上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

项目名称：会不会说人话

团队名称：Three Snakes

团队成员：胡栋侃 519120910183

王家栋 519120910162

郑思滢 519120910148

指导老师：鲍杨

一、 选题原因

近年来，随着互联网的不断普及和发展，各类网络文体不断涌现，可谓风格迥异，百花齐放，百家争鸣。例如前段时间，形如“二月十三是怎么回事呢？二月十三相信大家都很熟悉，但是二月十三是怎么回事呢，下面就让小编带大家一起了解吧。”的小编体风靡网络。对不同的文本进行分析，不难发现，每种类型的网络文体往往有鲜明的特点，相对固定的写作模版，内容上有也有着一定的相似性。如果通过机器学习的方式利用计算机对网络文体的典型特征进行归纳，能否让计算机写出各种经典的网络文体呢？

从这个问题出发，我们小组计划设计一款翻译器，实现各类网络文体与正常文体之间的双向转换。考虑到流行程度和文体特征，我们小组最终选择翻译体，小编体，鲁迅体这三类文体的双向转换，并由每一位小组成员分别负责一类文体的转换。

二、 项目简介

2.1 解决思路

将整个项目分为正向翻译（原始文本--->网络文体）和反向翻译（网络文体--->原始文本）两个部分。正向翻译通过人为获取网络文体的典型特征，并与输入的原始文本相结合，实现原始文本到网络文体的正向翻译。反向翻译，首先通过对数据的收集，构建所要翻译的网络文体的数据库，再通过机器学习算法实现输入网络文体文本分类的预测并得到各类网络文体文本中使用频率较高的词汇集合，根据预测结果，从输入文本中删除或是替换掉该类文本中出现频率较高的词汇，实现网络文体到原始文本的反向翻译

2.2 项目细节

2.2.1 翻译体的正向翻译（胡栋侃）

导入项目所需的包和第三方类库

```
import jieba
import jieba.analyse
import random
import os
import sys
```

输入想要进行翻译的原始文本并对其进行文本的预处理，包括根据逗号所在的位置将文本进行初步的切分并将每一段存储到 src_list 列表中，通过 jieba.analyse 获得文本中的关键词，并将关键词存储到 n1 中。

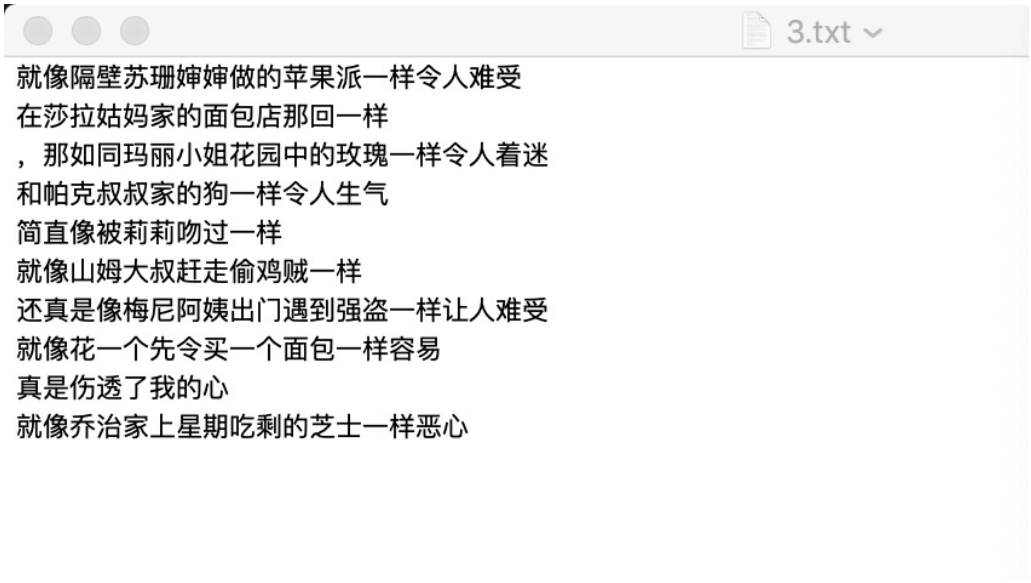
```
src= input("input your sentence here")
src_list=src.split(', ')
src_temp = src.strip()
re = jieba.analyse.extract_tags(sentence=src_temp,topK=1,allowPOS=('ns','n'))
n1= str(re[0])
```

