

**《程序设计基础》**

**AI诗人实验报告**

目录

[北京邮电大学数字媒体与设计艺术学院 1](#_Toc535029109)

[2019年 1 月 1 日 1](#_Toc535029110)

[一、 作业概述 3](#_Toc535029111)

[<1>项目目的 3](#_Toc535029112)

[<2>项目内容 3](#_Toc535029113)

[1.基于词频分析 3](#_Toc535029114)

[2. 基于tensorflow的机器学习 8](#_Toc535029115)

[3.（参考）诗词爬取和歌词爬取 10](#_Toc535029116)

[4.另一个机器学习的模型 10](#_Toc535029117)

[5.百度情感识别系统/百度DNN语言模型 14](#_Toc535029118)

[<3>项目意义 14](#_Toc535029119)

[二、 数据分析 14](#_Toc535029120)

[<1>基于词频 14](#_Toc535029121)

[<2>基于tensorflow 15](#_Toc535029122)

[<3>总结 18](#_Toc535029123)

[三、 诗词生成程序的算法比较 18](#_Toc535029124)

[<1>基于词频 18](#_Toc535029125)

[<2>基于tensorflow 22](#_Toc535029126)

[四、 实验结果分析 26](#_Toc535029127)

# 作业概述

## <1>项目目的

主要通过基于词频分析和基于tensorflow的深度学习两种方法让程序自动写诗，并通过写诗的结果来分析写诗的原理以及两种写诗方法的异同。

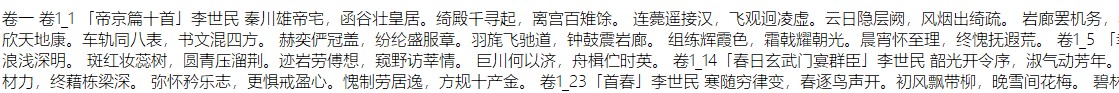
## <2>项目内容

### 1.基于词频分析

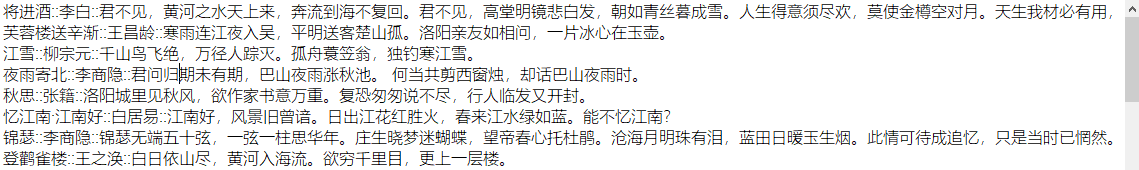
#### （1）爬虫

主要文件：\_\_init\_\_.py

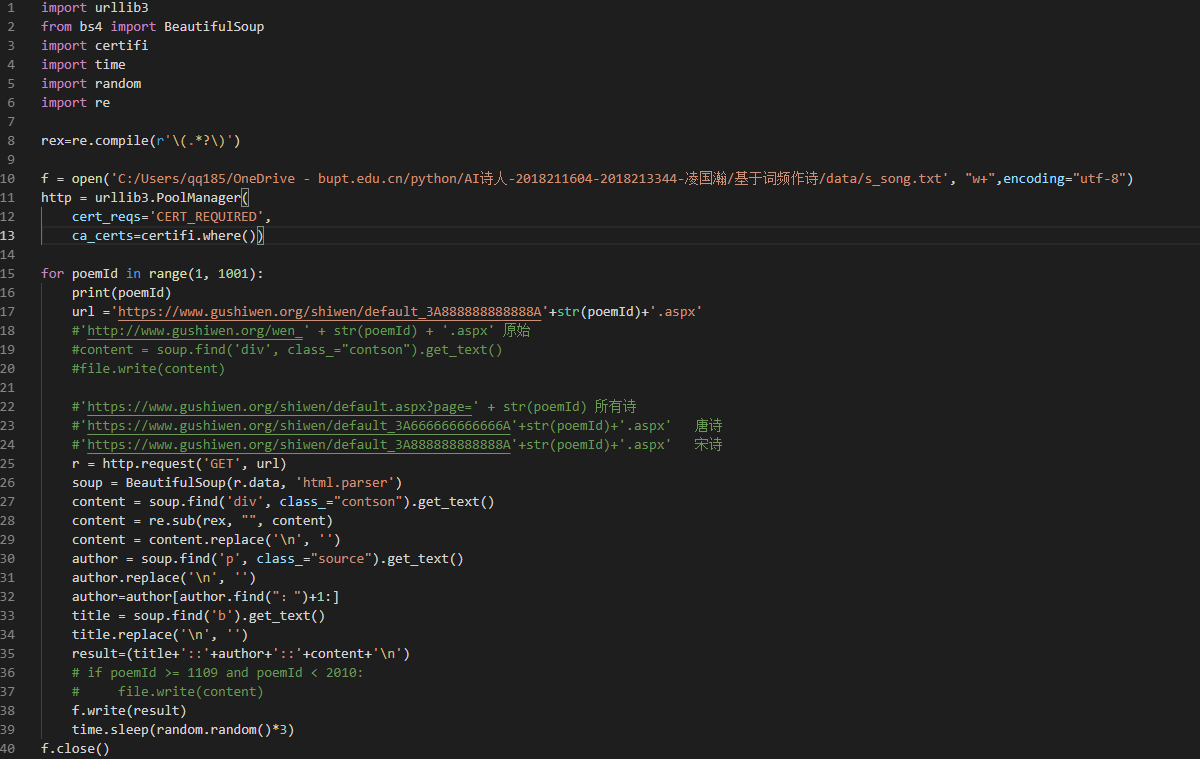
从<http://www.gushiwen.org>网站爬取诗句，用于词频分析时格式如下：



