'char'")
   return tokens

ef loaddata_english_test(mode):
   nltk.download('punkt', quiet=True)
   nltk.download('stopwords', quiet=True)
   nltk.download('gutenberg', quiet=True)

   fileid = 'austen-emma.txt'
   data = gutenberg.raw(fileid).lower()

   if mode == 'token':
       tokens = word_tokenize(data)
       tokens = [word for word in tokens if word not in string.punctuation]
       stop_words = set(stopwords.words('english'))
       tokens = [word for word in tokens if word not in stop_words]
   elif mode == 'char':
       tokens = list(data)
       tokens = [ch for ch in tokens if ch.isalpha()]
   else:
       raise ValueError("mode 参数只能为 'token' 或 'char'")
   return tokens
```

Step3:Calculate the information entropy of the first-, second- and third-order models based on the segmentation of characters, words and phrases.

(1) Frequency Statistics

get_tf(tf_dic, words): Computes unigram word frequency.

get_bigram_tf(tf_dic, words): Computes bigram word frequency.

get_trigram_tf(tf_dic, words): Computes trigram word frequency.

(2) Entropy Calculation

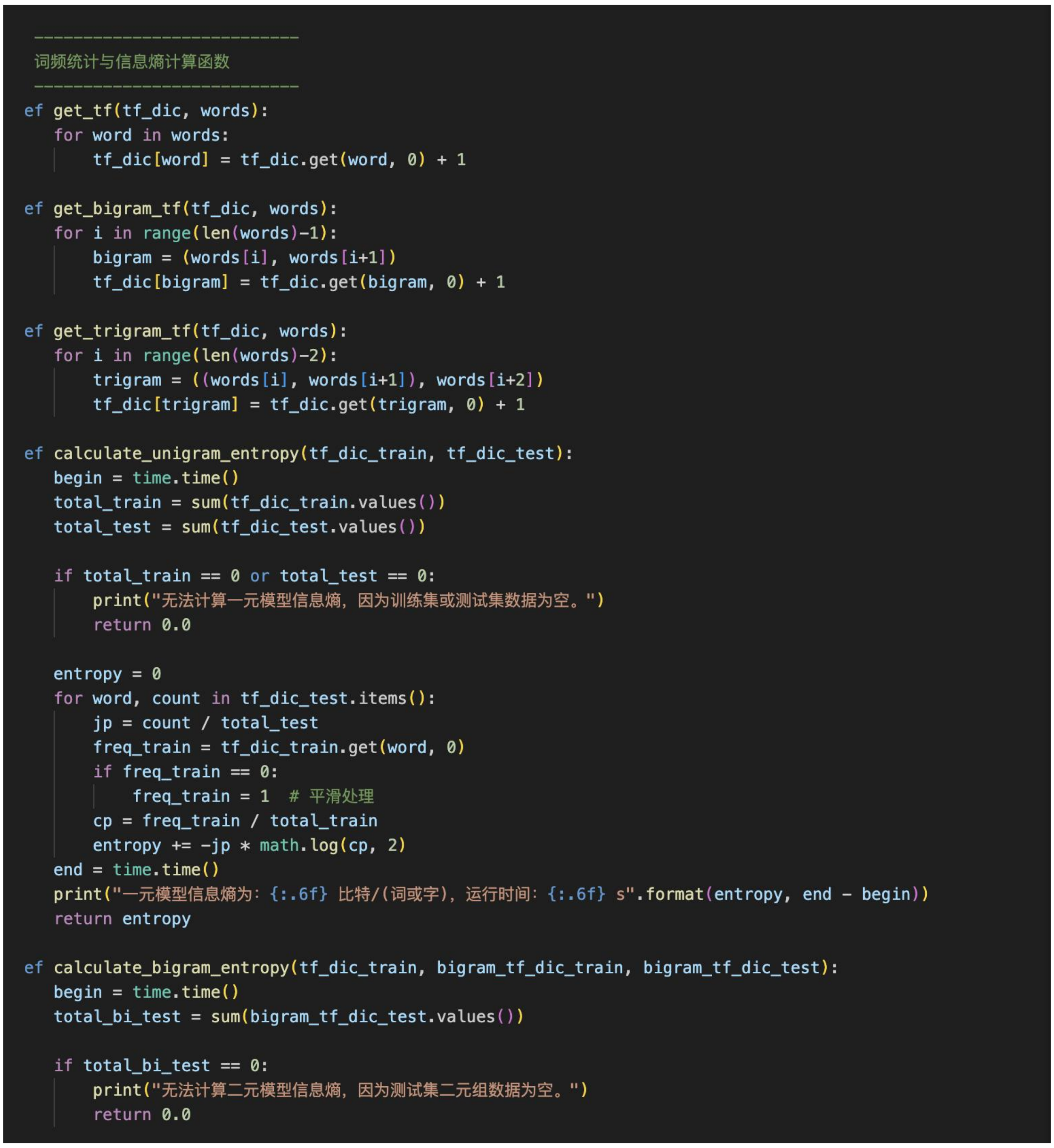
calculate_unigram_entropy(tf_dic_train, tf_dic_test):

Computes the probability of each word in the test set appearing in the training set.

Uses the entropy formula:

H(X)=−∑P(x)logP(x)H(X) = - \sum P(x) \log P(x)H(X)=−∑P(x)logP(x)

Outputs the unigram model entropy.