事先收集翻译体较为经典的句式结构，并分别存储在三个 txt 文件中。

1.txt

2.txt

3.txt



通过对三个 txt 文本的读取，将三个 txt 文件中的内容逐行保存在 words1，words2，words3 中，并通过 random 函数从中随机挑选一句并存储至变量 words_1,words_2,words_3 中。将所选取的句子与输入文本结合，输出翻译体文本。

```
f=open("3.txt","r")
words3=f.read().splitlines()
f.close
words_3=words3[random.randrange(0,len(words3))]
src_list.insert(1,"我是说"+n1+words_3)
src_temp=", ".join(src_list)
f=open("1.txt","r")
words1=f.read().splitlines()
f.close
words_1=words1[random.randrange(0,len(words1))]
f=open("2.txt","r")
words2=f.read().splitlines()
f.close
words_2=words2[random.randrange(0,len(words2))]
print(words_1+src_temp+words_2)
```

样例输入与输出：

```
input your sentence here怎么还有十几个人没做单元测试，下午两点之后我就导出成绩，导出成绩后再补交的同学不能得满分。
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/h0/sstgv3mn373gx01yf6_4z3_40000gn/T/jieba.cache
Loading model cost 0.668 seconds.
Prefix dict has been built successfully.
真是不敢相信！怎么还有十几个人没做单元测试，我是说成绩就像隔壁苏珊婶婶做的苹果派一样，下午两点之后我就导出成绩，导出成绩后再补交的同学不能得满分。真是见鬼！
```

2.2.2 小编体的正向翻译（王家栋）

导入项目所需的包和第三方类库

```
import jieba
import jieba.analyse
import random
import os
import sys
```

按输入的内容将小编体分为两种形式，第一种为没有具体内容的输入，这里定义为输入的文本长度小于 6 个字符。此情况较为简单，因此直接将输入的句子带入小编体格式。

```
src= input("input your sentence here")
if len(src)<6:
    print(f"{src}是什么意思？{src}是什么梗？{src}是谁？这个梗又是从何而来？为什么一瞬间就有好多人使用这个梗？为什么大家都在{src}？相信不少同学都很想了解，下面就让小编来为大家介绍一下{s
    print(f"{src}是什么意思？{src}是什么梗？{src}是谁？这个梗又是从何而来？为什么一瞬间就有好多人使用这个梗？为什么大家都在{src}？")
    print(f"以上就是{src}的全部内容，希望能够帮助大家")
```

以下为输入与输出样例

```
input your sentence here吃饭
吃饭是什么意思？吃饭是什么梗？吃饭是谁？这个梗又是从何而来？为什么一瞬间就有好多人使用这个梗？为什么大家都在吃饭？相信不
少同学都很想了解，下面就让小编来为大家介绍一下吃饭的详细内容。
吃饭是什么意思？吃饭是什么梗？吃饭是谁？这个梗又是从何而来？为什么一瞬间就有好多人使用这个梗？为什么大家都在吃饭？
以上就是吃饭的全部内容，希望能够帮助大家
(base) PS C:\Users\wjd\Desktop>
```

当输入的内容大于 6 个字符时，采取第二种处理方法。

首先输入想要进行翻译的原始文本并对其进行文本的预处理，再通过 jieba.analyse 获得文本中的关键词，并将权重最大的关键词存储到 n1 中。这里，我们通过“allowPOS=('ns','n')”命令优先选取名词。

```
try:
    src_temp = src.strip()
    print('src_temp',src_temp)
    re = jieba.analyse.extract_tags(sentence=src_temp,topK=1,allowPOS=('ns','n'))

    n1= str(re[0])
```

但是有时候句子中可能不存在名词关键词，会存在代码错误，因此采用 try/except 结构，当名词关键词不存在时，运行下面这段代码，提取任意词性的关键词，并储存到 n2 中，

```
except:
    src_temp = src.strip()
    print('src_temp',src_temp)
    re = jieba.analyse.extract_tags(sentence=src_temp,topK=1)
    n2= str(re[0])
```

