[Python人工智能] 二十一.Word2Vec+CNN中文文本分类详解及与机器学习（RF\DTC\SVM\KNN\NB\LR）分类对比

原创 Eastmount 2020-06-29 20:26:54 5719 收藏 18 原力计划 编辑 版权

分类专栏： Python+TensorFlow人工智能 文章标签： Python人工智能 文本分类 CNN 随机森林 Word2Vec

从本专栏开始，作者正式研究Python深度学习、神经网络及人工智能相关知识。前一篇文章分享了Keras实现RNN和LSTM的文本分类算法，并与传统的机器学习分类算法进行对比实验。这篇文章我们将继续巩固文本分类知识，主要讲解CNN实现中文文本分类的过程，并与贝叶斯、决策树、逻辑回归、随机森林、KNN、SVM等分类算法进行对比。注意，本文以代码为主，文本分类叙述及算法原理推荐阅读前面的文章。基础性文章，希望对您有所帮助~

本专栏主要结合作者之前的博客、AI经验和相关视频及论文介绍，后面随着深入会讲解更多的Python人工智能案例及应用。基础性文章，希望对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作者作为人工智能的菜鸟，希望大家能与我在这一笔一划的博客中成长起来。写了这么多年博客，尝试第一个付费专栏，但更多博客尤其基础性文章，还是会继续免费分享，但该专栏也会用心撰写，望对得起读者，共勉！

TF下载地址：https://github.com/eastmountyxz/AI-for-TensorFlow
Keras下载地址：https://github.com/eastmountyxz/AI-for-Keras
博客下载地址：https://github.com/eastmountyxz/CSDNBlog-AI-for-Python

本文参考B站“一只乌龟大王”老师的视频，并结合作者的实战经验进行讲解，再次感谢该老师。

- https://www.bilibili.com/video/BV1Zk4y1r7Gd

文章目录

同时推荐前面作者另外五个Python系列文章。从2014年开始，作者主要写了三个Python系列文章，分别是基础知识、网络爬虫和数据分析。2018年陆续增加了Python图像识别和Python人工智能专栏。

- Python基础知识系列：Python基础知识学习与提升
- Python网络爬虫系列：Python爬虫之Selenium+BeautifulSoup+Requests

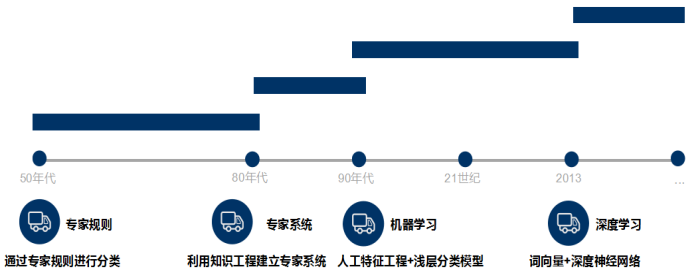
- Python数据分析系列：知识图谱、web数据挖掘及NLP
- Python图像识别系列：Python图像处理及图像识别
- Python人工智能系列：Python人工智能及知识图谱实战

前文：

[Python人工智能] 一.TensorFlow2.0环境搭建及神经网络入门
[Python人工智能] 二.TensorFlow基础及一元直线预测案例
[Python人工智能] 三.TensorFlow基础之Session、变量、传入值和激励函数
[Python人工智能] 四.TensorFlow创建回归神经网络及Optimizer优化器
[Python人工智能] 五.Tensorboard可视化基本用法及绘制整个神经网络
[Python人工智能] 六.TensorFlow实现分类学习及MNIST手写体识别案例
[Python人工智能] 七.什么是过拟合及dropout解决神经网络中的过拟合问题
[Python人工智能] 八.卷积神经网络CNN原理详解及TensorFlow编写CNN
[Python人工智能] 九.gensim词向量Word2Vec安装及《庆余年》中文短文本相似度计算
[Python人工智能] 十.Tensorflow+Opencv实现CNN自定义图像分类案例及与机器学习KNN图像分类算法对比
[Python人工智能] 十一.Tensorflow如何保存神经网络参数
[Python人工智能] 十二.循环神经网络RNN和LSTM原理详解及TensorFlow编写RNN分类案例
[Python人工智能] 十三.如何评价神经网络、loss曲线图绘制、图像分类案例的F值计算
[Python人工智能] 十四.循环神经网络LSTM RNN回归案例之sin曲线预测
[Python人工智能] 十五.无监督学习Autoencoder原理及聚类可视化案例详解
[Python人工智能] 十六.Keras环境搭建、入门基础及回归神经网络案例
[Python人工智能] 十七.Keras搭建分类神经网络及MNIST数字图像案例分析
[Python人工智能] 十八.Keras搭建卷积神经网络及CNN原理详解
[Python人工智能] 十九.Keras搭建循环神经网络分类案例及RNN原理详解
[Python人工智能] 二十.基于Keras+RNN的文本分类vs基于传统机器学习的文本分类
《人工智能狂潮》读后感——什么是人工智能？(一)

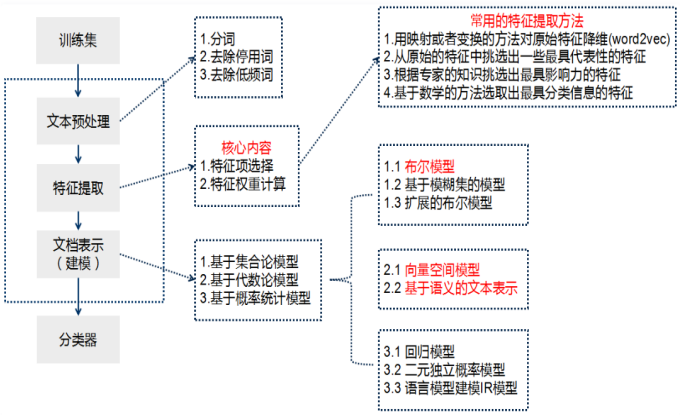
一.文本分类

文本分类旨在对文本集按照一定的分类体系或标准进行自动分类标记，属于一种基于分类体系的自动分类。文本分类最早可以追溯到上世纪50年代，那时主要通过专家定义规则来进行文本分类；80年代出现了利用知识工程建立的专家系统；90年代开始借助于机器学习方法，通过人工特征工程和浅层分类模型来进行文本分类。**现在多采用词向量以及深度神经网络来进行文本分类。**



牛亚峰老师将传统的文本分类流程归纳如下图所示。在传统的文本分类中，基本上大部分机器学习方法都在文本分类领域有所应用。主要包括：

- Naive Bayes
- KNN
- SVM
- 随机森林 \ 决策树
- 集合类方法
- 最大熵
- 神经网络



利用Keras框架进行文本分类的基本流程如下：

- 步骤 1：文本的预处理，分词->去除停用词->统计选择top n的词做为特征词
- 步骤 2：为每个特征词生成ID
- 步骤 3：将文本转化成ID序列，并将左侧补齐
- 步骤 4：训练集shuffle
- 步骤 5：Embedding Layer 将词转化为词向量
- 步骤 6：添加模型，构建神经网络结构
- 步骤 7：训练模型
- 步骤 8：得到准确率、召回率、F1值