```
 ----------------------------
 词频统计与信息熵计算函数
 ----------------------------
ef get_tf(tf_dic, words):
   for word in words:
       tf_dic[word] = tf_dic.get(word, 0) + 1

ef get_bigram_tf(tf_dic, words):
   for i in range(len(words)-1):
       bigram = (words[i], words[i+1])
       tf_dic[bigram] = tf_dic.get(bigram, 0) + 1

ef get_trigram_tf(tf_dic, words):
   for i in range(len(words)-2):
       trigram = ((words[i], words[i+1]), words[i+2])
       tf_dic[trigram] = tf_dic.get(trigram, 0) + 1

ef calculate_unigram_entropy(tf_dic_train, tf_dic_test):
   begin = time.time()
   total_train = sum(tf_dic_train.values())
   total_test = sum(tf_dic_test.values())

   if total_train == 0 or total_test == 0:
       print("无法计算一元模型信息熵，因为训练集或测试集数据为空。")
       return 0.0

   entropy = 0
   for word, count in tf_dic_test.items():
       jp = count / total_test
       freq_train = tf_dic_train.get(word, 0)
       if freq_train == 0:
           freq_train = 1  # 平滑处理
       cp = freq_train / total_train
       entropy += -jp * math.log(cp, 2)
   end = time.time()
   print("一元模型信息熵为：{:.6f} 比特/(词或字)，运行时间：{:.6f} s".format(entropy, end - begin))
   return entropy

ef calculate_bigram_entropy(tf_dic_train, bigram_tf_dic_train, bigram_tf_dic_test):
   begin = time.time()
   total_bi_test = sum(bigram_tf_dic_test.values())