下面是第二种情况的输入与输出样例：

```
(base) PS C:\Users\wjd\Desktop> python 小编体.py
input your sentence here风筝是由古代劳动人民发明于中国东周春秋时期的产物，至今已2000多年。相传墨翟以木头制成木鸟，研制三年而成，是人类最早的风筝起源。后来鲁班用竹子，改进墨翟的风筝材质，直至东汉期间，蔡伦改进造纸术后，坊间才开始以纸做风筝，称为"纸鸢"。
src_temp 风筝是由古代劳动人民发明于中国东周春秋时期的产物，至今已2000多年。相传墨翟以木头制成木鸟，研制三年而成，是人类最早的风筝起源。后来鲁班用竹子，改进墨翟的风筝材质，直至东汉期间，蔡伦改进造纸术后，坊间才开始以纸做风筝，称为"纸鸢"。
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\wjd\AppData\Local\Temp\jieba.cache
Loading model cost 0.772 seconds.
Prefix dict has been built successfully.
风筝是什么意思？风筝是什么梗？风筝是谁？这个梗又是从何而来？为什么一瞬间就有好多人使用这个梗？为什么大家都在风筝？相信不少同学都很想了解，下面就让小编来为大家介绍一下风筝的详细内容。
风筝是由古代劳动人民发明于中国东周春秋时期的产物，至今已2000多年。相传墨翟以木头制成木鸟，研制三年而成，是人类最早的风筝起源。后来鲁班用竹子，改进墨翟的风筝材质，直至东汉期间，蔡伦改进造纸术后，坊间才开始以纸做风筝，称为"纸鸢"。
以上就是风筝的全部内容，希望能够帮助大家
(base) PS C:\Users\wjd\Desktop>
```

2.2.3 鲁迅体的正向翻译（郑思滢）

导入项目所需包及第三方类库

```python
import requests
from lxml import etree
import pandas as pd
import time
import jieba
import jieba.analyse
import re
import sys
import random
sys.path.append('../')
```

从网站上爬取一些鲁迅的名言

```python
#爬一些鲁迅名言
#获取数据
headers = {"user-agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36" }
item_list = []
for i in range(0, 9):
    url = "http://www.shuoshuodaitupian.com/writer/128_" + str(i + 1)
    result = requests.get(url, headers=headers).content.decode()


    #解析数据
    html = etree.HTML(result)
    div_list = html.xpath('//div[@class="item statistic_item"]')
    div_list = div_list[1:-1]

    for div in div_list:
        item = {}
        item['content'] = div.xpath('./a/text()')[0]
        item['source'] = div.xpath('./div[@class="author_zuopin"]/text()')[0]
        item['source'] = div.xpath('.//a[@class="infobox zan like "]/span/text()')[0]

        item_list.append(item)

    print("正在爬第{}页".format(i + 1))
    time.sleep(0.1)
```

	A	B	C	D	E
1		content	source		
2	0	我家门前有两棵树，一棵是枣树，另一棵也是枣树。	2		
3	1	愿中国青年都摆脱冷气，只是向上走，不必听自暴自弃者流的话。能	13		
4	2	我以为就是圣贤豪杰，也不必自惭他的童年，自惭，倒是一个错误。	2		
5	3	倘若说，作品愈高，知音愈少，那么，推论起来，谁也不懂的东西，	6		
6	4	诚信为人之本。	2		
7	5	使一个人的有限的生命，更加有效，也即等于延长了人的生命。	2		
8	6	要在文化上有成绩，则非韧不可。	1		
9	7	经历一多，便能从前因而知后果，我的预测时时有验，只不过由此一	2		
10	8	人生的旅途，前途很远，也很暗。然而不要怕，不怕的人的面前才有	3		
11	9	哪里有天才，我是把别人喝咖啡的工夫都用在了工作上了。	4		
12	10	与其找糊涂导师，倒不如自己走。	2		
13	11	不孝的人是世界最可恶的人。	2		
14	12	中国惟有国魂是最可宝贵的，惟有他发扬起来，中国人才真有进步。	3		
15	13	死亡的生命已经朽腐，我对于这朽腐有大欢喜，因为我借此知道它还	2		
16	14	过去的生命已经死亡，我对于这死亡有大欢喜，因为我借此知道它曾	2		
17	15	惟沉默是最高的轻蔑。	15		
18	16	其实世上本没有路，走的人多了，也便成了路。	4		
19	17	人生得一知己足矣。	1		
20	18	感谢命运，感谢人民，感谢思想，感谢一切我要感谢的人。	1		
21	19	金子做了骨髓，也还是站不直。	1		
22	20	贪安稳就没有自由，要自由就要历些危险，只有这两条路。	2		