用于tensorflow作诗时格式如下：



代码如下：

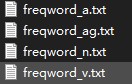


与老师所给代码不同的是，此处我使用了urllib3库而不是requests库

#### （2）高频词统计

主要文件：dataHandler.py

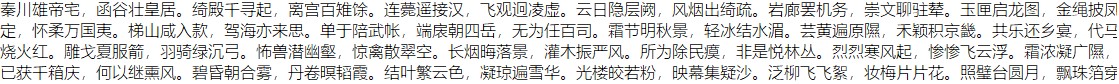
使用jieba库，统计形容词，副词，名称以及动词并且提取其高频词



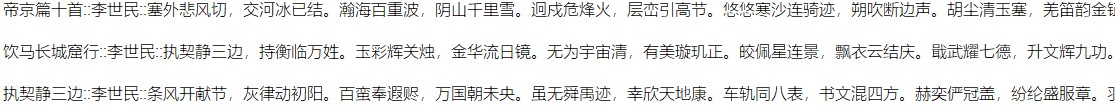
#### （3）数据处理

主要文件：zzcf.py mlzzcf.py

Zzcf.py使用正则表达式，在爬取的数据中仅提取诗句与标点，处理后格式如下：



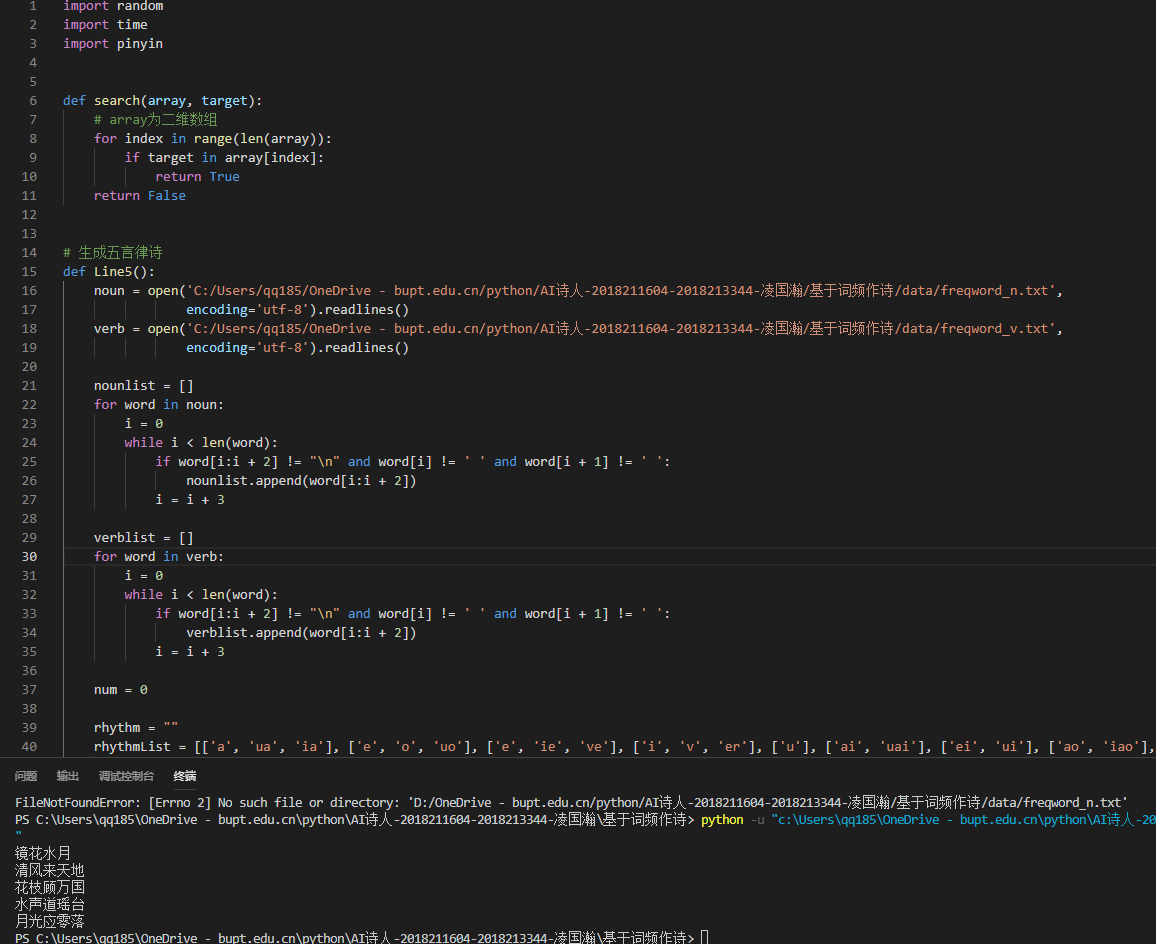
Mlzzcf.py（原创）则是将数据处理为适用于基于tensorflow的作诗代码，可以添加进其源数据之中，处理后格式如下：



