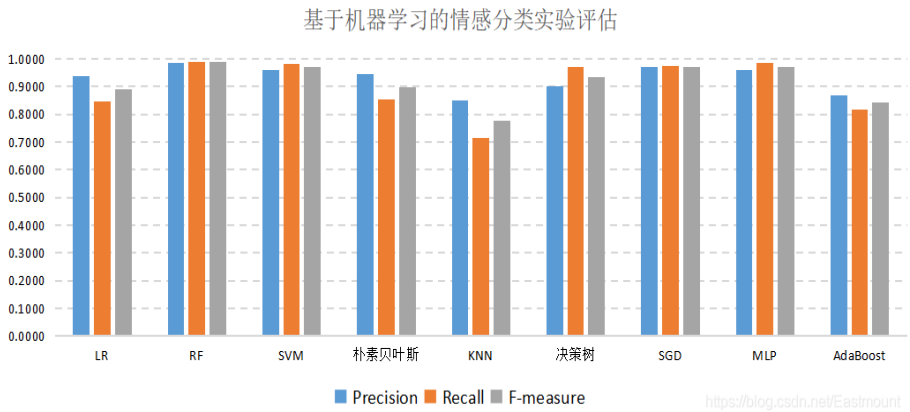
[Python人工智能] 二十三.基于机器学习和TFIDF的情感分类（含详细的NLP数据清洗）

原创 Eastmount 2020-08-17 14:38:30 13682 收藏 252 编辑 版权

分类专栏： Python+TensorFlow人工智能 文章标签： Python人工智能 情感分析 情感分类 机器学习 文本挖掘

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。前一篇文章分享了自定义情感词典（大连理工词典）实现情感分析和情绪分类的过程。这篇文章将详细讲解自然语言处理过程，基于机器学习和TFIDF的情感分类算法，并进行了各种分类算法（SVM、RF、LR、Boosting）对比。这篇文章主要结合作者的书籍《Python网络数据爬取及分析从入门到精通（分析篇）》进行讲解，再次带领大家好好看看Python中文文本分析的基本步骤。个人感觉还不错，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
情感分析地址：https://github.com/eastmountyxz/Sentiment-Analysis

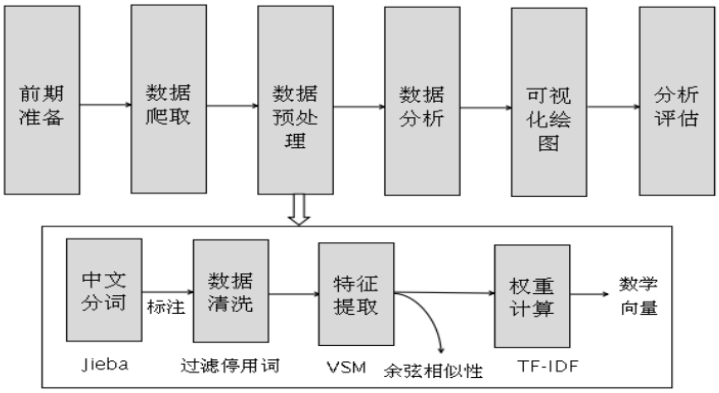
文章目录

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
[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
[Python人工智能] 二十二.基于大连理工情感词典的情感分析和情绪计算

在数据分析和数据挖掘中，通常需要经历前期准备、数据爬取、数据预处理、数据分析、数据可视化、评估分析等步骤，而数据分析之前的工作几乎要花费数据工程师近一半的工作时间，其中的数据预处理也将直接影响后续模型分析的好坏。图是数据预处理的基本步骤，包括中文分词、词性标注、数据清洗、特征提取（向量空间模型存储）、权重计算（TF-IDF）等。



一.中文分词

当读者使用Python爬取了中文数据集之后，首先需要对数据集进行中文分词处理。由于英文中的词与词之间是采用空格关联的，按照空格可以直接划分词组，所以不需要进行分词处理，而中文汉字之间是紧密相连的，并且存在语义，词与词之间没有明显的分隔点，所以需要借助中文分词技术将语料中的句子按空格分割，变成一段段词序列。下面开始详细介绍中文分词技术及Jiaba中文分词工具。

中文分词（Chinese Word Segmentation）指将汉字序列切分成一个个单独的词或词串序列，它能够在没有词边界的中文字符串中建立分隔标志，通常采用空格分隔。下面举个简单示例，对句子“我是程序员”进行分词操作。

```
输入：我是程序员
输出1：我\是\程\序\员
输出2：我是\是程\程序\序员
输出3：我\是\程序员
```

简单举个例子，代码中主要导入Jieba扩展包，然后调用其函数进行中文分词。

```
#encoding=utf-8
import jieba

text = "北京理工大学生前来应聘"

data = jieba.cut(text,cut_all=True)   #全模式
print("[全模式]: ", " ".join(data))

data = jieba.cut(text,cut_all=False)  #精确模式
print("[精确模式]: ", " ".join(data))

data = jieba.cut(text)  #默认是精确模式
print("[默认模式]: ", " ".join(data))

data = jieba.cut_for_search(text)  #搜索引擎模式
print("[搜索引擎模式]: ", " ".join(data))
```