通过 jieba 分词对鲁迅作品分词并做词频分析，筛选出鲁迅喜欢使用的高频词。在最初几轮词频分析中先筛选出一些停用词包括年份以及不符合当代语境的用词等，最终分析所得高频词有便是、大抵、至于等。创建字典储存对应替换词。

```
#查找鲁迅常用词
r1 = '[a-zA-Z0-9’!"#$%&\'()*+,-./:;<=>?@，。？、…【】《》？“”‘’！[\\]^_`{|}~]+'

for i in range(1,8):
    with open("鲁迅卷{}.txt".format(i), 'r', errors='ignore') as file:
        text = file.read()
    text = re.sub(r1, '', text)
    jieba.analyse.set_stop_words("stop_words.txt")
    keys = jieba.analyse.extract_tags(text, topK=20)
    print(",".join(keys))


#common_results=[青年,诗人,便是,大抵,还是,社会,思想,似乎,至于,觉得,似的,无常,但是,之类,不过,对于,还是,并非,好像,觉得,关于,至于,人们,还有,文坛,恐怕,当然,实在]

transfer = {
'认为':'以为',
'大约':'大抵',
'大概':'大抵',
'特别':'尤',
'吧':'罢了',
'你':'伊',
'越':'愈发',
'等':'之类',
'都':'皆',
'不':'莫',
}
```

用户输入待转换句子后，先对句子做分词，遍历每个分词，如出现字典中替换词则替换为对应的鲁迅常用词。

```
src= input("input your sentence here")

src_temp = src.strip()
#print('src_temp',src_temp)
re = list(jieba.cut(sentence=src_temp))
#print(re)


for i in range(len(re)):
    if re[i] in transfer:
        re[i] = transfer[re[i]]
print(re)
re = ''.join(re)
```

再从先前创建的鲁迅名言中随机选取一句，最后以固定格式将用户输入句子按鲁迅风格的格式输出。

```
f = open('鲁迅名言.txt','r',encoding='UTF-8')
sentences = f.read().splitlines()
f.close()
add_sentence = sentences[random.randrange(0,len(sentences))]
add_sentence = add_sentence[:-1]
print('至于'+ str(re) + ',我想大抵是因为' + str(add_sentence)+'的缘故罢')

```

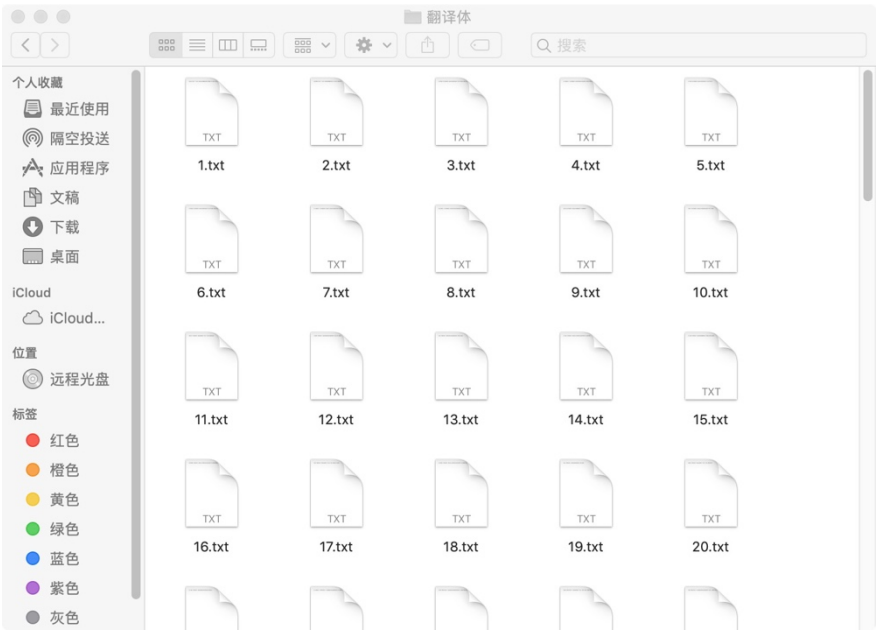
以下为输出案例

```
(base) PS C:\Users\hp\Desktop\python\groupproject\鲁迅体> ipython .\test.py
input your sentence here一定要早睡，但早睡大概是不存在的
src_temp 一定要早睡，但早睡大概是不存在的
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\hp\AppData\Local\Temp\jieba.cache
Loading model cost 0.874 seconds.
Prefix dict has been built successfully.
['一定', '要', '早睡', ',', ',', '但', '早睡', '大概', '是', '不', '存在', '的']
['一定', '要', '早睡', ',', ',', '但', '早睡', '大抵', '是', '莫', '存在', '的']
至于一定要早睡，但早睡大抵是莫存在的,我想大抵是因为改革自己，总比制止别人来得难的缘故罢
```

```
(base) PS C:\Users\hp\Desktop\python\groupproject\鲁迅体\鲁迅体> ipython .\luxuntest.py
input your sentence here当你越了解自己以及自己想要的东西，你就越不会被外界困扰
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\hp\AppData\Local\Temp\jieba.cache
Loading model cost 0.873 seconds.
Prefix dict has been built successfully.
['当', '你', '越', '了解', '自己', '以及', '自己', '想要', '的', '东西', '，', '，', '你', '就', '越', '不会', '被', '外界', '困扰']
['当', '伊', '愈发', '了解', '自己', '以及', '自己', '想要', '的', '东西', '，', '，', '伊', '就', '愈发', '不会', '被', '外界', '困扰']
至于当伊愈发了解自己以及自己想要的东西，伊就愈发不会被外界困扰，我想大抵是因为我觉得坦途在前，人又何必因了一点小障碍而不走路呢的缘故罢
```