#### （4）生成五言律诗（主要调研对象）

主要文件：TangshiGene.py

代码如下：

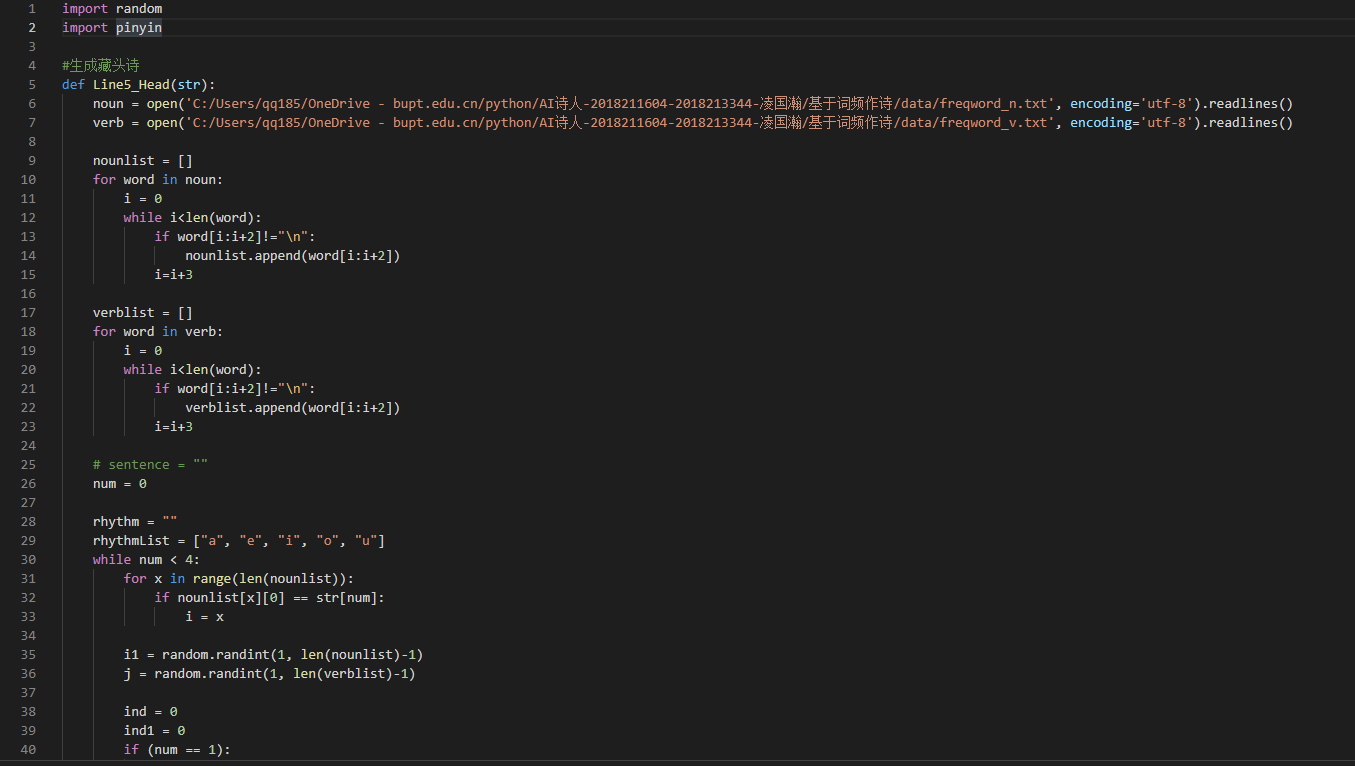


#### （5）生成藏头诗

主要文件：TangshiGene2.py

此处仅需要输入藏头诗的第一个字组成的词语便可以生成

代码如下：

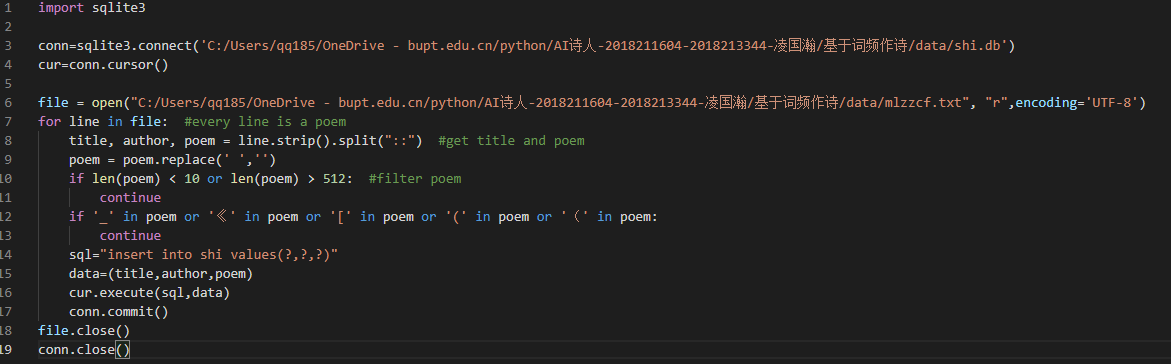


#### （6）生成数据库

主要文件：toDB.py，shi.db

将使用的诗句（mlzzcf.txt）转换为数据库，便于处理

代码如下：

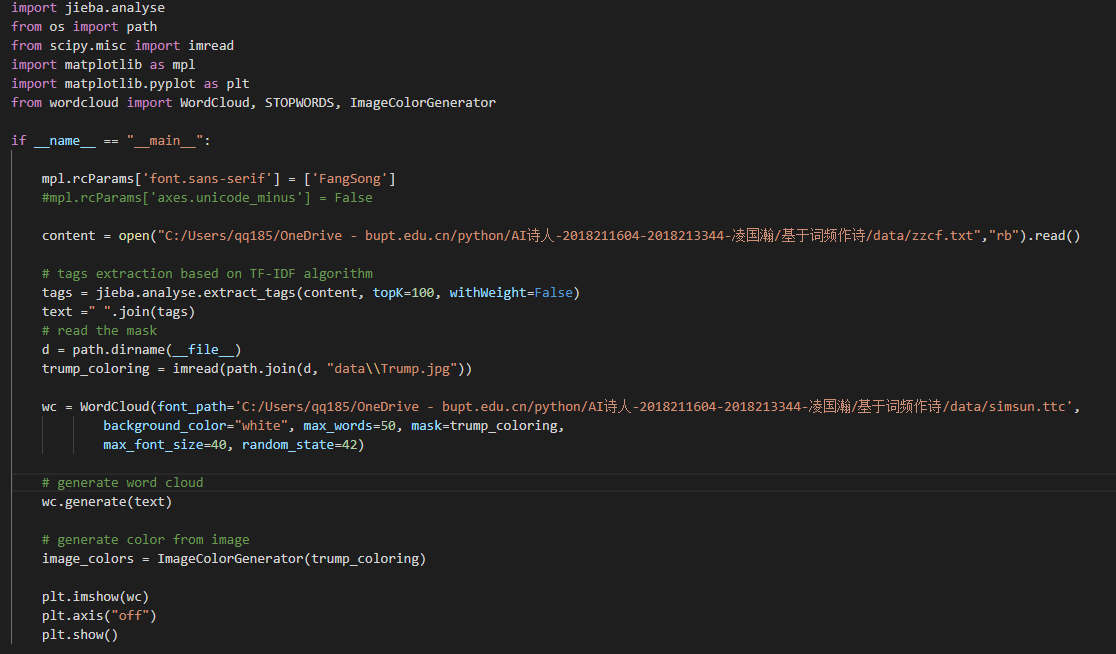


#### （7）生成词云

主要文件：wordcould.py, Figure\_1.png