注意，如果使用TFIDF而非词向量进行文档表示，则直接分词去停后生成TFIDF矩阵后输入模型。本文将采用词向量、TFIDF两种方式进行实验。

在知乎史老师的“https://zhuanlan.zhihu.com/p/34212945”里总结归类来说，基于深度学习的文本分类主要有5个大类别：

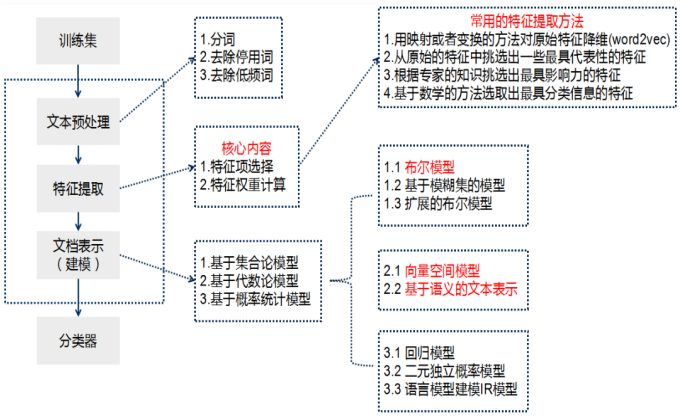
- 词嵌入向量化：word2vec, FastText等
- 卷积神经网络特征提取：TextCNN（卷积神经网络）、Char-CNN等
- 上下文机制：TextRNN（循环神经网络）、BiRNN、BiLSTM、RCNN、TextRCNN(TextRNN+CNN)等
- 记忆存储机制：EntNet， DMN等
- 注意力机制：HAN、TextRNN+Attention等

推荐牛亚峰老师的文章：基于 word2vec 和 CNN 的文本分类 ： 综述 & 实践

二.基于随机森林的文本分类

该部分主要围绕常见的文本分类案例进行讲解，由于随机森林效果较好，故主要分享该方法。具体步骤包括：

- 读取CSV中文文本
- 调用Jieba库实现中文分词及数据清洗
- 特征提取采用TF-IDF或Word2Vec词向量表示
- 基于机器学习的分类
- 准确率、召回率、F值计算及评估



1.文本分类

(1).数据集

本文的数据为近期贵州黄果树瀑布的旅游评论文本，来自大众点评网，共有240条数据，其中差评数据114条，好评数据126条，如下图所示：

	A	B
1	ocntent	label
2	还记得小时候，常常守在电视机前，等候《西游记》的播出。“你挑着担,我牵着	好评
3	飞流直下三千尺 疑是银河落九天！	好评
4	黄果树大瀑布景区，位置坐落在贵州省安顺市关岭县黄果树风景区。这里是贵州	好评
5	国家重点风景名胜区和首批国家5A级旅游景区 今天黄果树景区时而艳阳高照，时	好评
6	黄果树瀑布群绝对是值得一看的。景区非常大，不止黄果树瀑布，天星桥景区绝	好评
7	到了贵州，必须要到的就是黄果树瀑布！以黄果树瀑布为中心，周边分布着十几	好评
8	状观的黄果树瀑布 晴天隐隐响春雷， 绝顶奔来云深处。	好评
9	带父母带贵州玩，那一定要来贵州最有名的黄果树瀑布看看。父母年纪大了，所	好评
10	去年的时候来玩的，当时一直想着来到新看到的景色如何。没想到来到这之后，	好评
11	进门以后乘坐景区大巴前往第一站陡坡塘瀑布，据说这里是拍摄西游记的地方，	好评
12	暑假去的，大瀑布景区真的是人山人海 听说是西游记的取景地，还是蛮期待的	好评
13	来黄果树本来我们是自由行，没想到下车被一个包车的导游接错人，结果就加入	好评
14	黄果树瀑布绝对可以说是贵州旅游的标志，像我们这样的外地人去贵州旅游一般	好评
15	贵州黄果树瀑布景区。瀑布之宽，之高之大。像白色的幕布。落下之后，像湛蓝	好评
16	天星桥景区位于黄果树大瀑布下游，这里主要观赏石、水、树的美妙结合，是水	好评
17	多民族聚居地，山美、水美、人更美—贵州 贵州位于云贵高原上，地理位置独特	好评
18	来贵州玩自然要来黄果树瀑布，这个景区真的太大了，总共有三个点，中间大巴	好评
19	新冠肺炎疫情在家，2月份都没出过小区，只能找找存货点评一下了…… 从小就	好评
20	黄果树瀑布，即黄果树大瀑布。 古称白水河瀑布，亦名“黄葛墅”瀑布或“黄	好评
21	去年去的，不仅仅会收门票，车票也是要钱的，因为进去还有坐十来分钟大巴，	好评
22	去年夏天到贵州办事，顺便到向往已久的黄果树风景区。景区分为陡坡塘，天星	好评
23	黄果树瀑布位于贵州省安顺市镇宁布依族苗族自治县，古称白水河瀑布，黄果树	好评

(2) 随机森林文本分类

本文不再详细叙述代码实现过程，前面很多文章都介绍过，并且源代码有详细的注释供大家参考。

```
# -*- coding:utf-8 -*-
import csv
import numpy as np
import jieba
```

```
import jieba.analyse
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

#----------------------------------第一步 读取文件--------------------------------
file = "data.csv"

with open(file, "r", encoding="UTF-8") as f:
    # 使用csv.DictReader读取文件中的信息
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        # 数据元素获取
        if row['label'] == '好评':
            res = 0
        else:
            res = 1
        labels.append(res)
        content = row['content']
        seglist = jieba.cut(content,cut_all=False)  #精确模式
        output = ' '.join(list(seglist))            #空格拼接
        #print(output)
        contents.append(output)

print(labels[:5])
print(contents[:5])

#----------------------------------第二步 数据预处理--------------------------------
# 将文本中的词语转换为词频矩阵 矩阵元素a[i][j] 表示j词在i类文本下的词频
vectorizer = CountVectorizer()

# 该类会统计每个词语的tf-idf权值
transformer = TfidfTransformer()

#第一个fit_transform是计算tf-idf 第二个fit_transform是将文本转为词频矩阵
tfidf = transformer.fit_transform(vectorizer.fit_transform(contents))
for n in tfidf[:5]:
    print(n)
#tfidf = tfidf.astype(np.float32)
print(type(tfidf))

# 获取词袋模型中的所有词语
word = vectorizer.get_feature_names()
for n in word[:5]:
    print(n)
print("单词数量:", len(word))

# 将tf-idf矩阵抽取出来，元素w[i][j]表示j词在i类文本中的tf-idf权重
X = tfidf.toarray()
print(X.shape)

# 使用 train_test_split 分割 X y 列表
# X_train矩阵的数目对应 y_train列表的数目(一一对应)  -->> 用来训练模型
# X_test矩阵的数目对应     (一一对应) -->> 用来测试模型的准确性
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=1)
```

```
#---------------------------------第三步 机器学习分类-------------------------------
# 随机森林分类方法模型
# n_estimators: 森林中树的数量
clf = RandomForestClassifier(n_estimators=20)

# 训练模型
clf.fit(X_train, y_train)

# 使用测试值 对 模型的准确度进行计算
print('模型的准确度:{}'.format(clf.score(X_test, y_test)))
print("\n")

# 预测结果
pre = clf.predict(X_test)
print('预测结果:', pre[:10])
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
```

输出结果如下图所示，随机森林的平均准确率为0.86，召回率为0.86，F值也为0.86。

```
(0, 1303)       0.17060573217088465
(0, 1196)       0.27907653808908767
(0, 1096)       0.13580752783150682
(0, 624)        0.21499224630100788
(0, 517)        0.1544254885845423
(0, 479)        0.18344871402168958
(0, 332)        0.10896963025271979
(0, 241)        0.13953826904454383
(0, 84)         0.36689742804337916
(0, 51)         0.10896963025271979
Squeezed text (51 lines).
<class 'scipy.sparse.csr.csr_matrix'>
00
10
100
1000
101
单词数量: 3785
(240, 3785)
模型的准确度:0.8611111111111112

预测结果: [1 0 0 0 0 0 0 0 1 1]
72 72
              precision    recall  f1-score   support

           0       0.88      0.88      0.88        41
           1       0.84      0.84      0.84        31

    accuracy                           0.86        72
   macro avg       0.86      0.86      0.86        72
weighted avg       0.86      0.86      0.86        72
```


