程序测试

计卓 02-20195633-李燕琴，NLP 第一次作业

```
开始初始化
初始化成功
开始测试
1. 可输入 exit 退出测试程序
2. 平滑方法，1:加一平滑，2:katz平滑

请输入测试句子：今天是个好日子
请输入平滑方法：1
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Maxpicca\AppData\Local\Temp\jieba.c
测试句子：今天是个好日子
分词结果：['BOS', '今天', '是', '个', '好日子', 'EOS']
成句概率：2.6711044590310412e-20
请输入测试句子：Loading model cost 1.032 seconds.
Prefix dict has been built successfully.
今天是个好日子
请输入平滑方法：2
测试句子：今天是个好日子
分词结果：['BOS', '今天', '是', '个', '好日子', 'EOS']
成句概率：2.6711044590310412e-20
请输入测试句子：Loading model cost 1.032 seconds.
Prefix dict has been built successfully.
今天是个好日子
请输入平滑方法：2
测试句子：今天是个好日子
分词结果：['BOS', '今天', '是', '个', '好日子', 'EOS']
生词：[]
成句概率：0.9837011759183024
请输入测试句子：香港回归祖国怀抱
请输入平滑方法：1
测试句子：香港回归祖国怀抱
分词结果：['BOS', '香港', '回归祖国', '怀抱', 'EOS']
成句概率：1.4359145533240236e-18
请输入测试句子：香港回归祖国怀抱
请输入平滑方法：2
测试句子：香港回归祖国怀抱
分词结果：['BOS', '香港', '回归祖国', '怀抱', 'EOS']
成句概率：1.4359145533240236e-18
请输入测试句子：香港回归祖国怀抱
请输入平滑方法：2
测试句子：香港回归祖国怀抱
分词结果：['BOS', '香港', '回归祖国', '怀抱', 'EOS']
生词：['回归祖国']
成句概率：9.785656236048336e-11
请输入测试句子：一定要克服困难呀
请输入平滑方法：1
测试句子：一定要克服困难呀
分词结果：['BOS', '一定', '要', '克服困难', '呀', 'EOS']
成句概率：1.7478553683929957e-22
请输入测试句子：一定要克服困难呀
请输入平滑方法：2
测试句子：一定要克服困难呀
分词结果：['BOS', '一定', '要', '克服困难', '呀', 'EOS']
成句概率：1.7478553683929957e-22
请输入测试句子：一定要克服困难呀
请输入平滑方法：2
测试句子：一定要克服困难呀
分词结果：['BOS', '一定', '要', '克服困难', '呀', 'EOS']
生词：['克服困难']
成句概率：9.795859724650118e-11
请输入测试句子：
```

程序源码

LangModel_v2.1.py

```
# -*- codeing = utf-8 -*-
# @Author: Maxpicca
# @Description: 语言模型 LangModel_v1.0，加一平滑，katz 函数实现

import jieba
import pandas as pd
import math
import time
from collections import Counter
import os
import sys
from utils import get_word_freq,get_unigram_list,my_read_dict

class LangModel():
    '''
    语言模型 V2.1
    '''
    def __init__(self,unigram_path="",bigram_path="",corpus_path="./训练语料
utf-8.txt",islog=False) -> None:
        '''
        Language Model 语言模型训练初始化，加载相应训练结果\n
        :param unigram_path:
        :param bigram_path: unigra
        :param corpus_path: 语料库路径 utf-8
        :param islog: 是否打印日志
        '''
        if islog:
            print("开始初始化")
        if not (os.path.exists(unigram_path) and os.path.exists(bigram_path)):
            get_word_freq(corpus_path,unigram_path,bigram_path)

        self.unigram_counter = Counter(my_read_dict(unigram_path))
        self.bigram_counter = Counter(my_read_dict(bigram_path))
        self.bigram_cnt_counter = Counter(list(self.bigram_counter.values()))

        self.NoBosEos_total_cnt = sum(self.unigram_counter.values()) -
self.unigram_counter['BOS'] - self.unigram_counter['EOS']
        self.NoBos_total_cnt = sum(self.unigram_counter.values()) -
self.unigram_counter['BOS']

        self.gt1max = 0
```

```
        self.gt2max = 10
        self.A = (self.gt2max+1)*self.bigram_cnt_counter[self.gt2max+1] /
self.bigram_cnt_counter[1]
        self.no_word_p = 1e-10  # 陌生词汇概率 → 语料库越大，陌生词汇概率越小
        self.no_word_list = []  # 生词序列

        if islog:
            print("初始化成功")

    def add_one_smmoth(self,sent,islog=False):
        '''
        加一平滑：\n
        - 优点：简单易实现，训练语料库时间快
        - 缺点：容易受到语料库中词汇量大小的影响
        :param sent: 测试句子
        :param islog:
        :return:
        '''
        test_list = get_unigram_list(sent)
        if islog:
            print(f"测试句子：{sent}")
            print(f"分词结果：{test_list}")