   if total_bi_test == 0:
       print("无法计算二元模型信息熵，因为测试集二元组数据为空。")
       return 0.0
```

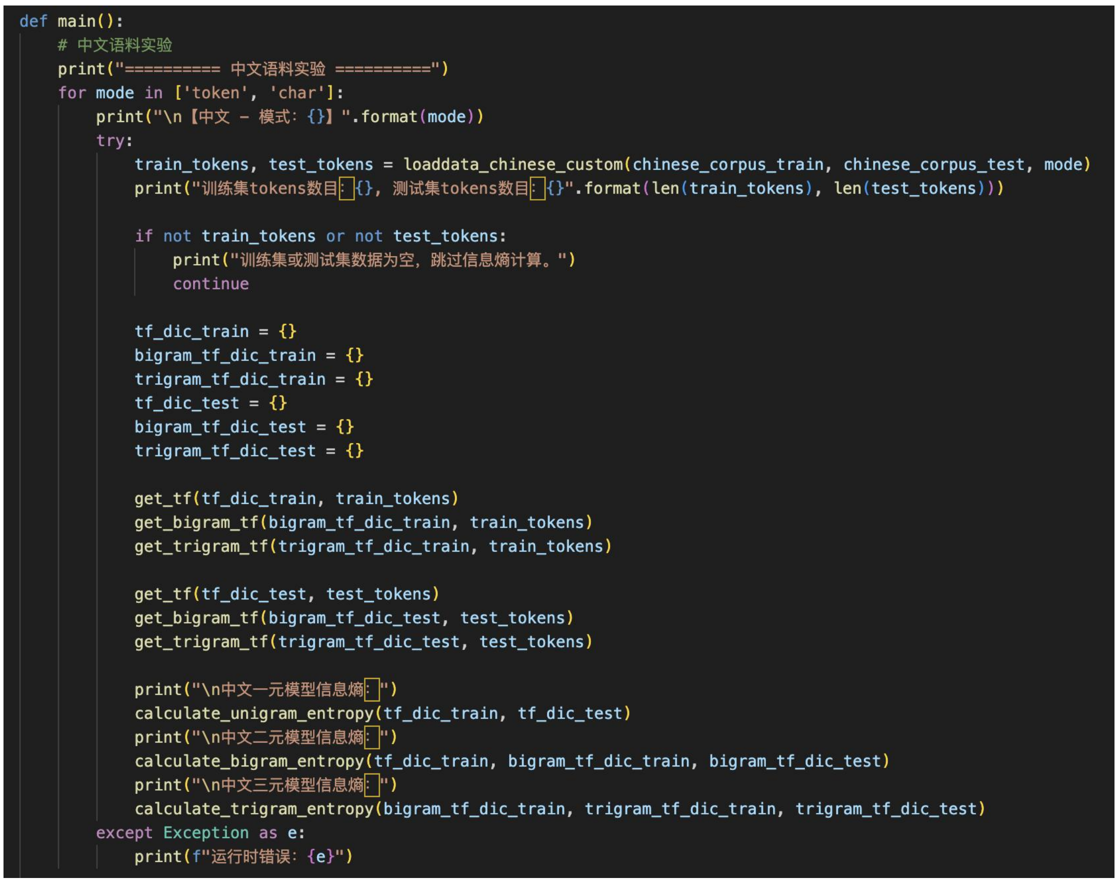
Step4:Main Function Execution

Sequential execution steps:

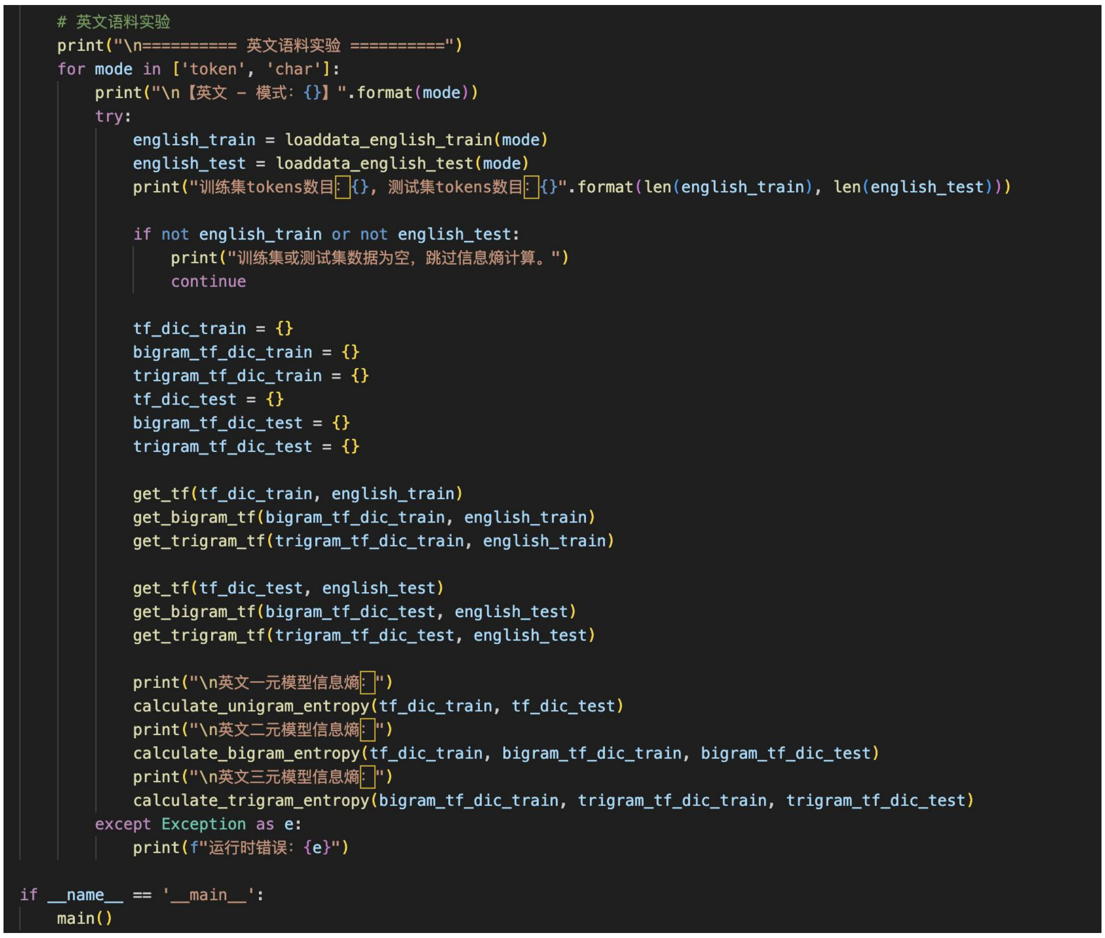
Compute entropy for both token and char modes in Chinese.Compute entropy for both token and char modes in English.

Execution Process:

Load and preprocess data.Compute N-gram word frequencies.Compute entropy for different N-gram models.Output final entropy results.



