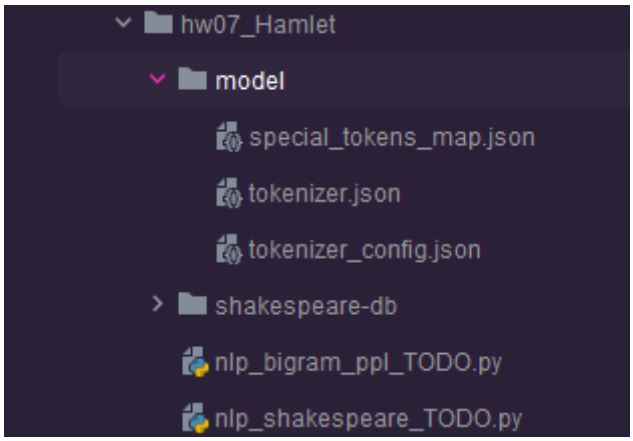
HW-7-陈宸

Q1.实现 Bi-Gram PPL 计算

下载tokenizer等相关文件至model目录下



计算bi-grams ppl相关代码如下

```
query_unigrams_count = Counter([w[0] for w in list(ngrams(query_token, 1))])
bigram_count = Counter(bigrams)

ppl_list = []
mult_ppl = 1

ppl_list.append(1 / (unigrams[query_bigram[0][0]] / len(token)))
for bg in query_bigram:
    ppl_list.append(1 / (bigram_count[bg] / unigrams[bg[0]]))

for i in range(len(ppl_list)):
    mult_ppl *= ppl_list[i]

print(mult_ppl ** (1 / len(ppl_list)))
```

output:

bi-gram ppl of text1: 2.5694703142468787

bi-gram ppl of text2: 2.78651802273122

解释：Text2中的某些双词短语在语料库出现的频率高，但Text1的双词短语在训练语料库中的出现频率也较高且分布均匀，整体上使得 PPL 值更低。

NLTK分词原理

NLTK利用多种分词器综合对不同特征的文本进行分词，简单介绍几个常用的分词器及其工作原理。

Punkt Tokenizer 该分词器基于统计模型，通过训练大量文本数据来识别句子边界和单词边界，使用了最大概率估计和贝叶斯估计来进行判断；Treebank Tokenizer基于预定义的规则，通过一系列正则表达式来识别单词和标点符号。它适用于英语和其他一些语言；WordPunct Tokenizer该分词器将单词和标点符号视为两类不同的标记，通过词典和规则来进行分词。Regexp Tokenizer该分词器使用正则表达式来定义分词规则，用户可以自定义分词规则来适应不同的语言和应用场景。

reference:https://www.nltk.org/book/ch03.html#tokenization

Q2.Hamlet's Question

计算ppl相关代码如下：

```
question_bigrams = Counter(list(ngrams(the_question_tokens, 2)))
N = len(question_bigrams)+1
ppl_val = np.log(unigrams[the_question_tokens[0]] / len(sent_all))

for word_pair in question_bigrams:
    prob = bigrams[word_pair] / unigrams[word_pair[0]]
    ppl_val += np.log(prob)

ppl_val = np.exp(-1 / N * ppl_val)
print("Perplexity for the given string: {}".format(ppl_val))
```

output:

```
Perplexity for the given string: 86.05482426311308
```