上述代码输出如下，包括全模式、精确模式和搜索引擎模式输出的结果。

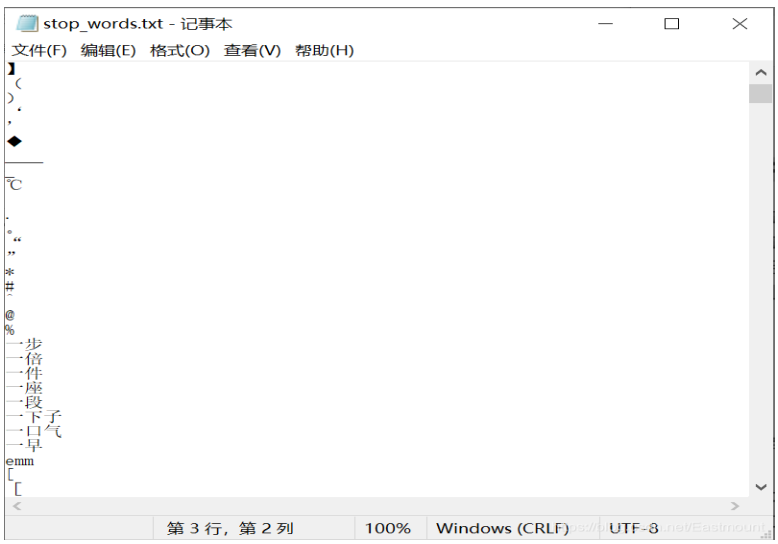
```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\xiuzhang\AppData\Local\Temp\jieba.cache
Loading model cost 0.987 seconds.
Prefix dict has been built succesfully.
[全模式]: 北京 北京理工 北京理工大学 理工 理工大 理工大学 工大 大学 大学生 学生 生前 前来 应聘
[精确模式]: 北京理工大学 生前 来 应聘
[默认模式]: 北京理工大学 生前 来 应聘
[搜索引擎模式]: 北京 理工 工大 大学 理工大 北京理工大学 生前 来 应聘
>>>
```

二.数据清洗

在分析语料的过程中，通常会存在一些脏数据或噪声词组干扰我们的实验结果，这就需要对分词后的语料进行数据清洗（Data Cleaning）。比如前面使用Jieba工具进行中文分词，它可能存在一些脏数据或停用词，如“我们”、“的”、“吗”等。这些词降低了数据质量，为了得到更好的分析结果，需要对数据集进行数据清洗或停用词过滤等操作。

- 残缺数据
- 重复数据
- 错误数据
- 停用词

这里主要讲解停用词过滤，将这些出现频率高却不影响文本主题的停用词删除。在Jieb分词过程中引入stop_words.txt停用词词典，如果存在则过滤即可。



下面是从大众点评、美团之类的网站抓取“黄果树瀑布”的评论信息，我们通过Jieba工具对其进行中文分词。

- 好评：5000条
- 差评：1000条

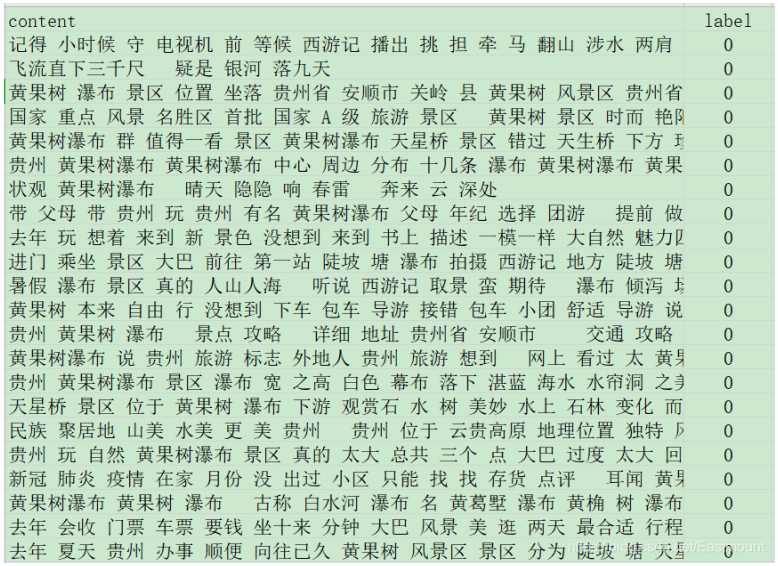
spot	content	label
黄果树	还记得小时候，常常守在电视机前，等候《西游记》的播出。“你挑着	好评
黄果树	飞流直下三千尺 疑是银河落九天！	好评
黄果树	黄果树大瀑布景区，位置坐落在贵州省安顺市关岭县黄果树风景区。这	好评
黄果树	国家重点风景名胜区和首批国家A级旅游景区 今天黄果树景区时而艳阳	好评
黄果树	黄果树瀑布群绝对是值得一看的。景区非常大，不止黄果树瀑布，天星	好评
黄果树	到了贵州，必须要到的就是黄果树瀑布！以黄果树瀑布为中心，周边分	好评
黄果树	状观的黄果树瀑布 晴天隐隐响春雷， 绝顶奔来云深处。	好评
黄果树	带父母带贵州玩，那一定要来贵州最有名的黄果树瀑布看看。父母年纪	好评
黄果树	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	好评
黄果树	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	好评
黄果树	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3	好评
黄果树	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	好评
黄果树	贵州黄果树大瀑布 ？【景点攻略】 ??详细地址：贵州省安顺市 ？交通	好评
黄果树	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	好评
黄果树	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	好评
黄果树	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	好评
黄果树	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	好评
黄果树	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，	好评
黄果树	新冠肺炎疫情在家，月份都没出过小区，只能找找存货点评一下了……	好评
黄果树	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	好评
黄果树	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	好评
黄果树	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	好评