        p = 1
        V = len(self.unigram_counter) - 2  # 除去BOS和EOS
        for i in range(1,len(test_list)):
            cb = 0
            cu = 0
            word = test_list[i-1]
            # if islog:
            #     print(word)
            bigram = test_list[i-1]+'@'+test_list[i]
            cb = self.bigram_counter[bigram] #  if bigram in self.bigram_counter else
0
            cu = self.unigram_counter[word]     # if word in self.unigram_counter else
0
            p*=(cb+1)/(cu+V)
        if islog:
            print(f"成句概率：{p}")
        return p

    def katz_smooth(self,sent,islog=False):
        '''
        katz平滑，结合Good Touring假设频次进行计算\n
        参考实现：
```

```
    - https://zhuanlan.zhihu.com/p/100256789
    - https://github.com/Neesky/Bigram/blob/master/katz.py
    :param sent: 测试句子
    :param islog:
    :return:
    '''
    if sent=="":
        return 0
    test_list = get_unigram_list(sent)
    if islog:
        print(f"测试句子：{sent}")
        print(f"分词结果：{test_list}")

    p=1
    self.no_word_list = []
    word1 = test_list[0]
    for i in range(1,len(test_list)):
        word2 = test_list[i]
        p*=self.katz_pred(word1,word2)
    if islog:
        print(f"生词：{self.no_word_list}")
        print(f"成句概率：{p}")
    self.no_word_list = []
    return p

def katz_pred(self,word1,word2):
    '''
    出现word1后，word1和word2同时出现的概率\n
    :param word1:
    :param word2:
    :return:
    '''
    cnt_word1 = self.unigram_counter[word1]
    cnt_word2 = self.unigram_counter[word2]
    cnt_word12 = self.bigram_counter[word1+'@'+word2]
    if cnt_word1!=0 and cnt_word2!=0:
        if cnt_word12==0:
            bow1 = self.cal_bow1(word1)
            p2 = self.unigram_counter[word2]/self.NoBosEos_total_cnt
            p = bow1*p2
        else:
            p = self.cal_faz(cnt_word1,cnt_word12)
        return math.pow(10,p)
    else:
```

```
            if cnt_word1==0:
                self.no_word_list.append(word1)
            if cnt_word2==0:
                self.no_word_list.append(word2)
            return self.no_word_p

    def cal_faz(self,cnt_word1,cnt_word12):
        '''
        计算已知二元词的概率\n
        :param cnt_word1:
        :param cnt_word12:
        :return:
        '''
        max_estimation = cnt_word12 / cnt_word1
        if cnt_word12 >= self.gt2max:
            return max_estimation
        else:
            # 折扣率
            d = (self.new_cnt(cnt_word12)/cnt_word12 - self.A)/(1-self.A)
            return d*max_estimation

    def new_cnt(self,cnt_word12):
        '''
        计算 Good Touring 的假设频次\n
        :param cnt_word12:
        :return:
        '''
        cnt_next = cnt_word12 + 1
        while self.bigram_cnt_counter[cnt_next]==0:
            cnt_next += 1
            if cnt_next > 5*cnt_word12:
                sys.exit(-1)
        return cnt_next*self.bigram_cnt_counter[cnt_next] /
self.bigram_cnt_counter[cnt_word12]

    def cal_bow1(self,word1):
        '''
        计算单元平滑系数 bow1\n
        :param word1:
        :return:
        '''
        sum_f1x = 0
        sum_fx = 0
        cnt_word1 = self.unigram_counter[word1]
```

```
        for wordx in list(self.unigram_counter.keys()):
            cnt_word1x = self.bigram_counter[word1+'@'+wordx]
            if cnt_word1x>0:
                sum_f1x += self.cal_faz(cnt_word1,cnt_word1x)