根据高频词生成词云

代码如下：

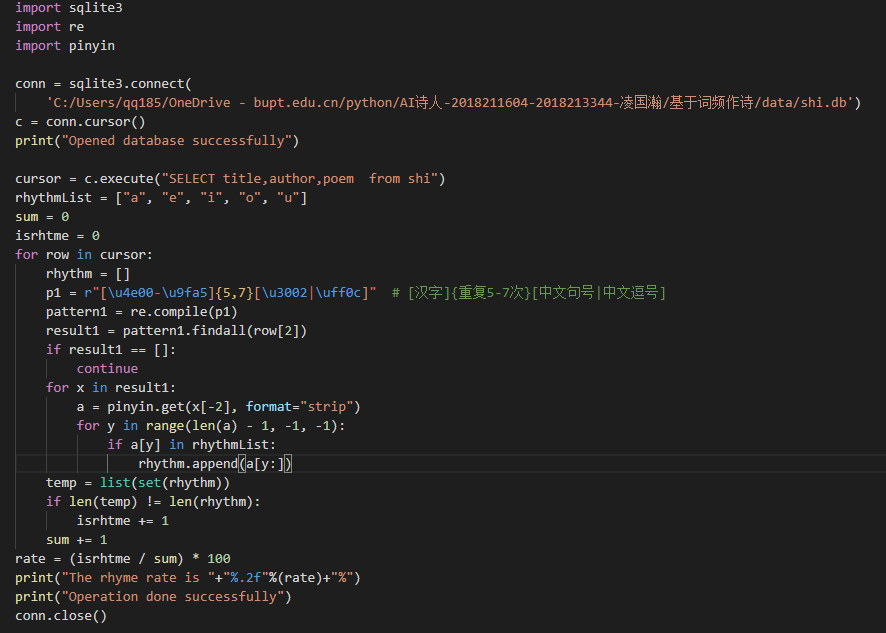


效果见数据分析

#### （8）数据押韵分析

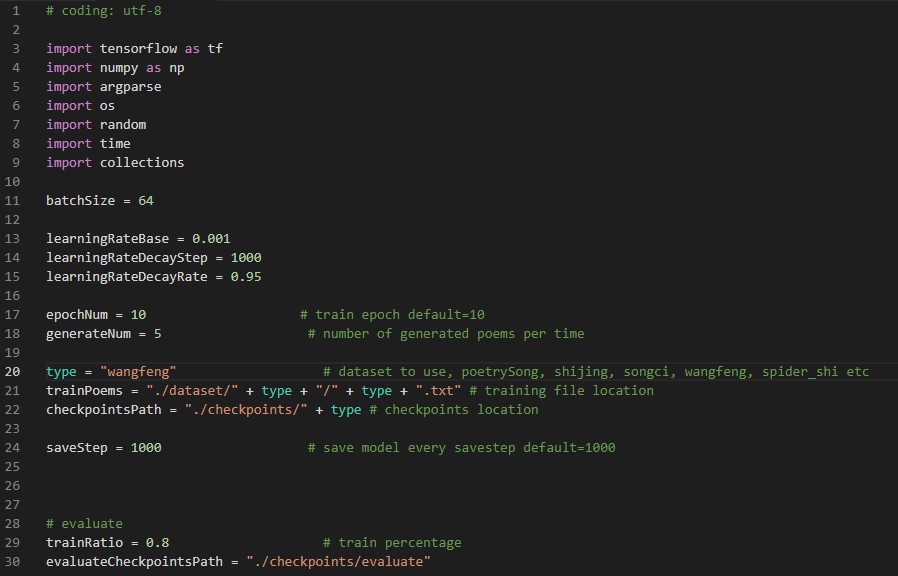
主要文件：rhymerate.py

统计源数据中押韵的诗句比例

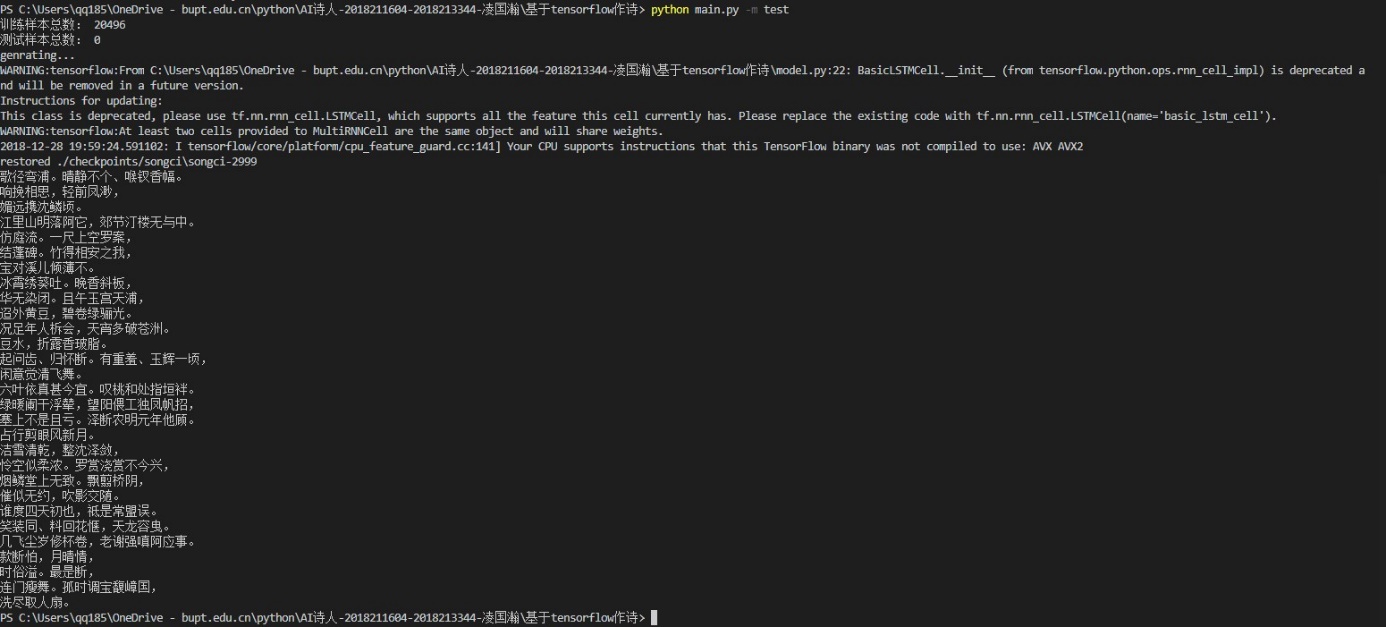


### 2. 基于tensorflow的机器学习

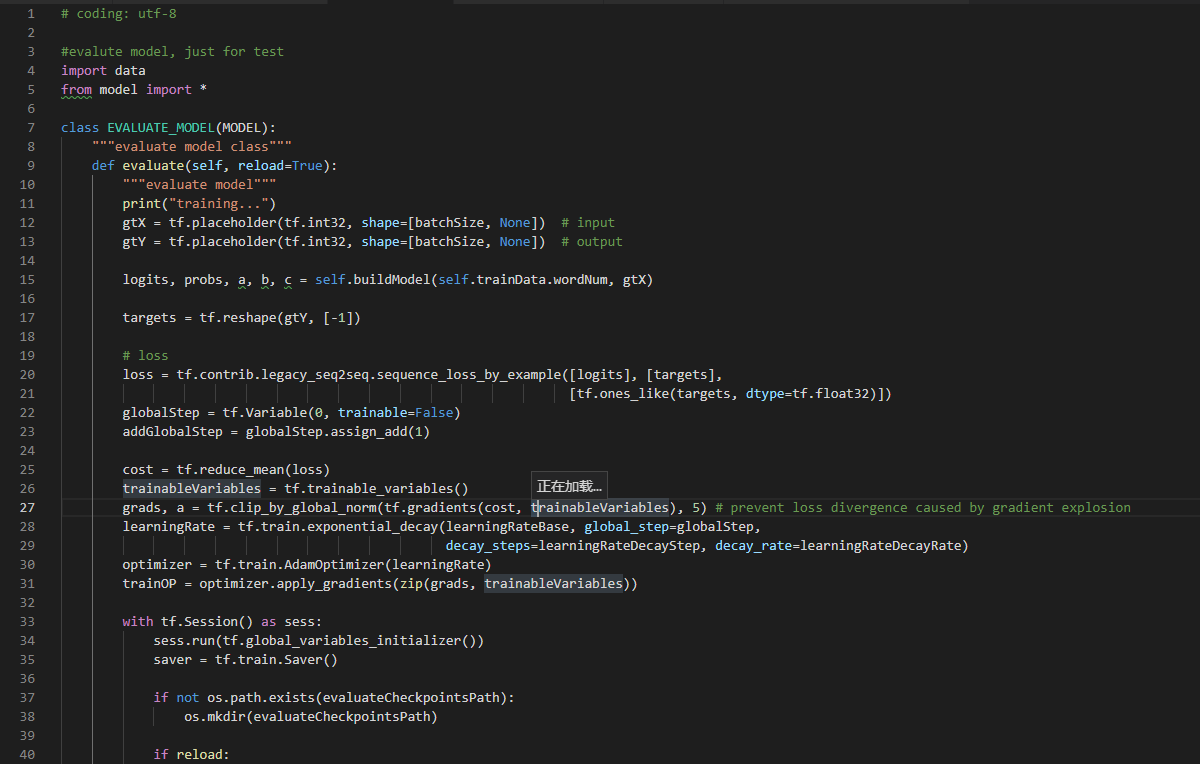
使用python的tensorflow库创建训练(train)、作诗(test)和藏头诗(head)的模型。在一个统一的项目config.py中设置相关参数（训练次数、训练诗句类型等）并调用相关库：