完整代码：

```
# -*- coding:utf-8 -*-
import csv
import pandas as pd
import numpy as np
import jieba
import jieba.analyse

#添加自定义词典和停用词典
jieba.load_userdict("user_dict.txt")
stop_list = pd.read_csv('stop_words.txt',
                        engine='python',
                        encoding='utf-8',
                        delimiter="\n",
                        names=['t'])['t'].tolist()

#中文分词函数
def txt_cut(juzi):
    return [w for w in jieba.lcut(juzi) if w not in stop_list]

#写入分词结果
fw = open('fenci_data.csv', "a+", newline = '',encoding = 'gb18030')
writer = csv.writer(fw)
writer.writerow(['content','label'])

# 使用csv.DictReader读取文件中的信息
labels = []
contents = []
file = "data.csv"
with open(file, "r", encoding="UTF-8") as f:
    reader = csv.DictReader(f)
    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 = txt_cut(content)
        output = ' '.join(list(seglist))            #空格拼接
        contents.append(output)

        #文件写入
```

```
            tlist = []
            tlist.append(output)
            tlist.append(res)
            writer.writerow(tlist)

    print(labels[:5])
    print(contents[:5])
    fw.close()
```

运行结果如下图所示，一方面它将特殊标点符号、停用词过滤，另一方面导入了user_dict.txt词典，将“黄果树瀑布”、“风景区”等专有名词分词，否则它可能会划分为“黄果树”和“瀑布”、“风景”和“区”。



content	label
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	0
飞流直下三千尺 疑是 银河 落九天	0
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	0
国家 重点 风景 名胜区 首批 国家 A 级 旅游 景区 黄果树 景区 时而 艳阳	0
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	0
贵州 黄果树瀑布 黄果树瀑布 中心 周边 分布 十几条 瀑布 黄果树瀑布 黄果	0
状观 黄果树瀑布 晴天 隐隐 响 春雷 奔来 云 深处	0
带 父母 带 贵州 玩 贵州 有名 黄果树瀑布 父母 年纪 选择 团游 提前 做	0
去年 玩 想着 来到 新 景色 没想到 来到 书上 描述 一模一样 大自然 魅力四	0
进门 乘坐 景区 大巴 前往 第一站 陡坡 塘 瀑布 拍摄 西游记 地方 陡坡 塘	0
暑假 瀑布 景区 真的 人山人海 听说 西游记 取景 蛮 期待 瀑布 倾泻	0
黄果树 本来 自由 行 没想到 下车 包车 导游 接错 包车 小团 舒适 导游 说	0
贵州 黄果树 瀑布 景点 攻略 详细 地址 贵州省 安顺市 交通 攻略	0
黄果树瀑布 说 贵州 旅游 标志 外地人 贵州 旅游 想到 网上 看过 太 黄果	0
贵州 黄果树瀑布 景区 瀑布 宽 之高 白色 幕布 落下 湛蓝 海水 水帘洞 之美	0
天星桥 景区 位于 黄果树 瀑布 下游 观赏石 水 树 美妙 水上 石林 变化 而	0
民族 聚居地 山美 水美 更 美 贵州 贵州 位于 云贵高原 地理位置 独特	0
贵州 玩 自然 黄果树瀑布 景区 真的 太大 总共 三个 点 大巴 过度 太大 回	0
新冠 肺炎 疫情 在家 月份 没 出过 小区 只能 找 找 存货 点评 耳闻 黄果	0
黄果树瀑布 黄果树 瀑布 古称 白水河 瀑布 名 黄葛墅 瀑布 黄桷 树 瀑布	0
去年 会收 门票 车票 要钱 坐十来 分钟 大巴 风景 美 逛 两天 最合适 行程	0
去年 夏天 贵州 办事 顺便 向往已久 黄果树 风景区 景区 分为 陡坡 塘 [illegible]	0