2.2.4 三类文本的反向翻译（合作）

第一步是数据的收集工作，每位同学负责提供各自负责的一类文本的数据，共 120 组数据，其中 100 组为投喂数据，用以建立 3 种文体的特征词频矩阵，分装在 data 文件夹中，20 组为测试数据，用以检验所用的算法的正确率，并和 5 组 Error 数据分装在 test 文件夹中。在计算词频的时候，会将一些常用词语设置为停用词，避免加入到词频的计算中，停用词的数据放在 stop 文件夹中。



第二步是对文本分类算法的设计

导入样本所需要的包和第三方类库

```
import jieba
from numpy import *
import pickle  # 持久化
import os
from sklearn.feature_extraction.text import TfidfTransformer  # TF-IDF向量转换类
from sklearn.feature_extraction.text import TfidfVectorizer  # TF_IDF向量生成类
from sklearn.datasets.base import Bunch
from sklearn.naive_bayes import MultinomialNB  # 多项式贝叶斯算法
```

读取各文件夹及二级文件夹中的数据，并进行数据的预处理，包括删除多余的空行和空格，将文本利用 jieba 分词工具分开，分词结果用空格隔开，并将所得到的结果保存在 split 文件夹中

```
def segText(inputPath, resultPath):
    fatherLists = os.listdir(inputPath)  # 主目录
    for eachDir in fatherLists:  # 遍历主目录中各个文件夹
        eachPath = inputPath + eachDir + "/"  # 保存主目录中每个文件夹目录，便于遍历二级文件
        each_resultPath = resultPath + eachDir + "/"  # 分词结果文件存入的目录
        if not os.path.exists(each_resultPath):
            os.makedirs(each_resultPath)
        childLists = os.listdir(eachPath)  # 获取每个文件夹中的各个文件
        for eachFile in childLists:  # 遍历每个文件夹中的子文件
            eachPathFile = eachPath + eachFile  # 获得每个文件路径
            #  print(eachFile)
            content = readFile(eachPathFile)  # 调用上面函数读取内容
            # content = str(content)
            result = (str(content)).replace("\r\n", "").strip()  # 删除多余空行与空格
            # result = content.replace("\r\n","").strip()

            cutResult = jieba.cut(result)  # 默认方式分词，分词结果用空格隔开
            saveFile(each_resultPath + eachFile, " ".join(cutResult))  # 调用上面函数保存文件
```

```
def getStopWord(inputFile):
    stopWordList = readFile(inputFile).splitlines()
    return stopWordList
```