在调试控制台中输入python main.py -m test/head/train 可以分别开启作诗、作藏头诗、训练三种功能，以作诗为例：



此处是用已经训练的宋词模型作的诗



判断模型拟合程度的代码

除此之外，还有段程序能够使得json格式的文本转换为该项目所需的数据格式，见dataset文件夹中的jsonToTxt.py

注：该项目中也有数据库生成、词云生成和数据押韵分析，也有提取纯诗句的程序（onlypoem.py）用于生成词云，但与基于词频作诗项目大同小异，此处略去

### 3.（参考）诗词爬取和歌词爬取

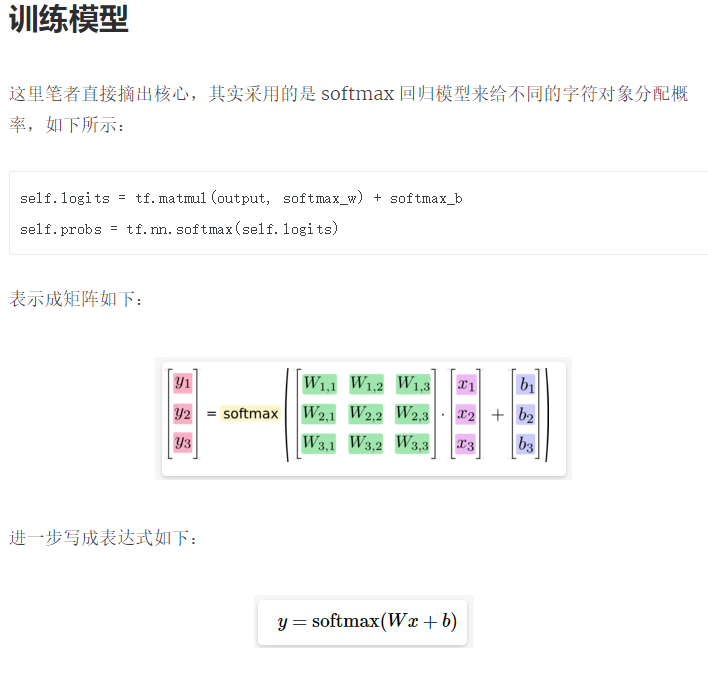
由老师提供的代码完成的诗词与歌词的爬取，但由于歌词样本过小，同时所爬取的诗词类型混杂，下均不以此两例说明

### 4.另一个机器学习的模型

仍然是基于tensorflow的机器学习模型，但是作者给出了更为详细的代码分析

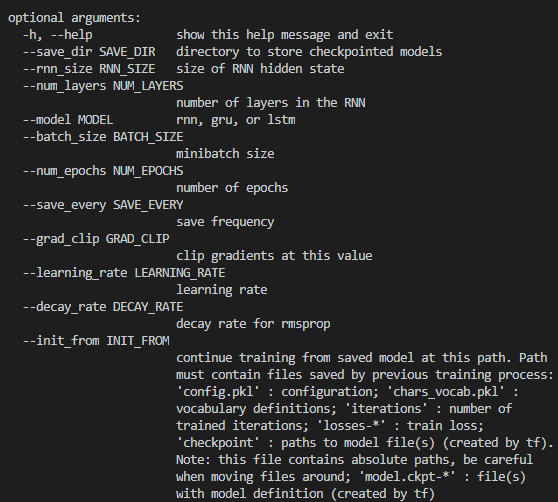


作者将诗句处理为矩阵便于程序处理

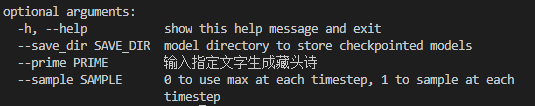


作者称，此处采用的是softmax回归模型：<http://deeplearning.stanford.edu/wiki/index.php/Softmax%E5%9B%9E%E5%BD%92>

但这方面已经超出了我们的学习范围



训练可选项，可在控制台通过python trainer.py+相关可选项 来调用



作诗可选项，可在控制台通过python generator.py+相关可选项 来调用



生成诗句

上文：陶令啄文地，诗书屐齿声。晚阴须一树，前艇得薰弦。干\*书高处，吟贫匪薜萝。独含荒草在，我上伴僧期。泛酒相携赋，苍龙终日求。别离空羽盖，占耳问新汀。三叶开迷径，危泉隔怕春。秋云临碧水，蕙叶下房墙。



藏头诗

### 5.百度情感识别系统/百度DNN语言模型

#### （1）百度情感识别系统

目的：对包含主观观点信息的文本进行情感极性类别（积极、消极、中性）的判断，并给出相应的置信度。

主要文件：sentiment analysis.py



基于tensorflow的诗词数据的情感分析



基于词频作诗的诗词数据的情感分析

#### （2）百度DNN语言模型

目的：判断一句话是否符合语言表达习惯

主要文件：dnnmode.py



基于tensorflow的诗词源数据的可读性分析



基于词频作诗的诗词数据的可读性分析

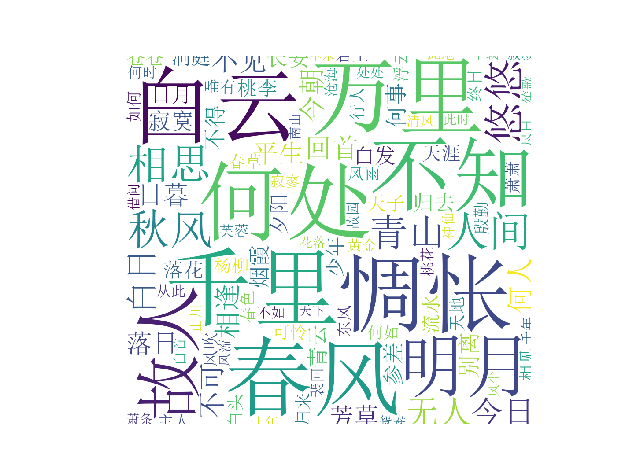
## <3>项目意义

初步了解词频生成和机器学习两种机器自动作诗方法，并比较两者的异同，了解两者的优劣，特别是当前行业热点的机器学习方面，使得对python这门语言在更深层次上的应用有了更多的了解。与此同时，也能够初步了解不同算法对相同任务处理后的结果，认识到算法的重要性。

# 数据分析

## <1>基于词频

该项目的数据基于<http://www.gushiwen.org>网站所爬取的几千首诗句，以及基于tensorflow项目中的唐诗数据，将其中的高频词提出后生成如下词云（不分词性，来源：zzcf.txt）