- 数据清洗前

> 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。

- 数据清洗后

> 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

三.特征提取及TF-IDF计算

1.基本概念

权重计算是指通过特征权重来衡量特征项在文档表示中的重要程度，给特征词赋予一定的权重来衡量统计文本特征词。TF-IDF（Term Frequency-Invers Document Frequency）是近年来用于数据分析和信息处理经典的权重计算技术。该技术根据特征词在文本中出现的次数和在整个语料中出现的文档频率来计算该特征词在整个语料中的重要程度，其优点是能过滤掉一些常见却无关紧要的词语，尽可能多的保留影响程度高的特征词。

TF-IDF的计算公式如下，式中TF-IDF表示词频TF和倒文本词频IDF的乘积，TF-IDF中权重与特征项在文档中出现的频率成正比，与在整

个语料中出现该特征项的文档数成反比。TF-IDF值越大则该特征词对这个文本的重要程度越高。

$$tfidf_{i,j} = tf_{i,j} \times idf_{i,j}$$

其中，TF词频的计算公式如下，$n_{i,j}$ 为特征词 t_i 在训练文本 D_j 中出现的次数，分母是文本 D_j 中所有特征词的个数，计算的结果即为某个特征词的词频。

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

倒文档频率（Inverse Document Frequency，简称IDF）是Spark Jones在1972年提出的，用于计算词与文献相关权重的经典方法。计算公式如下，参数|D|表示语料的文本总数，$|D_t|$ 表示文本所包含特征词 t_j 的数量。

$$idf_{i,j} = log_{10} \frac{|D|}{1 + |D_{t_i}|}$$

在倒文档频率方法中，权重是随着特征词的文档数量的变化呈反向变化。如某些常用词“我们”、“但是”、“的”等，在所有文档中出现频率很高，但它的IDF值却非常低。甚至如果它每篇文档都出现，则log1的计算结果为0，从而降低了这些常用词的作用；相反，如果某篇介绍“人工智能”的词，仅仅在该篇文档中出现很多次，它的作用就非常高。

TF-IDF技术的核心思想是如果某个特征词在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来做权重计算。TF-IDF算法简单快速，结果也符合实际情况，是文本挖掘、情感分析、主题分布等领域的常用手段。

2.代码实现

Scikit-Learn中主要使用Scikit-Learn中的两个类CountVectorizer和TfidfTransformer，用来计算词频和TF-IDF值。

- **CountVectorizer**
  该类是将文本词转换为词频矩阵的形式。比如“I am a teacher”文本共包含四个单词，它们对应单词的词频均为1，“I”、“am”、“a”、“teacher”分别出现一次。CountVectorizer将生成一个矩阵 a[M][N]，共M个文本语料，N个单词，比如a[i][j]表示单词j在i类文本下的词频。再调用fit_transform()函数计算各个词语出现的次数，get_feature_names()函数获取词库中的所有文本关键词。

```
>>>
一只 一场 中国 产物 位于 分类 可以 回归 多彩 存储 平凡 广泛应用 往往 数学 数据 数据分析 无数次
普通 林城 爬取 爱情 珍藏 甜美 生活 相关 相结合 省会 神州大地 简称 算法 紧密 美誉 聚类 苦涩 西南
地区 计算机科学 贵州 贵州省 贵阳市 走遍 过程 醉美 鸡蛋
[[0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
  1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
  0 0 1 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
  1 1 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
  0 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0
  0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 2 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
  0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 0 0 0 0]
 [1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0]]
>>>
```


- **TfidTransformer**
  当使用CountVectorizer类计算得到词频矩阵后，接下来通过TfidfTransformer类实现统计vectorizer变量中每个词语的TF-IDF值。TF-IDF值采用矩阵数组的形式存储，每一行数据代表一个文本语料，每一行的每一列都代表其中一个特征对应的权重，得到TF-IDF后就可以运用各种数据分析算法进行分析，比如聚类分析、LDA主题分布、舆情分析等等。

```
TfidfTransformer(norm=u'l2', smooth_idf=True, sublinear_tf=False,
         use_idf=True)
[[ 0.          0.          0.46061063  0.          0.46061063  0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.46061063  0.          0.          0.          0.          0.
   0.46061063  0.          0.          0.38903907  0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.          0.
   0.          0.4472136   0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.4472136   0.          0.          0.          0.          0.          0.
   0.          0.          0.4472136   0.          0.          0.4472136
   0.          0.4472136   0.        ]
 [ 0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.46061063  0.          0.
   0.          0.          0.          0.          0.          0.46061063
   0.          0.          0.          0.          0.46061063  0.          0.
   0.          0.          0.          0.38903907  0.46061063  0.          0.
   0.          0.        ]
```