算法主体部分，通过 TF-IDF 进行 data 数据文本词频的计算，通过 TfidfTransformer 实现 TF-IDF 向量的转换，TfidfVectorizer 实现 TF_IDF 向量的生成，MultinomialNB 为多项式贝叶斯算法，是文本分类用到的主要算法。通过将文本转换为词频矩阵，实现了文本的机器可读性，并将文本的频率保存在 testSpace_arr.txt 文档中，便于程序运行过程的可视化。构建完成数据集的向量空间后，导入训练集的词袋，并获取训练集的词频信息，通过训练集和数据集的词频比对进行文本类别的预测

```
def getTFIDFMat(inputPath, stopWordList, outputPath,
                tftfidfspace_path,tfidfspace_arr_path,tfidfspace_vocabulary_path):  # 求得TF-IDF向量
    bunch = readBunch(inputPath)
    tfidfspace = Bunch(target_name=bunch.target_name, label=bunch.label, filenames=bunch.filenames, tdm=[],
                       vocabulary={})
    tfidfspace_out = str(tfidfspace)
    saveFile(tftfidfspace_path, tfidfspace_out)
    # 初始化向量空间
    vectorizer = TfidfVectorizer(stop_words=stopWordList, sublinear_tf=True, max_df=0.5)
    transformer = TfidfTransformer()  # 该类会统计每个词语的TF-IDF权值
    # 文本转化为词频矩阵，单独保存字典文件
    tfidfspace.tdm = vectorizer.fit_transform(bunch.contents)
    tfidfspace_arr = str(vectorizer.fit_transform(bunch.contents))
    saveFile(tfidfspace_arr_path, tfidfspace_arr)
    tfidfspace.vocabulary = vectorizer.vocabulary_  # 获取词汇
    tfidfspace_vocabulary = str(vectorizer.vocabulary_)
    saveFile(tfidfspace_vocabulary_path, tfidfspace_vocabulary)
    writeBunch(outputPath, tfidfspace)
```

```
def getTestSpace(testSetPath, trainSpacePath, stopWordList, testSpacePath,
                 testSpace_path,testSpace_arr_path,trainbunch_vocabulary_path):
    bunch = readBunch(testSetPath)
    # 构建测试集TF-IDF向量空间
    testSpace = Bunch(target_name=bunch.target_name, label=bunch.label, filenames=bunch.filenames, tdm=[],
                      vocabulary={})
    testSpace_out = str(testSpace)
    saveFile(testSpace_path, testSpace_out)
    # 导入训练集的词袋
    trainbunch = readBunch(trainSpacePath)
    # 使用TfidfVectorizer初始化向量空间模型  使用训练集词袋向量
    vectorizer = TfidfVectorizer(stop_words=stopWordList, sublinear_tf=True, max_df=0.5,
                                 vocabulary=trainbunch.vocabulary)
    transformer = TfidfTransformer()
    testSpace.tdm = vectorizer.fit_transform(bunch.contents)
    testSpace.vocabulary = trainbunch.vocabulary
    testSpace_arr = str(testSpace.tdm)
    trainbunch_vocabulary = str(trainbunch.vocabulary)
    saveFile(testSpace_arr_path, testSpace_arr)
    saveFile(trainbunch_vocabulary_path, trainbunch_vocabulary)
    # 持久化
    writeBunch(testSpacePath, testSpace)
```

```
def bayesAlgorithm(trainPath, testPath,tfidfspace_out_arr_path,
                   tfidfspace_out_word_path,testspace_out_arr_path,
                   testspace_out_word_apth):
    trainSet = readBunch(trainPath)
    testSet = readBunch(testPath)
    clf = MultinomialNB(alpha=0.001).fit(trainSet.tdm, trainSet.label)
    # alpha:0.001 alpha 越小，迭代次数越多，精度越高
    # print(shape(trainSet.tdm))  #输出单词矩阵的类型
    # print(shape(testSet.tdm))
```

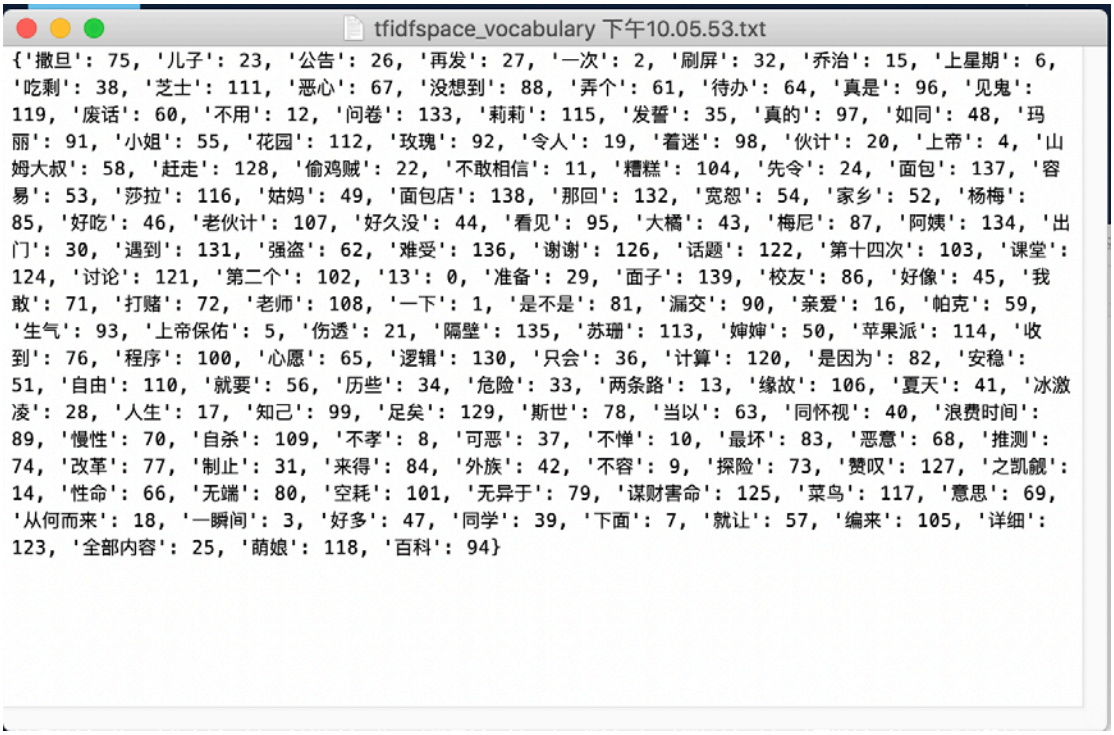
结果处理部分，输出文本的预测类别和实际类别，若文本的预测类别与实际类别不符，则将该项内容记录到错误率中。

```
    '''处理结束'''
    predicted = clf.predict(testSet.tdm)
    total = len(predicted)
    rate = 0
    for flabel, fileName, expct_cate in zip(testSet.label, testSet.filenames, predicted):
        if flabel != expct_cate:
            rate += 1
        print(fileName, ":实际类别: ", flabel, "-->预测类别: ", expct_cate)
    print("erroe rate:", float(rate) * 100 / float(total), "%")
```