```python
def main():
    # 中文语料实验
    print("========== 中文语料实验 ==========")
    for mode in ['token', 'char']:
        print("\n【中文 - 模式：{}】".format(mode))
        try:
            train_tokens, test_tokens = loaddata_chinese_custom(chinese_corpus_train, chinese_corpus_test, mode)
            print("训练集tokens数目：{}，测试集tokens数目：{}".format(len(train_tokens), len(test_tokens)))

            if not train_tokens or not test_tokens:
                print("训练集或测试集数据为空，跳过信息熵计算。")
                continue

            tf_dic_train = {}
            bigram_tf_dic_train = {}
            trigram_tf_dic_train = {}
            tf_dic_test = {}
            bigram_tf_dic_test = {}
            trigram_tf_dic_test = {}

            get_tf(tf_dic_train, train_tokens)
            get_bigram_tf(bigram_tf_dic_train, train_tokens)
            get_trigram_tf(trigram_tf_dic_train, train_tokens)

            get_tf(tf_dic_test, test_tokens)
            get_bigram_tf(bigram_tf_dic_test, test_tokens)
            get_trigram_tf(trigram_tf_dic_test, test_tokens)

            print("\n中文一元模型信息熵：")
            calculate_unigram_entropy(tf_dic_train, tf_dic_test)
            print("\n中文二元模型信息熵：")
            calculate_bigram_entropy(tf_dic_train, bigram_tf_dic_train, bigram_tf_dic_test)
            print("\n中文三元模型信息熵：")
            calculate_trigram_entropy(bigram_tf_dic_train, trigram_tf_dic_train, trigram_tf_dic_test)
        except Exception as e:
            print(f"运行时错误：{e}")
```