完整代码：

```
# -*- coding:utf-8 -*-
import csv
import pandas as pd
import numpy as np
import jieba
import jieba.analyse
from scipy.sparse import coo_matrix
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

#---------------------------------第一步 读取文件--------------------------------
with open('fenci_data.csv', 'r', encoding='UTF-8') as f:
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        labels.append(row['label']) #0-好评 1-差评
        contents.append(row['content'])

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)
print(type(tfidf))

# 获取词袋模型中的所有词语
word = vectorizer.get_feature_names()
for n in word[:10]:
    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()
X = coo_matrix(tfidf, dtype=np.float32).toarray() #稀疏矩阵 注意float
print(X.shape)
print(X[:10])
```

输出结果如下所示：

```
<class 'scipy.sparse.csr.csr_matrix'>
aaaaa
achievements
amazing
anananananan
ancient
anshun
aperture
app

单词数量: 20254
(6074, 20254)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

3.MemoryError内存溢出错误

当我们数据量很大时，矩阵往往存储不了这么大的数据，会出现如下错误：

- ValueError: array is too big; arr.size * arr.dtype.itemsize is larger than the maximum possible size.
- MemoryError: Unable to allocate array with shape (26771, 69602) and data type float64

我提供的解决方法如下：

- **停用词过滤降低不需要的特征词**
- **scipy包的提供了稀疏矩阵的创建，使用coo_matrix(tfidf, dtype=np.float32)转换tfidf**
- **CountVectorizer(min_df=5)增加min_df参数，过滤掉出现频率少的特征词，该参数可以不断调试**
  max_df用于删除过于频繁出现的术语，称为语料库特定的停用词，默认的max_df是1.0即忽略出现在100％文档的术语；min_df用于删除不经常出现的术语min_df=5表示忽略少于5个文档中出现的术语。
- **使用GPU或扩大内存解决**

四.基于逻辑回归的情感分类

获取文本TF-IDF值之后，本小节简单讲解使用TF-IDF值进行情感分类的过程，主要包括如下步骤：

- 对中文分词和数据清洗后的语料进行词频矩阵生成操作。主要调用CountVectorizer类计算词频矩阵，生成的矩阵为X。
- 调用TfidfTransformer类计算词频矩阵X的TF-IDF值，得到Weight权重矩阵。

- 调用Sklearn机器学习包执行分类操作，调用fit()函数训练，并将预测的类标赋值给pre数组。
- 调用Sklearn库PCA()函数进行降维操作，将这些特征降低为二维，对应X和Y轴，接着进行可视化呈现。
- 算法优化及算法评估。

逻辑回归完整代码：

```
# -*- coding:utf-8 -*-
import csv
import pandas as pd
import numpy as np
import jieba
import jieba.analyse
from scipy.sparse import coo_matrix
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.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import neighbors
from sklearn.naive_bayes import MultinomialNB

#---------------------------------第一步 读取文件--------------------------------
with open('fenci_data.csv', 'r', encoding='UTF-8') as f:
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        labels.append(row['label']) #0-好评 1-差评
        contents.append(row['content'])

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(min_df=5)

#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)
print(type(tfidf))

# 获取词袋模型中的所有词语
word = vectorizer.get_feature_names()
for n in word[:10]:
    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()
X = coo_matrix(tfidf, dtype=np.float32).toarray() #稀疏矩阵 注意float
```

```
print(X.shape)
print(X[:10])

#---------------------------------第三步 数据划分-------------------------------
#使用 train_test_split 分割 X y 列表
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    labels,
                                                    test_size=0.3,
                                                    random_state=1)

#---------------------------------第四步 机器学习分类-------------------------------
# 逻辑回归分类方法模型
LR = LogisticRegression(solver='liblinear')
LR.fit(X_train, y_train)
print('模型的准确度:{}'.format(LR.score(X_test, y_test)))
pre = LR.predict(X_test)
print("逻辑回归分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
print("\n")
```

运行结果如下图所示：

```
单词数量: 4923
(6074, 4923)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
模型的准确度:0.9418540866703237
逻辑回归分类
1823 1823
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      1520
           1       0.93      0.70      0.80       303