2.算法评价

接着作者尝试自定义准确率（Precision）、召回率（Recall）和F特征值（F-measure），其计算公式如下：

$$Precision = \frac{\text{正确被预测的总数}}{\text{预测出的分类总数}}$$

$$Recall = \frac{\text{正确被预测的总数}}{\text{测试集中存在的分类总数}}$$

$$F - measure = \frac{2 * Precision * Recall}{(Precision + Recall)}$$

由于本文主要针对2分类问题，其实验评估主要分为0和1两类，完整代码如下：

```
# -*- coding:utf-8 -*-
import csv
import numpy as np
import jieba
import jieba.analyse
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

#---------------------------------第一步 读取文件--------------------------------
file = "data.csv"

with open(file, "r", encoding="UTF-8") as f:
    # 使用csv.DictReader读取文件中的信息
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        # 数据元素获取
        if row['label'] == '好评':
            res = 0
        else:
            res = 1
        labels.append(res)
        content = row['content']
        seglist = jieba.cut(content,cut_all=False)  #精确模式
        output = ' '.join(list(seglist))            #空格拼接
        #print(output)
        contents.append(output)

print(labels[:5])
print(contents[:5])

#---------------------------------第二步 数据预处理--------------------------------
# 将文本中的词语转换为词频矩阵 矩阵元素a[i][j] 表示j词在i类文本下的词频
vectorizer = CountVectorizer()

# 该类会统计每个词语的tf-idf权值
transformer = TfidfTransformer()

#第一个fit_transform是计算tf-idf 第二个fit_transform是将文本转为词频矩阵
tfidf = transformer.fit_transform(vectorizer.fit_transform(contents))
for n in tfidf[:5]:
    print(n)
#tfidf = tfidf.astype(np.float32)
print(type(tfidf))

# 获取词袋模型中的所有词语
word = vectorizer.get_feature_names()
for n in word[:5]:
    print(n)
print("单词数量:", len(word))

# 将tf-idf矩阵抽取出来，元素w[i][j]表示j词在i类文本中的tf-idf权重
X = tfidf.toarray()
print(X.shape)
```

```
# 使用 train_test_split 分割 X y 列表
# X_train矩阵的数目对应 y_train列表的数目(一一对应)  -->> 用来训练模型
# X_test矩阵的数目对应     (一一对应) -->> 用来测试模型的准确性
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=1)

#---------------------------------第三步 机器学习分类-------------------------------
# 随机森林分类方法模型
# n_estimators：森林中树的数量
clf = RandomForestClassifier(n_estimators=20)

# 训练模型
clf.fit(X_train, y_train)

# 使用测试值 对 模型的准确度进行计算
print('模型的准确度:{}'.format(clf.score(X_test, y_test)))
print("\n")

# 预测结果
pre = clf.predict(X_test)
print('预测结果:', pre[:10])
print(len(pre), len(y_test))
print(classification_report(y_test, pre))

#---------------------------------第四步 评价结果-------------------------------
def classification_pj(name, y_test, pre):
    print("算法评价:", name)

    # 正确率 Precision = 正确识别的个体总数 /  识别出的个体总数
    # 召回率 Recall = 正确识别的个体总数 /  测试集中存在的个体总数
    # F值 F-measure = 正确率 * 召回率 * 2 / (正确率 + 召回率)

    YC_B, YC_G = 0,0  #预测 bad good
    ZQ_B, ZQ_G = 0,0  #正确
    CZ_B, CZ_G = 0,0  #存在

    #0-good 1-bad 同时计算防止类标变化
    i = 0
    while i<len(pre):
        z = int(y_test[i])   #真实
        y = int(pre[i])      #预测

        if z==0:
            CZ_G += 1
        else:
            CZ_B += 1

        if y==0:
            YC_G += 1
        else:
            YC_B += 1

        if z==y and z==0 and y==0:
            ZQ_G += 1
        elif z==y and z==1 and y==1:
            ZQ_B += 1
        i = i + 1

    print(ZQ_B, ZQ_G, YC_B, YC_G, CZ_B, CZ_G)
    print("")

    # 结果输出
```

```
    P_G = ZQ_G * 1.0 / YC_G
    P_B = ZQ_B * 1.0 / YC_B
    print("Precision Good 0:", P_G)
    print("Precision Bad 1:", P_B)

    R_G = ZQ_G * 1.0 / CZ_G
    R_B = ZQ_B * 1.0 / CZ_B
    print("Recall Good 0:", R_G)
    print("Recall Bad 1:", R_B)

    F_G = 2 * P_G * R_G / (P_G + R_G)
    F_B = 2 * P_B * R_B / (P_B + R_B)
    print("F-measure Good 0:", F_G)
    print("F-measure Bad 1:", F_B)

# 函数调用
classification_pj("RandomForest", y_test, pre)
```

输出结果如下图所示，其中好评的准确率、召回率、F值分别为0.9268、0.9268、0.9268，差评的准确率、召回率、F值分别为0.9032、0.9032、0.9032。

```
预测结果: [1 1 0 0 0 0 0 0 1 1]
72 72
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        41
           1       0.90      0.90      0.90        31

    accuracy                           0.92        72
   macro avg       0.92      0.92      0.92        72
weighted avg       0.92      0.92      0.92        72

算法评价: RandomForest
28 38 31 41 31 41

Precision Good 0: 0.926829268292683
Precision Bad 1: 0.9032258064516129
Recall Good 0: 0.926829268292683
Recall Bad 1: 0.9032258064516129
F-measure Good 0: 0.926829268292683
F-measure Bad 1: 0.9032258064516129
```