以下是输出结果，由输出结果可以看出，文本的预测较为成功，每组 test 文件中的 errortest 都被顺利的识别，其他的预测也保持着较高的准确率，验证了机器学习的算法的可行度

```
./split/test_split/鲁迅体/14.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/16.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/17.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/13.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/12.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/10.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/11.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/Error2.txt :实际类别： 鲁迅体 -->预测类别： 翻译体
./split/test_split/鲁迅体/Error3.txt :实际类别： 鲁迅体 -->预测类别： 翻译体
./split/test_split/鲁迅体/Error1.txt :实际类别： 鲁迅体 -->预测类别： 小编体
./split/test_split/鲁迅体/Error4.txt :实际类别： 鲁迅体 -->预测类别： 翻译体
./split/test_split/鲁迅体/Error5.txt :实际类别： 鲁迅体 -->预测类别： 小编体
./split/test_split/鲁迅体/9.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/8.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/5.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/4.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/6.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/7.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/3.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/2.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/1.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/20.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/19.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/鲁迅体/18.txt :实际类别： 鲁迅体 -->预测类别： 鲁迅体
./split/test_split/待测/待测1.txt :实际类别： 待测 -->预测类别： 翻译体
./split/test_split/小编体/15.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/14.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/16.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/17.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/13.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/12.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/10.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/11.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/Error2.txt :实际类别： 小编体 -->预测类别： 鲁迅体
./split/test_split/小编体/Error3.txt :实际类别： 小编体 -->预测类别： 翻译体
./split/test_split/小编体/Error1.txt :实际类别： 小编体 -->预测类别： 翻译体
./split/test_split/小编体/Error4.txt :实际类别： 小编体 -->预测类别： 鲁迅体
./split/test_split/小编体/Error5.txt :实际类别： 小编体 -->预测类别： 翻译体
./split/test_split/小编体/9.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/8.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/5.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/4.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/6.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/7.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/3.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/2.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/1.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/20.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/19.txt :实际类别： 小编体 -->预测类别： 小编体
./split/test_split/小编体/18.txt :实际类别： 小编体 -->预测类别： 小编体
erroe rate: 21.05263157894737 %
```

在本程序运行的过程中也获得了三类文本的高频词汇，作为进行反向翻译的重要依据。



第三步，进行对输入文本的预测和反向翻译。

将所要预测的输入文本的 txt 形式放到 test 文件夹中的预测文本文件夹中，运行函数即可获得输入文本的预测数据

预测文本.txt

```
./split/test_split/预测文本/预测文本.txt        预测类别：  翻译体
```

运行反向翻译.py 文件，再次输入需要翻译的文本并进行反向翻译，通过对文本进行分词，读取之前得到的高频词汇文件，并将文本中出现的高频词汇删去实现反向翻译

```python
import time
import jieba
import jieba.analyse
import re
import sys
import random
#以上为用到的数据库
sys.path.append('../')
src= input("input your sentence here")#输入想要翻译的文本

src_temp = src.strip()
re = list(jieba.cut(sentence=src_temp))#对文本进行分词
print(re)
f = open('frequency.txt','r',encoding='UTF-8')
sentences = f.read().split(', ')
f.close()#读取高频词汇的文件并保存在sentences列表中
for i in range(len(re)):
    if re[i] in sentences or re[i]==', ':
        re[i] = ''#将文本中出现的高频词汇删去
re_1 = ''.join(re)
print(re_1)
```