    accuracy                           0.94      1823
   macro avg       0.94      0.85      0.88      1823
weighted avg       0.94      0.94      0.94      1823
```

五.算法性能评估

算法评价很多实时需要我们自己编写程序去实现，比如绘制ROC曲线、统计各种特征信息、显示4位数结果。这里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：

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

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

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

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 pandas as pd
import numpy as np
import jieba
import jieba.analyse
from scipy.sparse import coo_matrix
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.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import neighbors
from sklearn.naive_bayes import MultinomialNB

#----------------------------------第一步 读取文件--------------------------------
with open('fenci_data.csv', 'r', encoding='UTF-8') as f:
    reader = csv.DictReader(f)
    labels = []
    contents = []
    for row in reader:
        labels.append(row['label']) #0-好评 1-差评
        contents.append(row['content'])

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(min_df=5)

#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)
print(type(tfidf))

# 获取词袋模型中的所有词语
word = vectorizer.get_feature_names()
for n in word[:10]:
    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()
X = coo_matrix(tfidf, dtype=np.float32).toarray() #稀疏矩阵 注意float
print(X.shape)
print(X[:10])

#----------------------------------第三步 数据划分--------------------------------
#使用 train_test_split 分割 X y 列表
X_train, X_test, y_train, y_test = train_test_split(X,
```

```
                                                    labels,
                                                    test_size=0.3,
                                                    random_state=1)

#--------------------------------第四步 机器学习分类--------------------------------
# 逻辑回归分类方法模型
LR = LogisticRegression(solver='liblinear')
LR.fit(X_train, y_train)
print('模型的准确度:{}'.format(LR.score(X_test, y_test)))
pre = LR.predict(X_test)
print("逻辑回归分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))

#---------------------------------第五步 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("LogisticRegression", y_test, pre)
```

输出结果如下：

```
逻辑回归分类
1823 1823
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      1520
           1       0.93      0.70      0.80       303

    accuracy                           0.94      1823
   macro avg       0.94      0.85      0.88      1823
weighted avg       0.94      0.94      0.94      1823

算法评价: LogisticRegression
213 1504 229 1594 303 1520

Precision Good 0: 0.9435382685069009
Precision Bad 1: 0.9301310043668122
Recall Good 0: 0.9894736842105263
Recall Bad 1: 0.7029702970297029
F-measure Good 0: 0.9659601798330122
F-measure Bad 1: 0.800751879699248
```

六.算法对比实验

1.RandomForest

代码如下：

```
# 随机森林分类方法模型 n_estimators：森林中树的数量
clf = RandomForestClassifier(n_estimators=20)
clf.fit(X_train, y_train)
print('模型的准确度:{}'.format(clf.score(X_test, y_test)))
print("\n")
pre = clf.predict(X_test)
print('预测结果:', pre[:10])
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("RandomForest", y_test, pre)
print("\n")
```

输出结果：

```
预测结果: ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
1823 1823
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1520
           1       0.97      0.99      0.98       303

    accuracy                           0.99      1823
   macro avg       0.98      0.99      0.99      1823
weighted avg       0.99      0.99      0.99      1823

算法评价: RandomForest
299 1511 308 1515 303 1520

Precision Good 0: 0.9973597359735974
Precision Bad 1: 0.9707792207792207
Recall Good 0: 0.9940789473684211
Recall Bad 1: 0.9867986798679867
F-measure Good 0: 0.9957166392092257
F-measure Bad 1: 0.978723404255319
```


2.SVM

代码如下:

```
# 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)
print('模型的准确度:{}'.format(SVM.score(X_test, y_test)))
pre = SVM.predict(X_test)
print("支持向量机分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("LinearSVC", y_test, pre)
print("\n")
```

输出结果:

```
支持向量机分类
1823 1823
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1520
           1       0.92      0.98      0.95       303

    accuracy                           0.98      1823
   macro avg       0.96      0.98      0.97      1823
weighted avg       0.98      0.98      0.98      1823

算法评价: LinearSVC
296 1495 321 1502 303 1520

Precision Good 0: 0.9953395472703063
Precision Bad 1: 0.9221183800623053
Recall Good 0: 0.9835526315789473
Recall Bad 1: 0.976897689768977
F-measure Good 0: 0.9894109861019192
F-measure Bad 1: 0.9487179487179488
```


3.朴素贝叶斯

代码如下:

```
#朴素贝叶斯模型
nb = MultinomialNB()
nb.fit(X_train, y_train)
print('模型的准确度:{}'.format(nb.score(X_test, y_test)))
pre = nb.predict(X_test)
```

```
print("朴素贝叶斯分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("MultinomialNB", y_test, pre)
print("\n")
```

输出结果：

```
模型的准确度:0.9462424574876577
朴素贝叶斯分类
1823 1823
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1520
           1       0.94      0.72      0.82       303

    accuracy                           0.95      1823
   macro avg       0.95      0.86      0.89      1823
weighted avg       0.95      0.95      0.94      1823

算法评价: MultinomialNB
218 1507 231 1592 303 1520

Precision Good 0: 0.946608040201005
Precision Bad 1: 0.9437229437229437
Recall Good 0: 0.9914473684210526
Recall Bad 1: 0.7194719471947195
F-measure Good 0: 0.9685089974293059
F-measure Bad 1: 0.8164794007490638
```


4.KNN

该算法准确率不高，并且执行时间较长，不建议大家用于文本分析。某些情况的算法对比倒是还行，核心代码如下：

```
#最近邻算法
knn = neighbors.KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
print('模型的准确度:{}'.format(knn.score(X_test, y_test)))
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")
```