所采用的数据中押韵诗句与非押韵诗句比率如下：



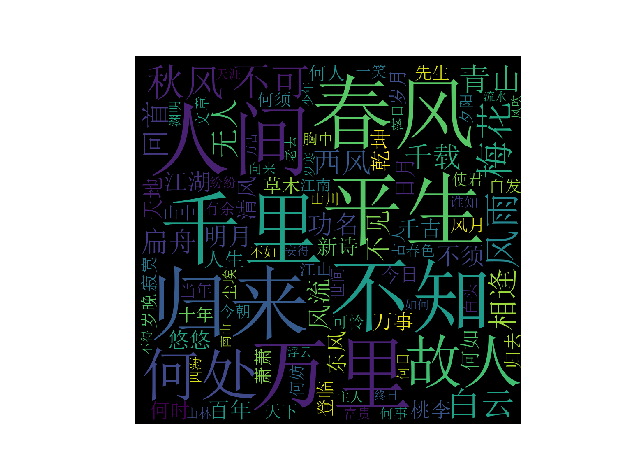
为97.34%

## <2>基于tensorflow

该项目诗词数据来源：<https://github.com/chinese-poetry/chinese-poetry>

包含5.5万首唐诗、26万首宋诗和2.1万首宋词

各个数据生成的词云如下（来源：poetySong.txt）：



以下为原始数据作者整理的词云数据：



唐诗高频词



宋诗高频词



宋词高频词

所采用的数据中押韵诗句与非押韵诗句比率如下：



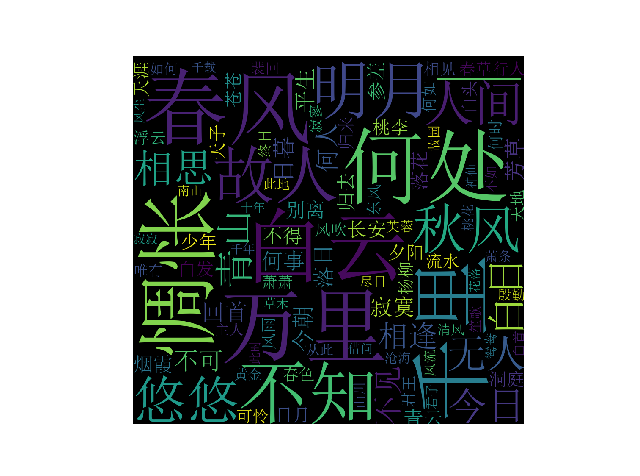
为94.74%

数据准确率如下（evaluate.py）：

epoch为4 epoch为10

对于额外的tensorflow模型，通过wordcloud也统计出了其词云，如下图：



另一个tensorflow模型的词云

## <3>总结

显然，机器学习作诗所需的数据量要远远高于词频作诗。因为词频作诗仅仅是提取诗词中的高频词，再按照一定规律随机生成诗句，数据量的大小和输出诗句的形式和优美程度没有相关性。

而机器学习则需要庞大的数据量来训练模型才能够确保输出的准确性，动辄上万的数据量显然要大的多。当然，数据量不能无限制的增大，否则会出现“过拟合”等现象。

而二者数据的押韵比率差别不大，可以预见其对结果的影响不如其他变量。

# 诗词生成程序的算法比较

## <1>基于词频

该项目的基本思想是，在一定规则（押韵）下，在高频词中选取词语随机生成诗句。

爬虫获取数据

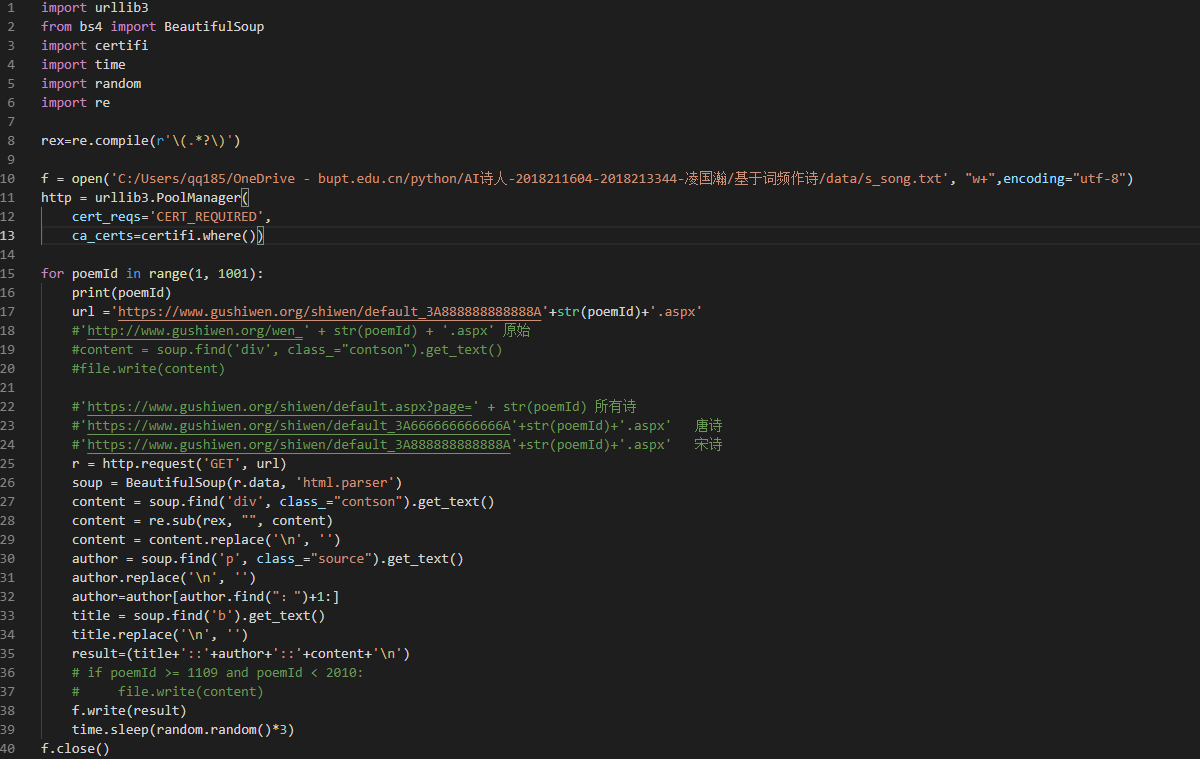
数据处理

词频统计

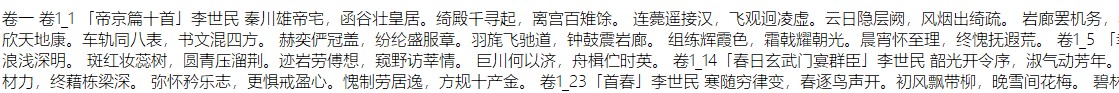
随机生成

输出诗句

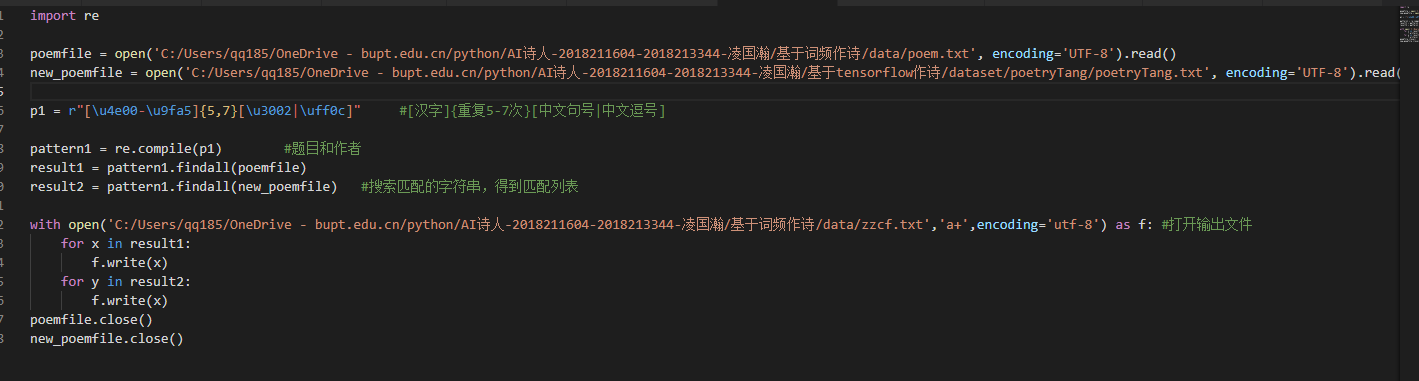
在爬虫部分，该项目是基于urllib3（爬取HTML）和BeautifulSoup4（处理HTML）爬取古诗文网的数据，代码如下：