3.算法对比

最后作者给出机器学习RF、DTC、SVM、KNN、NB、LR的文本分类结果，这也是写论文中很常见的操作。

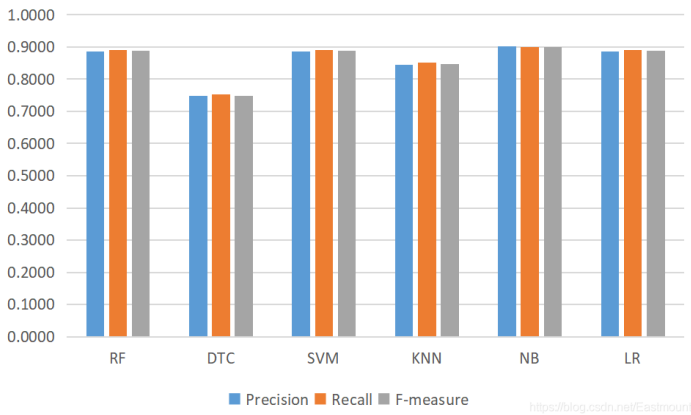
```
# -*- coding:utf-8 -*-
import csv
import numpy as np
import jieba
import jieba.analyse
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import neighbors
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
```

```
#---------------------------------第一步 读取文件-------------------------------
file = "data.csv"

with open(file, "r", encoding="UTF-8") as f:
    # 使用csv.DictReader读取文件中的信息
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        # 数据元素获取
        if row['label'] == '好评':
            res = 0
        else:
            res = 1
        labels.append(res)
        content = row['content']
        seglist = jieba.cut(content,cut_all=False)  #精确模式
        output = ' '.join(list(seglist))            #空格拼接
        #print(output)
        contents.append(output)

print(labels[:5])
print(contents[:5])

#---------------------------------第二步 数据预处理-------------------------------
# 将文本中的词语转换为词频矩阵 矩阵元素a[i][j] 表示j词在i类文本下的词频
vectorizer = CountVectorizer()

# 该类会统计每个词语的tf-idf权值
transformer = TfidfTransformer()

#第一个fit_transform是计算tf-idf 第二个fit_transform是将文本转为词频矩阵
tfidf = transformer.fit_transform(vectorizer.fit_transform(contents))
for n in tfidf[:5]:
    print(n)
#tfidf = tfidf.astype(np.float32)
print(type(tfidf))

# 获取词袋模型中的所有词语
word = vectorizer.get_feature_names()
for n in word[:5]:
    print(n)
print("单词数量:", len(word))

# 将tf-idf矩阵抽取出来，元素w[i][j]表示j词在i类文本中的tf-idf权重
X = tfidf.toarray()
print(X.shape)

# 使用 train_test_split 分割 X y 列表
# X_train矩阵的数目对应 y_train列表的数目(一一对应)  -->> 用来训练模型
# X_test矩阵的数目对应     (一一对应) -->> 用来测试模型的准确性
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=1)

#---------------------------------第四步 评价结果-------------------------------
def classification_pj(name, y_test, pre):
    print("算法评价:", name)

    # 正确率 Precision = 正确识别的个体总数 /  识别出的个体总数
    # 召回率 Recall = 正确识别的个体总数 /  测试集中存在的个体总数
    # F值 F-measure = 正确率 * 召回率 * 2 / (正确率 + 召回率)

    YC_B, YC_G = 0,0  #预测 bad good
```

```
    ZQ_B, ZQ_G = 0,0  #正确
    CZ_B, CZ_G = 0,0  #存在

    #0-good 1-bad 同时计算防止类标变化
    i = 0
    while i<len(pre):
        z = int(y_test[i])   #真实
        y = int(pre[i])      #预测

        if z==0:
            CZ_G += 1
        else:
            CZ_B += 1

        if y==0:
            YC_G += 1
        else:
            YC_B += 1

        if z==y and z==0 and y==0:
            ZQ_G += 1
        elif z==y and z==1 and y==1:
            ZQ_B += 1
        i = i + 1
    print(ZQ_B, ZQ_G, YC_B, YC_G, CZ_B, CZ_G)

    # 结果输出
    P_G = ZQ_G * 1.0 / YC_G
    P_B = ZQ_B * 1.0 / YC_B
    print("Precision Good 0:{:.4f}".format(P_G))
    print("Precision Bad 1:{:.4f}".format(P_B))
    print("Avg_precision:{:.4f}".format((P_G+P_B)/2))

    R_G = ZQ_G * 1.0 / CZ_G
    R_B = ZQ_B * 1.0 / CZ_B
    print("Recall Good 0:{:.4f}".format(R_G))
    print("Recall Bad 1:{:.4f}".format(R_B))
    print("Avg_recall:{:.4f}".format((R_G+R_B)/2))

    F_G = 2 * P_G * R_G / (P_G + R_G)
    F_B = 2 * P_B * R_B / (P_B + R_B)
    print("F-measure Good 0:{:.4f}".format(F_G))
    print("F-measure Bad 1:{:.4f}".format(F_B))
    print("Avg_fmeasure:{:.4f}".format((F_G+F_B)/2))

#---------------------------------第三步 机器学习分类--------------------------------
# 随机森林分类方法模型
rf = RandomForestClassifier(n_estimators=20)
rf.fit(X_train, y_train)
pre = rf.predict(X_test)
print("随机森林分类")
print(classification_report(y_test, pre))
classification_pj("RandomForest", y_test, pre)
print("\n")

# 决策树分类方法模型
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
pre = dtc.predict(X_test)
print("决策树分类")
print(classification_report(y_test, pre))
classification_pj("DecisionTree", y_test, pre)
```

```
print("\n")

# SVM分类方法模型
SVM = svm.LinearSVC() #支持向量机分类器LinearSVC
SVM.fit(X_train, y_train)
pre = SVM.predict(X_test)
print("支持向量机分类")
print(classification_report(y_test, pre))
classification_pj("LinearSVC", y_test, pre)
print("\n")

# KNN分类方法模型
knn = neighbors.KNeighborsClassifier() #n_neighbors=11
knn.fit(X_train, y_train)
pre = knn.predict(X_test)
print("最近邻分类")
print(classification_report(y_test, pre))
classification_pj("KNeighbors", y_test, pre)
print("\n")

# 朴素贝叶斯分类方法模型
nb = MultinomialNB()
nb.fit(X_train, y_train)
pre = nb.predict(X_test)
print("朴素贝叶斯分类")
print(classification_report(y_test, pre))
classification_pj("MultinomialNB", y_test, pre)
print("\n")

# 逻辑回归分类方法模型
LR = LogisticRegression(solver='liblinear')
LR.fit(X_train, y_train)
pre = LR.predict(X_test)
print("逻辑回归分类")
print(classification_report(y_test, pre))
classification_pj("LogisticRegression", y_test, pre)
print("\n")
```

输出结果如下所示，发现贝叶斯算法在文本分类中的效果还是很棒；同时随机森林、逻辑回归、SVM效果都还不错。



完整结果如下：

```
随机森林分类
              precision    recall  f1-score   support

           0       0.92      0.88      0.90        41
           1       0.85      0.90      0.88        31
```

```
    accuracy                           0.89        72
   macro avg       0.89      0.89      0.89        72
weighted avg       0.89      0.89      0.89        72

算法评价: RandomForest
28 36 33 39 31 41
Precision Good 0:0.9231
Precision Bad 1:0.8485
Avg_precision:0.8858
Recall Good 0:0.8780
Recall Bad 1:0.9032
Avg_recall:0.8906
F-measure Good 0:0.9000
F-measure Bad 1:0.8750
Avg_fmeasure:0.8875


决策树分类
              precision    recall  f1-score   support

           0       0.81      0.73      0.77        41
           1       0.69      0.77      0.73        31

    accuracy                           0.75        72
   macro avg       0.75      0.75      0.75        72
weighted avg       0.76      0.75      0.75        72

算法评价: DecisionTree
24 30 35 37 31 41
Precision Good 0:0.8108
Precision Bad 1:0.6857
Avg_precision:0.7483
Recall Good 0:0.7317
Recall Bad 1:0.7742
Avg_recall:0.7530
F-measure Good 0:0.7692
F-measure Bad 1:0.7273
Avg_fmeasure:0.7483


支持向量机分类
最近邻分类
朴素贝叶斯分类
逻辑回归分类
......
```