```python
    # 英文语料实验
    print("\n========== 英文语料实验 ==========")
    for mode in ['token', 'char']:
        print("\n【英文 - 模式：{}】".format(mode))
        try:
            english_train = loaddata_english_train(mode)
            english_test = loaddata_english_test(mode)
            print("训练集tokens数目：{}，测试集tokens数目：{}".format(len(english_train), len(english_test)))

            if not english_train or not english_test:
                print("训练集或测试集数据为空，跳过信息熵计算。")
                continue

            tf_dic_train = {}
            bigram_tf_dic_train = {}
            trigram_tf_dic_train = {}
            tf_dic_test = {}
            bigram_tf_dic_test = {}
            trigram_tf_dic_test = {}

            get_tf(tf_dic_train, english_train)
            get_bigram_tf(bigram_tf_dic_train, english_train)
            get_trigram_tf(trigram_tf_dic_train, english_train)

            get_tf(tf_dic_test, english_test)
            get_bigram_tf(bigram_tf_dic_test, english_test)
            get_trigram_tf(trigram_tf_dic_test, english_test)

            print("\n英文一元模型信息熵：")
            calculate_unigram_entropy(tf_dic_train, tf_dic_test)
            print("\n英文二元模型信息熵：")
            calculate_bigram_entropy(tf_dic_train, bigram_tf_dic_train, bigram_tf_dic_test)
            print("\n英文三元模型信息熵：")
            calculate_trigram_entropy(bigram_tf_dic_train, trigram_tf_dic_train, trigram_tf_dic_test)
        except Exception as e:
            print(f"运行时错误：{e}")

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

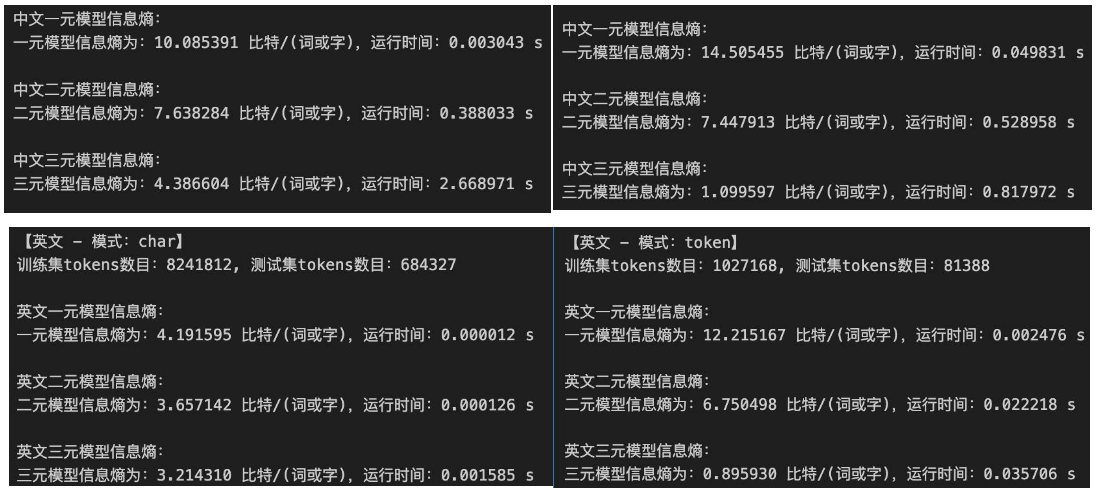
Experimental Studies

The experimental results are presented in the following table.