在以上代码中，urllib3用于爬取HTML，BeautifulSoup4用于处理HTML，certifi的作用是令爬虫使用Mozilla的根证书（反反爬），最后不能缺少time模块的停顿。这样以后才能使得数据正确的被下载。最终得到如下数据（使用原始的url而非例图的唐诗url）：

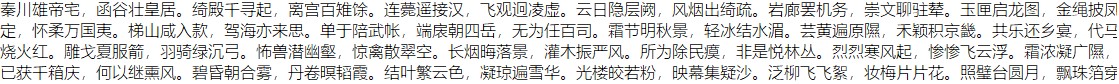


再使用正则表达式将其变为可处理的形式，代码如下：

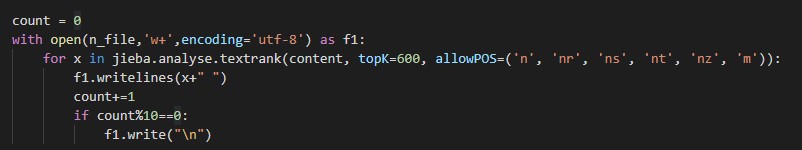


此处额外使用了tensorflow项目的数据

处理后文件格式为：



接下来使用jieba模块提取各个词性的高频词，数据每10个换一行

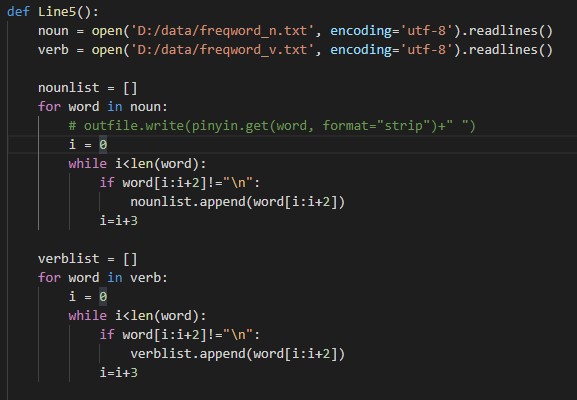


以名词为例

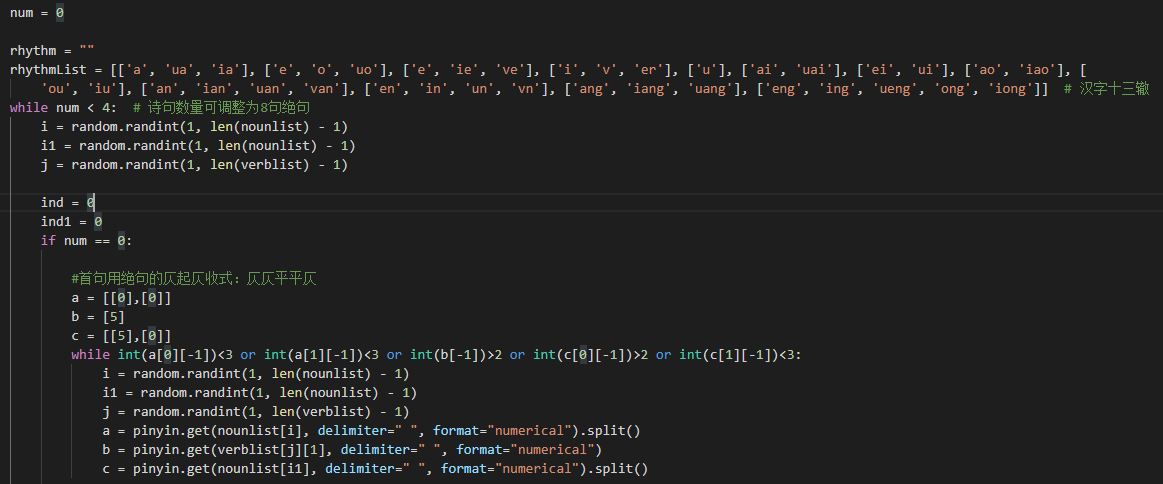


数据如上

最后是诗句的生成，通过pinyin模块控制平仄和韵脚，再以“名名动名名”的形式输出诗句



储存名词和动词



所有诗句都使用random模块初始化，再利用pinyin库限制生成诗句的平仄关系



此处处理韵脚的押韵关系，保证每句诗都能押韵，最后以名名动名名的形式输出

在以上代码中，也可以更改num的限制条件，改变诗句的长度，诗的句数以及哪几句诗句押韵。

## <2>基于tensorflow

调用数据

进行训练

训练成功

Checkpoint文件

执行main文件

head

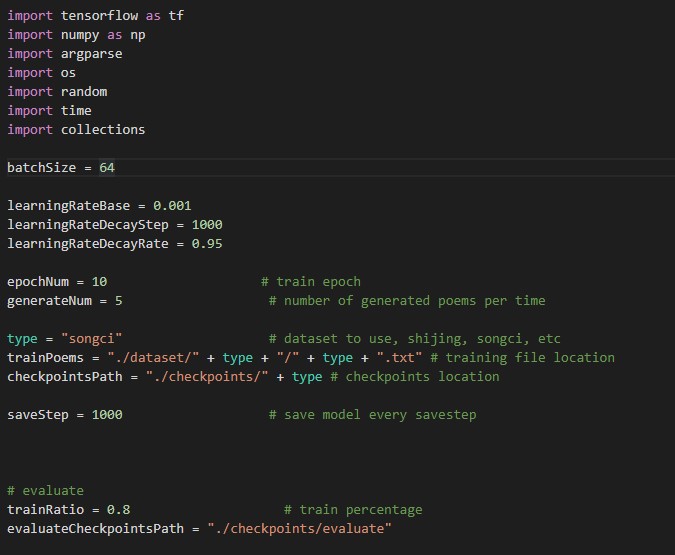
test

生成模型

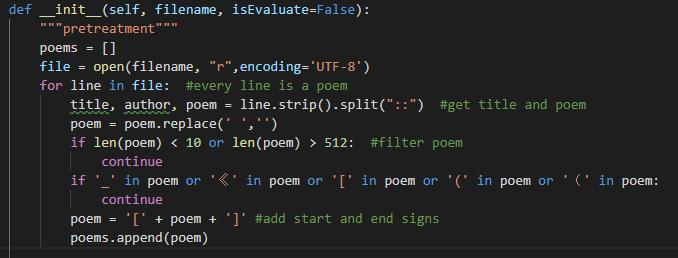
生成藏头诗/诗句

该项目的基本思想是，通过机器学习，让机器“学会”写诗。通过生成的模型，能够写出与数据相似的诗句。

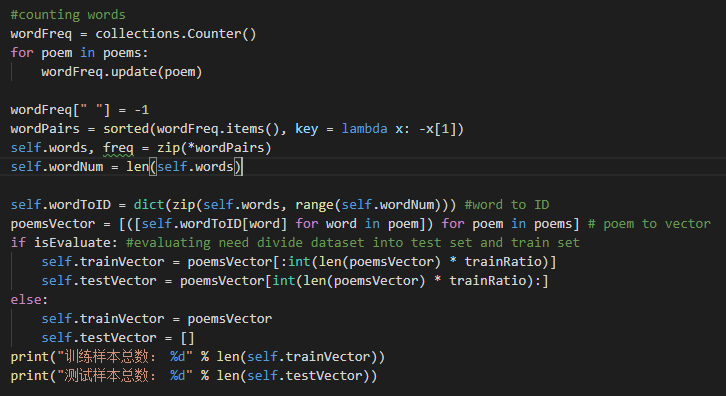
可操作和调整的主要代码在config.py中（主要是train方面）



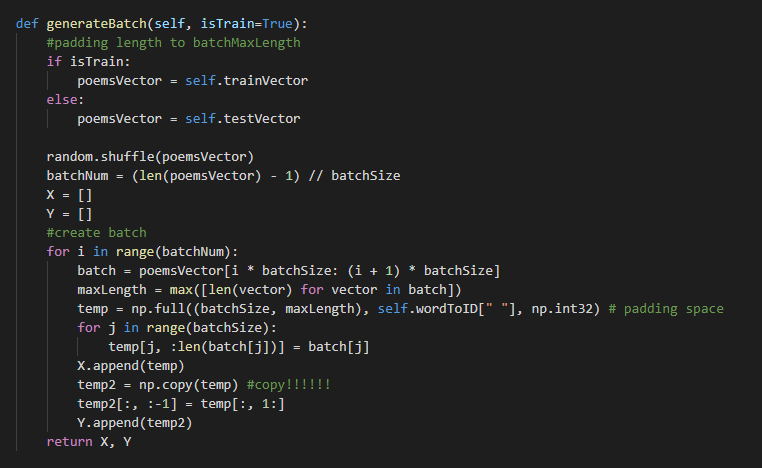
该文件可以调整数据的大小，批次，拟合数据，训练epoc数，以及在训练到何时保存数据



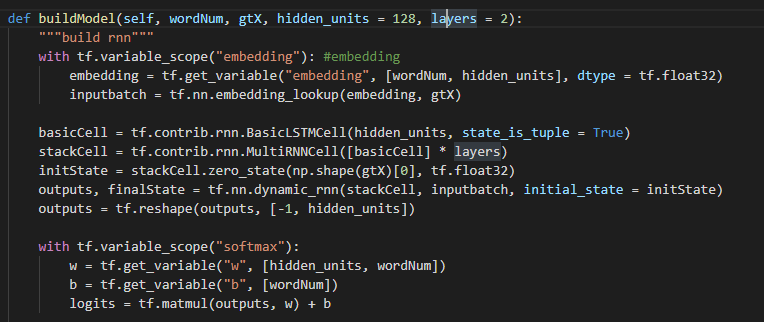
处理dataset里的诗词数据，将题目和内容分离，然后清洗过滤掉一些不好的训练样本，包含特殊符号、字数太少或太多的都去除，最后在诗的前后分别加上开始和结束符号（方括号）



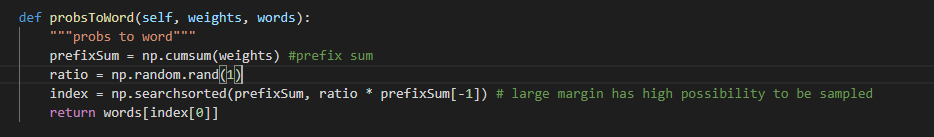
统计样本总数



因为诗的长度不一致，需要用空格填补，以便于模型学习