                sum_fx += self.unigram_counter[wordx] / self.NoBos_total_cnt
        return 1-sum_f1x/(1-sum_fx)

if __name__=="__main__":
    # 词频统计
    corpus_path = "./训练语料 utf-8.txt"    # 语料库 Corpus 所在路径
    unigram_savename = 'unigram.txt'
    unigram_path = './unigram.txt'

    bigram_savename = 'bigram'
    bigram_path = './bigram.ngram.txt'

    lang_model = LangModel(unigram_path,bigram_path,corpus_path,islog=True)
    # sent = input()
    print("开始测试\n"
    "1. 可输入 exit 退出测试程序\n"
    "2. 平滑方法，1:加一平滑，2:katz 平滑\n")
    while True:
        sent = input("请输入测试句子：")
        if sent=="exit":
            break
        type = input("请输入平滑方法：")
        if type=="1":
            lang_model.add_one_smmoth(sent, islog=True)
        elif type=="2":
            lang_model.katz_smooth(sent, islog=True)
        else:
            print("平滑方法输入错误")
```

utils.py

```
# -*- codeing = utf-8 -*-
# @Author: Maxpicca
# @Description: 工具包

from pyhanlp import *
import jieba
import time
from collections import Counter

# ==========导入 hanlp 类==========
```

```
# HanLP 统计单个单词词频，使用 DictionaryMaker
DictionaryMaker = SafeJClass('com.hankcs.hanlp.corpus.dictionary.DictionaryMaker')
# HanLP 统计两个单词词频，使用 NGramDictionaryMaker
NGramDictionaryMaker =
SafeJClass('com.hankcs.hanlp.corpus.dictionary.NGramDictionaryMaker')
# HanLP 语料库加载类
CorpusLoader = SafeJClass("com.hankcs.hanlp.corpus.document.CorpusLoader")
# HanLP Word 类
Word =  SafeJClass("com.hankcs.hanlp.corpus.document.sentence.word.Word")

def get_word_freq(corpus_path="./训练语料 utf-8.txt",
unigram_savename='unigram.txt', bigram_savename='bigram'):
    '''
    获取语料库词频\n
    :param corpus_path: 语料库路径
    :param unigram_savename: 单元，词频统计结果存储文件名
    :param bigram_savename: 双元，词频统计结果存储文件名
    :return:
    '''
    print("词频统计中...")
    # ==========读取语料==========
    sentences = CorpusLoader.convert2SentenceList(corpus_path)  # 返回
List<List<IWord>>类型
    for sent in sentences:
        # 设置头、尾部
        sent[0] = Word("BOS", "begin")
        sent.addLast(Word("EOS", "end"))

    # ==========创建词频统计对象=========
    dict_maker = DictionaryMaker()
    ngram_maker = NGramDictionaryMaker()

    # =========统计词频=========
    for sent in sentences:
        # 一阶频次，只需要统计
        dict_maker.add(sent[0])
        for i in range(1, len(sent)):
            dict_maker.add(sent[i])
            ngram_maker.addPair(sent[i - 1], sent[i])

    # =========本地存储词频统计结果=========
    dict_maker.saveTxtTo(unigram_savename)
    ngram_maker.saveTxtTo(bigram_savename)
```

```
def my_read_dict(filepath):
    '''
    文件读取词频结果，返回字典索引\n
    :param filepath: 文件存储路径\n
    :return: 结果字典
    '''
    res_dict = {}
    with open(filepath,encoding='utf-8') as f:
        for line in f:
            line = line.strip().split(" ")
            res_dict[line[0]] = int(line[-1])
    # num_list = list(unigram_counter.values())
    return res_dict

def get_unigram_list(sent):
    '''
    获取句子分词结果的单元列表\n
    :param sent:
    :return:
    '''
    temp_list = jieba.lcut(sent)
    unigram_list = ['BOS']+temp_list+['EOS']
    return unigram_list

if __name__=="__main__":
    corpus_path = "./训练语料 utf-8.txt"    # 语料库 Corpus 所在路径
    unigram_savename = 'unigram.txt'
    bigram_savename = 'bigram'
    get_word_freq(corpus_path,unigram_savename,bigram_savename)
```