样例输入与输出

```
input your sentence here哦，你这撒旦的儿子，我把群公告再发一次，没想到弄个群待办会刷屏真是见鬼！
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/h0/sstgv3mn373gx01yf6_4z3_40000gn/T/jieba.cache
Loading model cost 0.567 seconds.
Prefix dict has been built successfully.
['哦', '，', '你', '这', '撒旦', '的', '儿子', '，', '我', '把', '群', '公告', '再发', '一次', '，', '没想到', '弄个', '群', '待办', '会', '刷屏', '真是', '见鬼', '！']
我把群公告再发一次没想到弄个群待办会刷屏！
```

三、第三方类库介绍

3.1 jieba 中文分词工具

Jieba 中文分词工具基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)，并采用动态规划查找最大概率路径，找出基于词频的最大切分组合，通过 jieba 分词工具，我们实现了以下几个功能。

```python
src_temp = src.strip()
print('src_temp',src_temp)
re = list(jieba.cut(sentence=src_temp))
print(re)   #通过jieba分词工具将原始文本分为小节
```

通过 jieba.cut 实现中文文本的切分

```
src_temp = src.strip()
print('src_temp',src_temp)
re = jieba.analyse.extract_tags(sentence=src_temp,topK=1,allowPOS=('ns','n'))#通过工具jieba.analyse得到原始文本的关键词

n1= str(re[0])
```

通过 jieba.analyse 实现文本中关键词的获取

3.2 Scikit-learn 机器学习工具

Scikit-learn(sk-learn)是机器学习中常用的第三方模块，对常用的机器学习方法进行了封装，包括回归(Regression)、降维(Dimensionality Reduction)、分类(Classfication)、聚类(Clustering)等方法。Sklearn 具有以下特点：简单高效的数据挖掘和数据分析工具，让每个人能够在复杂环境中重复使用，建立于 NumPy、Scipy、MatPlotLib 之上。在本次项目中，我们主要运用了机器学习工具中的 TF-IDF 算法，用以评估字词对于一个文件集或一个语料库中的其中一份文件的重要程度。词的重要性随着在文件中出现的次数正比增加，同时随着它在语料库其他文件中出现的频率反比下降。就是说一个词在某一文档中出现次数比较多，其他文档没有出现，说明该词对该文档分类很重要。然而如果其他文档也出现比较多，说明该词区分性不大，就用 IDF 来降低该词的权重。

数学算法：

1.词频的统计

TF-IDF 与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比

TF-IDF = TF (词频) * IDF(逆文档频率)

词频：TF = 词在文档中出现的次数 / 文档中总词数

逆文档频率：IDF = log（语料库中文档总数 / 包含该词的文档数 +1 ）

避开停用词：啊，诶，诶呀等（具体可见文件夹 stopwords）

2.文本分类预测

运用到了贝叶斯算法，即$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$

在本项目中的具体表现形式为：$P(\text{类别}|\text{特征}) = \frac{\text{p(特征|类别)p（类别）}}{\text{p（特征）}}$

四、项目局限性与改进

1.反向翻译效度较低，翻译结果通顺程度较差，且考虑到标点符号的难以取舍，采用了将所有逗号去除的方式，所以项目在应对较长句子时会比较吃力。改进策略是提升机器学习的样本数量，进一步细化词频矩阵，对长句可以采用由标点符号分割为多个短句的方式来进一步精细化翻译结果

2.本项目仅供图一乐，中国汉字博大精深，无论是 jieba 分词工具还是机器学习文本分类，在对汉字的处理上仍然存在一定的缺陷，同时，由于项目数据库并不大，在识别方面仍旧会产生一定的误差，且目前只限于三类文本的双向翻译，改进策略为采用精度更高的分词和分类工具，同时进一步扩大项目数据库，以此推广到对更多文体的翻译，形成更完善，更强大的识别。

五、总结

通过本次大作业项目，我们小组熟悉了 python 中对中文的处理方式以及数据的爬取，构建自己的数据库等实用的技能，同时我们也初步了解了机器学习文本分类的方法，接触了 TDF-IF 算法和贝叶斯算法。虽然我们做出来的成果尚不能让人十分满意，存在一定的缺陷，

但每一位小组成员的付出都获得了收获，另外，项目的趣味性较强，项目的成果也给我们小组成员带来了很多的快乐，也许这就是 The Zen of Python。