搭建LSTM模型



该方法（probsToWord）引入了一些随机性，保证每次写的诗都不一样

# 实验结果分析



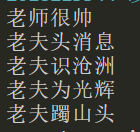
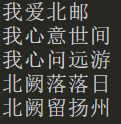
此两图为词频作诗结果

（以下分析以非藏头诗为例）

可以看到词频作诗的结果在格式上没有错误，也的确做到了第二句和第四句的押韵（“二”与“儿”），同时第一句诗还能做到“仄仄平平仄”的仄起仄收式的开头。而在诗词本身的意义方面，其百度DNN模型得分为34217.8



而且更值得一提的是，如果在词频作诗中的藏头诗的诗头不为四个名词，那么往往不能正确的作诗。（如下图）



并且，很多时候诗的最后一个字也会发生重复





此两图为机器学习作诗结果

反观机器学习的作诗结果，格式正确，但是韵脚的押韵却没能做到，不过在平仄关系上却表现尚佳。

平仄关系为：

平平平仄仄平平，平仄平平平平仄。仄仄平平仄仄仄，平平平仄平平平。

可以明显的看出，平仄关系较词频作诗有了长足进步（特别是第一句和第四句）

其百度DNN模型得分为4952.1



那么为什么会出现这样的原因呢？

在[**二、数据分析**](#_基于tensorflow)中已经提到，数据大小就不会影响词频作诗的准确率和可读性，即使是我们仅仅给程序几个关键词它也能作诗。很显然，词频作诗的一大优势就是其基于算法的稳定性，一旦算法要求哪几处押韵或平仄，该程序也会严格执行。但也正是固定的算法制约了它。可以说，只要不改变生成诗句的算法，基于词频生成的诗句几乎永远也不会拥有完整的平仄关系等诗的基本属性，甚至词性也会被限制，更不用说语义了（ppl=34217.8）。与此同时，基于高频关键字写诗也说明了作诗时使用的数据实际上不是爬取的诗，而仅仅是数个高频词。在如此小的数据量下写藏头诗，面对用户层出不穷的输入，程序也不免黔驴技穷，这就是藏头诗诗头和押韵诗韵脚出现雷同的原因。

而机器学习则完全不同。通过对相当数量的诗句的学习，该程序能写出越来越像人类诗人的诗句，在这个过程中，平仄关系和押韵关系也慢慢的被“学习”。可以预测的是，在样本量达到一定数量以后，甚至连如何写富有语义的诗句的能力也能被“学习”下来（ppl=4952.1）。然而在样本量或训练量不足的情况下，想要机器学习作诗也遵循严格的平仄和押韵还是勉为其难了。这主要是因为其在数据量和训练量不足时缺乏算法的约束。

因此，对于哪种作诗算法好，我们就可以粗略的得出结论：数据量小时，基于词频的算法作诗质量高；数据量大时，基于机器学习的算法作诗质量高。当然，当提供的数据不全是同一种类型的诗的时候（如唐诗+宋词）机器学习就有较大的劣势了。总之，对算法的输出结果影响最大的还是其使用的原始数据。