三.基于CNN的文本分类

接着我们开始通过CNN实现文本分类，该方法可以应用于很多领域，只要有数据集即可分析。这里仅给出最基础且可用的方法及源码，希望对您有所帮助。

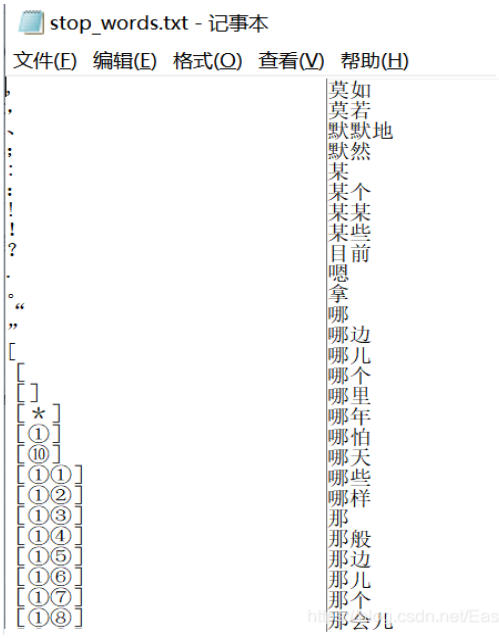
1.数据预处理

上一部分我在写机器学习文本分类时，已经介绍了中文分词等预处理操作，为什么这部分还要介绍呢？因为这里我要增加两个新的操作：

- **去停用词**

- **词性标注**

这两个操作在文本挖掘过程中非常重要，它一方面能提升我们的分类效果，另一方面能过滤掉无关的特征词，词性标注也能辅助我们进行其他的分析，如情感分析、舆情挖掘等。



该部分代码如下：

```
# -*- coding:utf-8 -*-
import csv
import numpy as np
import jieba
import jieba.analyse
import jieba.posseg as pseg
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

#---------------------------------第一步 数据预处理--------------------------------
file = "data.csv"

# 获取停用词
def stopwordslist(): #加载停用词表
    stopwords = [line.strip() for line in open('stop_words.txt', encoding="UTF-8").readlines()]
    return stopwords

# 去除停用词
def deleteStop(sentence):
    stopwords = stopwordslist()
    outstr = ""
    for i in sentence:
        # print(i)
        if i not in stopwords and i!="\n":
            outstr += i
    return outstr

# 中文分词
```

```
Mat = []
with open(file, "r", encoding="UTF-8") as f:
    # 使用csv.DictReader读取文件中的信息
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        # 数据元素获取
        if row['label'] == '好评':
            res = 0
        else:
            res = 1
        labels.append(res)

        # 中文分词
        content = row['content']
        #print(content)
        seglist = jieba.cut(content,cut_all=False)  #精确模式
        #print(seglist)

        # 去停用词
        stc = deleteStop(seglist) #注意此时句子无空格
        # 空格拼接
        seg_list = jieba.cut(stc,cut_all=False)
        output = ' '.join(list(seg_list))
        #print(output)
        contents.append(output)

        # 词性标注
        res = pseg.cut(stc)
        seten = []
        for word,flag in res:
            if flag not in ['nr','ns','nt','mz','m','f','ul','l','r','t']:
                seten.append(word)
        Mat.append(seten)

print(labels[:5])
print(contents[:5])
print(Mat[:5])

# 文件写入
fileDic = open('wordCut.txt', 'w', encoding="UTF-8")
for i in Mat:
    fileDic.write(" ".join(i))
    fileDic.write('\n')
fileDic.close()
words = [line.strip().split(" ") for line in open('WordCut.txt',encoding='UTF-8').readlines()]
print(words[:5])
```

运行结果如下图所示，可以看到原文本被分词，并且过滤掉了“还”、“，”、“常常”等停用词，并且以两种形式呈现，读者可以结合自己的需要进行后续分析。同时，将分词后的文本也写入到wordCut.txt文件中。

- **contents：显示已分词且以列表形式存在的句子**
- **Mat：显示已分词且以列表形式存在的词序列**

```
>>> "还记得小时候，常常守在电视机前，等候《西游记》的播出。“你挑着担,我牵着马。翻山涉水两肩双滑……”"熟悉的歌曲，又在耳边响起时。 这歌词中的水，就有贵州的水，准确的说，是贵州的黄果树瀑布；那一帘瀑布，流进了我们的童年，让我们流连忘返。 黄果树瀑布并不是只有一个瀑布，而是一个大景区，包括陡坡塘瀑布、天星桥景区、黄果树大瀑布，其中黄果树大瀑布是最有名的。收起评论"
```

```
[0, 0, 0, 0, 0]
['记得 小时候 守 电视机 前 等候 西游记 播出 挑担 牵马 翻山 涉水 两肩 双滑 … … ″ 熟悉 歌曲
耳边 响起 时   歌词 中 水 贵州 水 准确 说 贵州 黄果树 瀑布 那一帘 瀑布 流 进 童年 流连忘返
 黄果树 瀑布 瀑布 景区 包括 陡坡 塘 瀑布 天星桥 景区 黄果树 瀑布 黄果树 瀑布 有名 收起 评论
', '飞流直下三千尺   疑是 银河 落九天', '黄果树 瀑布 景区 位置 坐落 贵州省 安顺市 关岭 县
黄果树 风景区 贵州省 著名 景区 中国 最美 瀑布 享有 中华 第一 瀑布 美名 更是 世界 吉尼斯 纪
录 里 认证 世界 瀑布群 说 瀑布群 蕴含着 大小 瀑布 十八个 收起 评论', '国家 重点 风景 名胜区
首批 国家 5A 级 旅游 景区   黄果树 景区 时而 艳阳高照 时而 飘泼大雨 当地人 说 多月 晴朗 天
空 更是 下雨 如此这般 幸运 一会 晒 脸 ？？   一会 挨 浇 ？   ？？   人品 爆棚 景区 风景秀丽 环
境优美 值得 值得 收起 评论', '黄果树 瀑布群 值得一看 景区 黄果树 瀑布 天星桥 景区 错过 天生
桥 下方 珍珠 链 瀑布 值得一看 瀑布 黄果树 观景 距离 近多 触摸 水花 水汽 白链 如珠 水流 急
特别 壮观 实际上 危险 游玩 过后 星期 母子 瀑布 身亡 实际上 黄果树 瀑布 栏杆 外远观 无视 栏
杆 走近 水边 急 水流 危险 可想而知 收起 评论']
[['记得', '小时候', '守', '电视机', '等候', '西游记', '播出', '挑担', '牵马', '翻山', '涉水
', '两肩', '双滑', '…', '…', '″', '熟悉', '歌曲', '耳边', '响起', '时', '', '歌词', '水'
, '水', '准确', '说', '瀑布', '帘', '瀑布', '流', '进', '', '瀑布', '瀑布', '景区', '包括'
, '塘', '瀑布', '景区', '瀑布', '瀑布', '有名', '收起', '评论'], ['飞流直下三千尺', '', '',
疑', '是', '银河', '落九天'], ['瀑布', '景区', '位置', '坐落', '县', '风景区', '著名', '景
区', '最美', '瀑布', '享有', '中华', '瀑布', '更是', '世界', '纪录', '认证', '世界', '瀑布
群', '说', '瀑布群', '蕴含着', '大小', '瀑布', '收起', '评论'], ['国家', '重点', '风景',
名胜区', '国家', 'A', '级', '旅游', '景区', '', '景区', '时而', '艳阳高照', '时而', '飘泼
大雨', '当地人', '说', '晴朗', '天空', '更是', '下雨', '幸运', '晒', '脸', '?', '?', '',
挨', '浇', '?', '', '?', '?', '', '人品', '爆棚', '景区', '风景秀丽', '', '环境优美', '', '值得',
'值得', '收起', '评论'], ['瀑布群', '景区', '瀑布', '景区', '错过', '天生桥', '珍珠', '链',
'瀑布', '瀑布', '观景', '距离', '近', '触摸', '水花', '水汽', '白链', '如', '珠', '水流',
急', '特别', '壮观', '实际上', '危险', '游玩', '母子', '瀑布', '身亡', '实际上', '瀑布',
栏杆', '外远观', '无视', '栏杆', '走近', '水边', '急', '水流', '危险', '收起', '评论']]
```

2.特征提取及Word2Vec词向量转换

(1) 特征词编号

首先，我们先调用Tokenizer和fit_on_texts函数将文本中的每个词编号，词频出现越高其编号越小。如下图所示，“瀑布”、“景区”、“排队”、“水帘洞”等特征词出现较多，注意空格、“评论”、“收起”可以继续过滤掉，在停用词表中添加即可。

```
#fit_on_texts函数可以将输入的文本每个词编号 编号根据词频(词频越大编号越小)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(Mat)
vocab = tokenizer.word_index  #停用词已过滤,获取每个词的编号
print(vocab)
```