输出结果：

```
模型的准确度:0.8902907295666483
最近邻分类
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1520
           1       0.80      0.45      0.58       303

    accuracy                           0.89      1823
   macro avg       0.85      0.71      0.76      1823
weighted avg       0.88      0.89      0.88      1823

算法评价: KNeighbors
136 1487 169 1654 303 1520

Precision Good 0: 0.8990326481257558
Precision Bad 1: 0.8047337278106509
Recall Good 0: 0.9782894736842105
Recall Bad 1: 0.44884488448844884
F-measure Good 0: 0.9369880277252678
F-measure Bad 1: 0.576271186440678
```


5.决策树

代码如下：

```
#决策树算法
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
print('模型的准确度:{}'.format(dtc.score(X_test, y_test)))
pre = dtc.predict(X_test)
print("决策树分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("DecisionTreeClassifier", y_test, pre)
print("\n")
```

输出结果：

```
模型的准确度:0.9577619308831596
决策树分类
1823 1823
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1520
           1       0.80      0.99      0.89       303

    accuracy                           0.96      1823
   macro avg       0.90      0.97      0.93      1823
weighted avg       0.97      0.96      0.96      1823

算法评价: DecisionTreeClassifier
299 1447 372 1451 303 1520

Precision Good 0: 0.9972432804962095
Precision Bad 1: 0.803763440860215
Recall Good 0: 0.9519736842105263
Recall Bad 1: 0.9867986798679867
F-measure Good 0: 0.9740828004039044
F-measure Bad 1: 0.885925925925926
```


6.SGD

代码如下：

```
#SGD分类模型
from sklearn.linear_model.stochastic_gradient import SGDClassifier
sgd = SGDClassifier()
sgd.fit(X_train, y_train)
print('模型的准确度:{}'.format(sgd.score(X_test, y_test)))
pre = sgd.predict(X_test)
print("SGD分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("SGDClassifier", y_test, pre)
print("\n")
```

输出结果：

```
模型的准确度:0.9846407021393307
SGD分类
1823 1823
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1520
           1       0.95      0.96      0.95       303

    accuracy                           0.98      1823
   macro avg       0.97      0.97      0.97      1823
weighted avg       0.98      0.98      0.98      1823

算法评价: SGDClassifier
291 1504 307 1516 303 1520

Precision Good 0: 0.9920844327176781
Precision Bad 1: 0.9478827361563518
Recall Good 0: 0.9894736842105263
Recall Bad 1: 0.9603960396039604
F-measure Good 0: 0.9907773386034255
F-measure Bad 1: 0.9540983606557376
```


7.MLP

该算法时间比较慢，核心代码如下：

```
#MLP分类模型
from sklearn.neural_network.multilayer_perceptron import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train, y_train)
print('模型的准确度:{}'.format(mlp.score(X_test, y_test)))
pre = mlp.predict(X_test)
print("MLP分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("MLPClassifier", y_test, pre)
print("\n")
```

输出结果：

```
模型的准确度:0.9835436094349973
MLP分类
1823 1823
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1520
           1       0.92      0.99      0.95       303

    accuracy                           0.98      1823
   macro avg       0.96      0.98      0.97      1823
weighted avg       0.98      0.98      0.98      1823

算法评价: MLPClassifier
299 1494 325 1498 303 1520

Precision Good 0: 0.9973297730307076
Precision Bad 1: 0.92
Recall Good 0: 0.9828947368421053
Recall Bad 1: 0.9867986798679867
F-measure Good 0: 0.9900596421471173
F-measure Bad 1: 0.9522292993630573
```


8.GradientBoosting

该算法时间比较慢，代码如下：

```
#GradientBoosting分类模型
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
```

```
print('模型的准确度:{}'.format(gb.score(X_test, y_test)))
pre = gb.predict(X_test)
print("GradientBoosting分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("GradientBoostingClassifier", y_test, pre)
print("\n")
```

输出结果：

```
GradientBoosting分类
1823 1823
              precision    recall  f1-score   support

           0       0.92      0.99      0.95      1520
           1       0.91      0.58      0.71       303

    accuracy                           0.92      1823
   macro avg       0.91      0.78      0.83      1823
weighted avg       0.92      0.92      0.91      1823

算法评价: GradientBoostingClassifier
176 1502 194 1629 303 1520

Precision Good 0: 0.9220380601596071
Precision Bad 1: 0.9072164948453608
Recall Good 0: 0.9881578947368421
Recall Bad 1: 0.5808580858085809
F-measure Good 0: 0.9539536360749444
F-measure Bad 1: 0.7082494969818914
```


9.AdaBoost

代码如下：

```
#AdaBoost分类模型
from sklearn.ensemble import AdaBoostClassifier
AdaBoost = AdaBoostClassifier()
AdaBoost.fit(X_train, y_train)
print('模型的准确度:{}'.format(AdaBoost.score(X_test, y_test)))
pre = AdaBoost.predict(X_test)
print("AdaBoost分类")
print(len(pre), len(y_test))
print(classification_report(y_test, pre))
classification_pj("AdaBoostClassifier", y_test, pre)
print("\n")
```