Table 1: The result of information entropy calculation

Information Entropy (bit/char\token)	**Chinese Char**	**Chinese Token**	**English Char**	**English Token**
Unigram	10.085391	14.505455	4.191595	12.215167
Bigram	7.638284	7.447913	3.657142	6.750498
Trigram	4.386604	1.099597	3.214310	0.895030

The following is the actual data presentation.



```
中文一元模型信息熵:
一元模型信息熵为: 10.085391 比特/(词或字), 运行时间: 0.003043 s

中文二元模型信息熵:
二元模型信息熵为: 7.638284 比特/(词或字), 运行时间: 0.388033 s

中文三元模型信息熵:
三元模型信息熵为: 4.386604 比特/(词或字), 运行时间: 2.668971 s
```

```
中文一元模型信息熵:
一元模型信息熵为: 14.505455 比特/(词或字), 运行时间: 0.049831 s

中文二元模型信息熵:
二元模型信息熵为: 7.447913 比特/(词或字), 运行时间: 0.528958 s

中文三元模型信息熵:
三元模型信息熵为: 1.099597 比特/(词或字), 运行时间: 0.817972 s
```

```
【英文 - 模式: char】
训练集tokens数目: 8241812, 测试集tokens数目: 684327

英文一元模型信息熵:
一元模型信息熵为: 4.191595 比特/(词或字), 运行时间: 0.000012 s

英文二元模型信息熵:
二元模型信息熵为: 3.657142 比特/(词或字), 运行时间: 0.000126 s

英文三元模型信息熵:
三元模型信息熵为: 3.214310 比特/(词或字), 运行时间: 0.001585 s
```

```
【英文 - 模式: token】
训练集tokens数目: 1027168, 测试集tokens数目: 81388

英文一元模型信息熵:
一元模型信息熵为: 12.215167 比特/(词或字), 运行时间: 0.002476 s

英文二元模型信息熵:
二元模型信息熵为: 6.750498 比特/(词或字), 运行时间: 0.022218 s

英文三元模型信息熵:
三元模型信息熵为: 0.895930 比特/(词或字), 运行时间: 0.035706 s
```

Figure 1： Display of running results

Conclusions

By comparing the information entropy of characters and monomorphemic words, it can be found that the information entropy of monomorphemic words is greater than that of characters based on the character model, and it is lower than that of the monomorphemic word model. There are several main reasons for this:

The types of characters are fewer than those of words. There are several thousand types of Chinese characters, but the commonly used ones are less than a thousand. In contrast, the vocabulary of Chinese is very large, reaching tens of thousands or more. Therefore, the information entropy based on the monomorphemic word model is larger.

Characters have more definite meanings in different contexts. Compared with words, characters can more accurately represent a certain concept. Because in different contexts, the same word may have different meanings, while the same character generally has only one meaning in different contexts.

The character model is easier to recognize new words. If a new word is encountered when using a language model based on words, the model will be unable to handle it. However, the language model based on the character model can represent new words by combining existing characters, thus being more flexible and accurate.

The information entropy of the binary model is lower than that of the monomorphemic model because in the binary model, the occurrence probability of each character is related to the previous character. Therefore, the binary model can better capture the relationship between characters in language modeling, reducing the redundancy of adjacent characters and thus lowering the information entropy. In the monomorphemic model, the occurrence probability of each character is only related to itself, unable to capture the relationship between characters, so the redundancy of adjacent characters is larger, resulting in a higher information entropy. The reason why the information entropy of the trigram model is lower than that of the binary model is the same.

Through additional experiments, it was found that after removing stop words, the information entropy of all monomorphemic models has increased, while the information entropy of the binary and trigram models has decreased. Because after removing stop words, redundant information in the text is eliminated, and the remaining effective information is more concentrated. In the binary and trigram models, each element is statistically based on two or three consecutive words, so compared with the monomorphemic model, they can better reflect the semantic relationship and context information between words. Therefore, after removing stop words, the binary and trigram models can more accurately reflect the relevance and context of words in the text, resulting in lower information entropy.

References

[1] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. 1992. An Estimate of an Upper Bound for the Entropy of English.