输出结果如下图所示：

```
{' ': 1, '瀑布': 2, '景区': 3, '评论': 4, '收起': 5, '景点': 6, '?': 7, '排队': 8, '水
帘洞': 9, '人': 10, '小时': 11, '走': 12, '真的': 13, '…': 14, '壮观': 15, '塘': 16,
'时间': 17, '太': 18, '说': 19, '没': 20, '游客': 21, '体验': 22, '门票': 23, '建议':
24, '管理': 25, '坐': 26, '西游记': 27, '想': 28, '风景': 29, '地方': 30, '世界': 31,
'车': 32, '观光车': 33, '差': 34, '特别': 35, '大巴': 36, '旅游': 37, '是': 38, 'a':
39, '景色': 40, '水': 41, '玩': 42, '工作人员': 43, '很大': 44, '买': 45, '带': 46,
'（': 47, '）': 48, '高': 49, '票': 50, '扶梯': 51, '值得': 52, '交通': 53, '很': 54,
'买票': 55, '大巴车': 56, '宽': 57, '挤': 58, '著名': 59, '瀑布群': 60, '大': 61, '水
量': 62, '时': 63, '进': 64, '推荐': 65, '位于': 66, '挺': 67, '3': 68, '水流': 69,
'排': 70, '更': 71, '1': 72, '老人': 73, '来到': 74, '感受': 75, '只能': 76, '路':
77, '震撼': 78, '超级': 79, '导游': 80, '站': 81, '长': 82, '不错': 83, '到': 84, '实
在': 85, '看': 86, '拍照': 87, '网上': 88, '不': 89, '去': 90, '选择': 91, '雨衣':
92, '旺季': 93, '2': 94, '观光': 95, '不想': 96, '走路': 97, '花': 98, '游玩': 99,
```

(2) Word2Vec词向量训练

获取了特征词编号即将特征矩阵的表头定义好了，接下来我们需要将每一行文本转换成一维词向量，最终构建特征矩阵，用于训练和分类。注意，利用pad_sequences方法将CNN训练的长度统一，更好地进行训练。比如设置为100，如果句子超过100后面的单词会被切掉；如果句子未超过100，则会在句子前面补0，下图展示了补0过程。同时，分类结果[0,1]表示类标是好评0，[1,0]表示类标是差评1。

```
[[   0    0    0 ... 2481    5    4]
 [   0    0    0 ...  570   52   90]
 [   0    0    0 ...  187    5    4]
 ...
 [   0    0    0 ...   93    5    4]
 [   0    0    0 ...   30    5    4]
 [   0    0    0 ...   81   18   78]]
[[0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
```

此时的完整代码如下：

```
# 使用 train_test_split 分割 X y 列表
X_train, X_test, y_train, y_test = train_test_split(Mat, labels, test_size=0.3, random_state=1)
print(X_train[:5])
print(y_train[:5])

#----------------------------------第三步 词向量构建-------------------------------
# Word2Vec训练
maxLen = 100                  #词序列最大长度
num_features = 100            #设置词语向量维度
min_word_count = 3            #保证被考虑词语的最低频度
num_workers = 4               #设置并行化训练使用CPU计算核心数量
context = 4                   #设置词语上下文窗口大小

# 设置模型
model = word2vec.Word2Vec(Mat, workers=num_workers, size=num_features,
                          min_count=min_word_count,window=context)
# 强制单位归一化
model.init_sims(replace=True)
# 输入一个路径保存训练模型 其中./data/model目录事先存在
model.save("CNNw2vModel")
model.wv.save_word2vec_format("CNNVector",binary=False)
print(model)
# 加载模型 如果word2vec已训练好直接用下面语句
w2v_model = word2vec.Word2Vec.load("CNNw2vModel")

# 特征编号(不足的前面补0)
trainID = tokenizer.texts_to_sequences(X_train)
print(trainID)
testID = tokenizer.texts_to_sequences(X_test)
print(testID)
# 该方法会让CNN训练的长度统一
trainSeq = pad_sequences(trainID, maxlen=maxLen)
print(trainSeq)

# 标签独热编码 转换为one-hot编码
trainCate = to_categorical(y_train, num_classes=2) #二分类问题
print(trainCate)
testCate = to_categorical(y_test, num_classes=2)  #二分类问题
print(testCate)
```

输出结果如下：

```
[['景色', ' ', '景区', '太', '成熟', '从', '大', '瀑布', '景区', '出发', '景区', '观光车', '足足', '游客', '半小时
', '世博会', '路上', '摩肩接踵', '欣赏', '美景', '心情', '观光车', '上车', '处', '标明', '目的地', '入口处', '引导',
'走', '冤枉路', '稀里糊涂', '上车', '问', '司机', '到达', '司机', '含糊地', '说', '开出', '景区', '客运站', '七孔',
'景区', '开发', '完美', '收起', '评论'],
['淡季', '瀑布', '人', '少', '景美', '机票', '便宜', '值得', '去'],
['瀑布', '体验', '差', '五星', '好评', '全', '是', '刷', '道路', '很窄', '导致', '大面积', '堵塞', '排队', '崩溃',
'景区', '指引', '清晰', '排队', '大雨', '遮雨', '设计', '搞', '大人', '小孩', '老人', '淋雨', '景区', '接待', '能力
差', '瀑布', '真的', '徒有虚名', '七孔', '收起', '评论'],
['老爸', '分', '瀑布', '瀑布', '瀑布', '游览', '瀑布', '门票', '反正', '超过', ' ', '来到', '熟悉', '告知', '只能
', '出', '进入', '口', '回到', '高速', '出口', '直行', '回去', '倒', '指示', '清晰', '隔离', '栏杆', '自驾车', '导
进', '停车场', '停车场', '收费', '且', '时间', ' ', '停车场', '经查', '景区', '门票', '单人', '含', '交通', '车费',
'交通车', '需', '另付', '从外', '围绕', '路', '花', '不到', '分钟', '车费', '真心', '接受', ' ', '全家人', '不想',
'ㄱ', '（', '—', '__', '-', '）', 'ㄱ', '利益', '勾结', '剧烈', '涨费', '个金', '瀑布', '好看', '差', '评', ' ',
'图片', '未', '开发', '瀑布', '天坑', '瀑布', '壮观', '壮观', '有', '灵秀', '景区', '膨胀', '成', '收起', '评论'],
['全家', '票', '居民', '专享', '优惠', '票']]

[1, 0, 1, 1, 1]
Word2Vec(vocab=718, size=100, alpha=0.025)
```

```
[[   0    0    0 ... 2481    5    4]
 [   0    0    0 ...  570   52   90]
 [   0    0    0 ...  187    5    4]
 ...
 [   0    0    0 ...   93    5    4]
 [   0    0    0 ...   30    5    4]
 [   0    0    0 ...   81   18   78]]

 [[0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
```

3.CNN构建

接下来我们开始将构建好的特征矩阵拿去训练，计算不同文本或一维矩阵的相似度，这样会将好评和差评的不同句子按相似度分成两类。这里同样使用Word2Vec实现核心代码如下：

```
model = word2vec.Word2Vec(
    Mat,
    workers=num_workers,
    size=num_features,
    min_count=min_word_count,
    window=context
);
```

训练模型的结果为“Word2Vec(vocab=718, size=100, alpha=0.025)”，这里设置的过滤频度为3，相当于出现频率低于3的被过滤，最终得到718个特征词。num_features值为100，表示是100维的词向量。sg默认为连续词袋模型，也可以设置为1跳字模型。默认的优化方法负采样，更多参数解释请读者百度。