输出结果：

```
模型的准确度:0.9171695008228196
AdaBoost分类
1823 1823
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      1520
           1       0.80      0.67      0.73       303

    accuracy                           0.92      1823
   macro avg       0.87      0.82      0.84      1823
weighted avg       0.91      0.92      0.91      1823

算法评价: AdaBoostClassifier
202 1470 252 1571 303 1520

Precision Good 0: 0.9357097390197326
Precision Bad 1: 0.8015873015873016
Recall Good 0: 0.9671052631578947
Recall Bad 1: 0.6666666666666666
F-measure Good 0: 0.9511484956324814
F-measure Bad 1: 0.7279279279279279
```

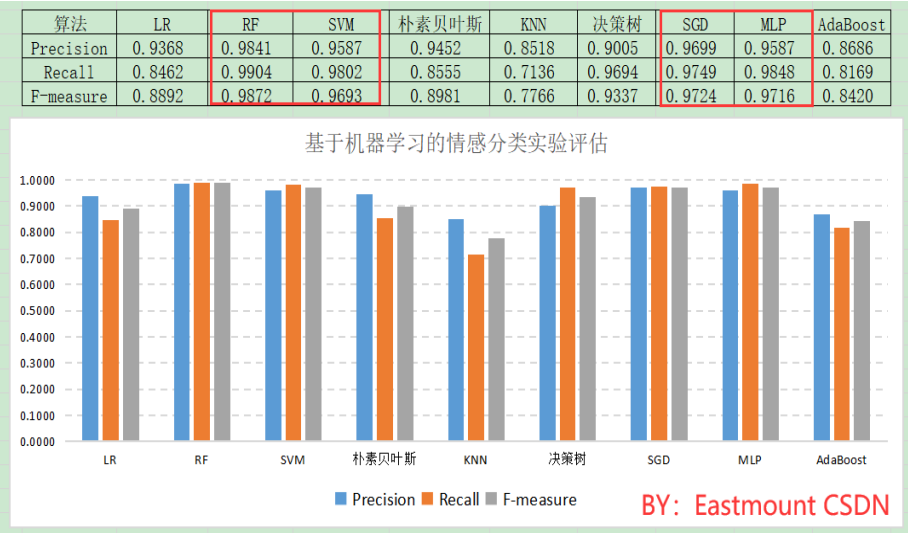
七.总结

写到这里，这篇文章就结束了，下一篇我将带领大家看看深度学习（BiLSTM-CNN）的情感分类方法。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Sklearn实现了各种机器学习的情感分类算法，并且我们可以进行实验对比，如下图所示，发现随机森林、SVM、SGD、MLP效果还不错，当然不同数据集效果是不同的，大家需要结合自己的数据集去完成。github下载代码，记得关注点赞喔。

- https://github.com/eastmountyxz/Sentiment-Analysis

算法	LR	RF	SVM	朴素贝叶斯	KNN	决策树	SGD	MLP	AdaBoost
Precision	0.9368	0.9841	0.9587	0.9452	0.8518	0.9005	0.9699	0.9587	0.8686
Recall	0.8462	0.9904	0.9802	0.8555	0.7136	0.9694	0.9749	0.9848	0.8169
F-measure	0.8892	0.9872	0.9693	0.8981	0.7766	0.9337	0.9724	0.9716	0.8420



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~

最近参加了奇安信和清华大学举办的大数据安全比赛，收获非常多，也意识到了鸿沟般的差距。我主要分析的是HC和恶意家族网站分类，大概能从200万条真实网站钟识别了十万多个HC网站，涉及数据抓取、恶意流量检测、跳转劫持判断、NLP和大数据等方法。最后五个方向获奖的主要是清华、中科院信工所、阿里巴巴团队，也有北大、浙大、上交等团队，好厉害，好想学习他们的writeup。真的非常珍惜这样的实战机会，希望未来继续加油，某年能冲进前三名拿个奖。虽然自己很菜，但接下来还是会分享我的大数据分析方法，与大家

一起进步。未知攻，焉知防，安全路上还请给位朋友和大佬多多请教，也希望自己能在学术和实战两个方向都进步。有差距不可怕，重要的是我努力过，分享过，加油。最后感谢女神点指导和开导，哈哈~

(By:Eastmount 2020-08-17 周一下午3点写于武汉 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。

参考文献：
[1] 杨秀璋《Python网络数据爬取及分析从入门到精通（分析篇）》
[2] https://blog.csdn.net/WANG_hl/article/details/105234432
[3] https://blog.csdn.net/qq_27590277/article/details/106894245
[4] https://www.cnblogs.com/alivinfer/p/12892147.html
[5] https://blog.csdn.net/qq_28626909/article/details/80382029
[6] https://www.jianshu.com/p/3da3f5608a7c