参考作者前文：

- 九.gensim词向量Word2Vec安装及《庆余年》中文短文本相似度计算
- word2vec词向量训练及中文文本相似度计算

如果我们存在一个训练集、一个测试集，如果测试集中不存在某个特征词，怎么解决呢？这里我们在获取某个特征词的词向量，并转换为训练矩阵时，使用了try-except异常捕获，如果未找到特征词则跳过即可，它会自动补0。

```
embedding_matrix = np.zeros((len(vocab)+1, 100))
for word, i in vocab.items():
    try:
        #提取词向量并放置训练矩阵
        embedding_vector = w2v_model[str(word)]
        embedding_matrix[i] ] = embedding_vector
    except KeyError: #单词未找到跳过
        continue
```


该部分代码如下所示：

```
#---------------------------------第四步 CNN构建---------------------------------
# 利用训练后的Word2vec自定义Embedding的训练矩阵 每行代表一个词(结合独热编码和矩阵乘法理解)
```

```
embedding_matrix = np.zeros((len(vocab)+1, 100)) #从0开始计数 加1对应之前特征词
for word, i in vocab.items():
    try:
        #提取词向量并放置训练矩阵
        embedding_vector = w2v_model[str(word)]
        embedding_matrix[i] = embedding_vector
    except KeyError: #单词未找到跳过
        continue

# 训练模型
main_input = Input(shape=(maxLen,), dtype='float64')
# 词嵌入 使用预训练Word2Vec的词向量 自定义权重矩阵 100是输出词向量维度
embedder = Embedding(len(vocab)+1, 100, input_length=maxLen,
                     weights=[embedding_matrix], trainable=False) #不再训练
# 建立模型
model = Sequential()
model.add(embedder)  #构建Embedding层
model.add(Conv1D(256, 3, padding='same', activation='relu')) #卷积层步幅3
model.add(MaxPool1D(maxLen-5, 3, padding='same'))            #池化层
model.add(Conv1D(32, 3, padding='same', activation='relu'))  #卷积层
model.add(Flatten())                                         #拉直化
model.add(Dropout(0.3))                                      #防止过拟合 30%不训练
model.add(Dense(256, activation='relu'))                     #全连接层
model.add(Dropout(0.2))                                      #防止过拟合
model.add(Dense(units=2, activation='softmax'))              #输出层

# 模型可视化
model.summary()

# 激活神经网络
model.compile(optimizer = 'adam',                    #优化器
              loss = 'categorical_crossentropy',     #损失
              metrics = ['accuracy']                 #计算误差或准确率
              )

#训练(训练数据、训练类标、batch_size每次256条训练、epochs、随机选择、验证集20%)
history = model.fit(trainSeq, trainCate, batch_size=256,
                    epochs=6, validation_split=0.2)
model.save("TextCNN")

#----------------------------------第五步 预测模型-------------------------------
# 预测与评估
mainModel = load_model("TextCNN")
result = mainModel.predict(testSeq) #测试样本
#print(result)
print(np.argmax(result,axis=1))
score = mainModel.evaluate(testSeq,
                           testCate,
                           batch_size=32)
print(score)
```

构建的模型如下：

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 100, 100)          290400
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 100, 256)          77056
_________________________________________________________________
```

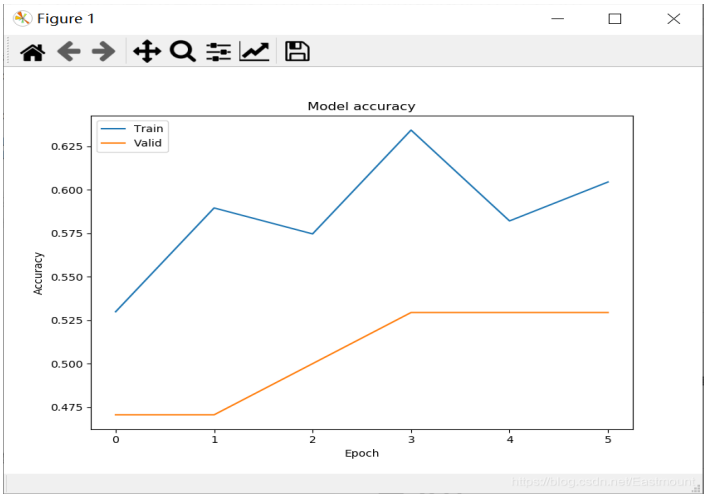
```
max_pooling1d_1 (MaxPooling1 (None, 34, 256)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 34, 32)            24608
_________________________________________________________________
flatten_1 (Flatten)          (None, 1088)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1088)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               278784
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 671,362
Trainable params: 380,962
Non-trainable params: 290,400
```

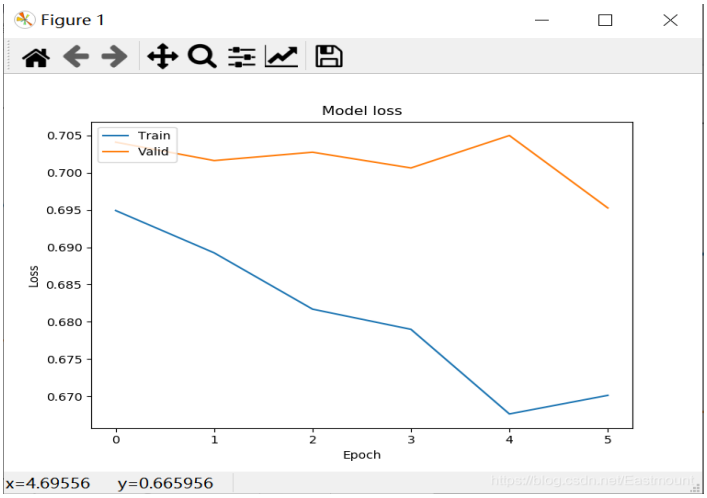
输出结果如下图所示，该模型的预测结果不是很理想，accuracy值仅为0.625，为什么呢？作者也还在进一步研究深度模型的优化，本文更重要的是提供一种可用的方法，效果不好也请见谅~

```
Train on 134 samples, validate on 34 samples
Epoch 1/6
134/134 [==============================] - 1s 7ms/step - loss: 0.6961 - accuracy: 0.4925 -
val_loss: 0.7194 - val_accuracy: 0.4412
Epoch 2/6
134/134 [==============================] - 0s 1ms/step - loss: 0.6955 - accuracy: 0.5299 -
val_loss: 0.7073 - val_accuracy: 0.4412
Epoch 3/6
134/134 [==============================] - 0s 1ms/step - loss: 0.6929 - accuracy: 0.5000 -
val_loss: 0.6990 - val_accuracy: 0.4118
Epoch 4/6
134/134 [==============================] - 0s 1ms/step - loss: 0.6835 - accuracy: 0.5896 -
val_loss: 0.6976 - val_accuracy: 0.4706
Epoch 5/6
134/134 [==============================] - 0s 1ms/step - loss: 0.6891 - accuracy: 0.5075 -
val_loss: 0.6980 - val_accuracy: 0.5000
Epoch 6/6
134/134 [==============================] - 0s 1ms/step - loss: 0.6777 - accuracy: 0.5746 -
val_loss: 0.7011 - val_accuracy: 0.5000
[0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0
 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0]
72/72 [==============================] - 0s 3ms/step
[0.6678964628113641, 0.625]
```


4.测试可视化

最后增加可视化代码，绘制图形如下图所示。再次强调，该算法效果确实不理想，误差不是逐渐递减，正确率也不是不断升高。如果读者发现原因或优化方法也恳请您告知，谢谢。





最后附上完整代码：

```
# -*- coding:utf-8 -*-
import csv
import numpy as np
import jieba
import jieba.analyse
import jieba.posseg as pseg
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras import models
from keras import layers
from keras import Input
from gensim.models import word2vec
from keras.preprocessing.text import Tokenizer
from keras.utils.np_utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Flatten, Dense, Dropout, Conv1D, MaxPool1D, Embedding

#---------------------------------第一步 数据预处理--------------------------------
file = "data.csv"

# 获取停用词
def stopwordslist(): #加载停用词表
    stopwords = [line.strip() for line in open('stop_words.txt', encoding="UTF-8").readlines()]
    return stopwords

# 去除停用词
def deleteStop(sentence):
    stopwords = stopwordslist()
    outstr = ""
    for i in sentence:
        # print(i)
        if i not in stopwords and i!="\n":
            outstr += i
    return outstr

# 中文分词
```

```
Mat = []
with open(file, "r", encoding="UTF-8") as f:
    # 使用csv.DictReader读取文件中的信息
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        # 数据元素获取
        if row['label'] == '好评':
            res = 0
        else:
            res = 1
        labels.append(res)

        # 中文分词
        content = row['content']
        #print(content)
        seglist = jieba.cut(content,cut_all=False)  #精确模式
        #print(seglist)

        # 去停用词
        stc = deleteStop(seglist) #注意此时句子无空格
        # 空格拼接
        seg_list = jieba.cut(stc,cut_all=False)
        output = ' '.join(list(seg_list))
        #print(output)
        contents.append(output)

        # 词性标注
        res = pseg.cut(stc)
        seten = []
        for word,flag in res:
            if flag not in ['nr','ns','nt','mz','m','f','ul','l','r','t']:
                #print(word,flag)
                seten.append(word)
        Mat.append(seten)

print(labels[:5])
print(contents[:5])
print(Mat[:5])

#----------------------------------第二步 特征编号--------------------------------
# fit_on_texts函数可以将输入的文本每个词编号 编号根据词频(词频越大编号越小)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(Mat)
vocab = tokenizer.word_index  #停用词已过滤,获取每个词的编号
print(vocab)

# 使用 train_test_split 分割 X y 列表
X_train, X_test, y_train, y_test = train_test_split(Mat, labels, test_size=0.3, random_state=1)
print(X_train[:5])
print(y_train[:5])

#----------------------------------第三步 词向量构建--------------------------------
# Word2Vec训练
maxLen = 100                #词序列最大长度
num_features = 100          #设置词语向量维度
min_word_count = 3          #保证被考虑词语的最低频度
num_workers = 4             #设置并行化训练使用CPU计算核心数量
context = 4                 #设置词语上下文窗口大小

# 设置模型
```

```
model = word2vec.Word2Vec(Mat, workers=num_workers, size=num_features,
                          min_count=min_word_count,window=context)
# 强制单位归一化
model.init_sims(replace=True)
# 输入一个路径保存训练模型 其中./data/model目录事先存在
model.save("CNNw2vModel")
model.wv.save_word2vec_format("CNNVector",binary=False)
print(model)
# 加载模型 如果word2vec已训练好直接用下面语句
w2v_model = word2vec.Word2Vec.load("CNNw2vModel")

# 特征编号(不足的前面补0)
trainID = tokenizer.texts_to_sequences(X_train)
print(trainID)
testID = tokenizer.texts_to_sequences(X_test)
print(testID)
# 该方法会让CNN训练的长度统一
trainSeq = pad_sequences(trainID, maxlen=maxLen)
print(trainSeq)
testSeq = pad_sequences(testID, maxlen=maxLen)
print(testSeq)

# 标签独热编码 转换为one-hot编码
trainCate = to_categorical(y_train, num_classes=2) #二分类问题
print(trainCate)
testCate = to_categorical(y_test, num_classes=2)  #二分类问题
print(testCate)

#---------------------------------第四步 CNN构建--------------------------------
# 利用训练后的Word2vec自定义Embedding的训练矩阵 每行代表一个词(结合独热编码和矩阵乘法理解)
embedding_matrix = np.zeros((len(vocab)+1, 100)) #从0开始计数 加1对应之前特征词
for word, i in vocab.items():
    try:
        #提取词向量并放置训练矩阵
        embedding_vector = w2v_model[str(word)]
        embedding_matrix[i] = embedding_vector
    except KeyError: #单词未找到跳过
        continue

# 训练模型
main_input = Input(shape=(maxLen,), dtype='float64')
# 词嵌入 使用预训练Word2Vec的词向量 自定义权重矩阵 100是输出词向量维度
embedder = Embedding(len(vocab)+1, 100, input_length=maxLen,
                     weights=[embedding_matrix], trainable=False) #不再训练
# 建立模型
model = Sequential()
model.add(embedder)  #构建Embedding层
model.add(Conv1D(256, 3, padding='same', activation='relu')) #卷积层步幅3
model.add(MaxPool1D(maxLen-5, 3, padding='same'))            #池化层
model.add(Conv1D(32, 3, padding='same', activation='relu'))  #卷积层
model.add(Flatten())                                         #拉直化
model.add(Dropout(0.3))                                      #防止过拟合 30%不训练
model.add(Dense(256, activation='relu'))                     #全连接层
model.add(Dropout(0.2))                                      #防止过拟合
model.add(Dense(units=2, activation='softmax'))              #输出层

# 模型可视化
model.summary()

# 激活神经网络
model.compile(optimizer = 'adam',                    #优化器
              loss = 'categorical_crossentropy',     #损失
```

```
                metrics = ['accuracy']                  #计算误差或准确率
                )

    #训练(训练数据、训练类标、batch_size每次256条训练、epochs、随机选择、验证集20%)
    history = model.fit(trainSeq, trainCate, batch_size=256,
                        epochs=6, validation_split=0.2)
    model.save("TextCNN")

    #---------------------------------第五步 预测模型-------------------------------
    # 预测与评估
    mainModel = load_model("TextCNN")
    result = mainModel.predict(testSeq) #测试样本
    print(result)
    print(np.argmax(result,axis=1))
    score = mainModel.evaluate(testSeq,
                               testCate,
                               batch_size=32)
    print(score)

    #---------------------------------第五步 可视化-------------------------------
    import matplotlib.pyplot as plt
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train','Valid'], loc='upper left')

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train','Valid'], loc='upper left')
    plt.show()
```

四.总结

写道这里，这篇文章就结束了。希望对您有所帮助，同时文章中不足或错误的地方，欢迎读者提出。这些实验都是我在做论文研究或项目评价常见的一些问题，希望读者带着这些问题，结合自己的需求进行深入的思考，更希望大家能学以致用。最后如果文章对您有帮助，请点赞、评论、收藏，这将是我分享最大的动力。

总之，本文通过Keras实现了一个CNN文本分类学习的案例，并详细介绍了文本分类原理知识及与机器学习对比。最后，作为人工智能的菜鸟，我希望自己能不断进步并深入，后续将它应用于图像识别、网络安全、对抗样本等领域，指导大家撰写简单的学术论文，一起加油！感谢这些年遇到很多以前进步的博友，共勉~

(By:Eastmount 2020-06-29 晚上9点夜于贵阳 http://blog.csdn.net/eastmount/)

2020年8月18新开的“娜璋AI安全之家”，主要围绕Python大数据分析、网络空间安全、人工智能、Web渗透及攻防技术进行讲解，同时分享CCF、SCI、南核北核论文的算法实现。娜璋之家会更加系统，并重构作者的所有文章，从零讲解Python和安全，写了近十年文章，真心想把自己所学所感所做分享出来，还请各位多多指教，真诚邀请您的关注！谢谢。