目录

【NLP 系列3】14种分类算法进行文本分类实战

27 3.6 k 117

本篇重视使用函数来提高代码复用率，使用14种分类算法进行文本分类，以此同时比较不同算法之间的性能情况，适合进阶和提高。

小知同学 LV4 4 · DAA数据与算法

Fork 点赞

内容 Fork记录 117 评论 3

内容

版本 5 2020-02-10 12:18 基础镜像 附件

In [10]:

```
# 查看当前挂载的数据集目录
!ls /home/kesci/work/xiaozhi
```

```
my_network_model.h5  paper.rar     stopword.txt             tx3.jpg
mydata.txt           pos.txt       text_classification
neg.txt              save_tx3.png  text_classification.zip
```

目录

前言

本篇文本分类实战训练是以完整的文本分类项目流程来写的，比较适合进阶和提高
《获取数据》——《数据分析和处理》——《特征工程与选择》——《算法模型》——《性能评估/参数调优》

这一篇训练比较重视批量读取和处理文本数据集；

其中也比较重视函数的使用以提高代码的复用率。

在分类算法这一块，本篇项目总共使用了 《14》 种分类算法来进行文本分类，

涵盖《sklearn》中的常规分类算法和集成学习算法；

竞赛和工业界比较得宠的集成学习算法《xgboost》和《lightgbm》；

深度学习框架《Keras》中的前馈神经网络和《LSTM》算法。

本篇目录：

往期文章学习

https://www.kesci.com/home/column/5cb43d67e0ad99002cad14d6 (https://www.kesci.com/home/column/5cb43d67e0ad99002cad14d6)

数据集介绍

获取数据

1 目录情况

```
——text_classification

    ——train
        ——女性
            ——.txt  ...
```

目录

```
        ——体育
            ——.txt  ...
        ——文学
            ——.txt  ...
        ——校园
            ——.txt  ...
    ——test
        ——女性
            ——.txt  ...
        ——体育
            ——.txt  ...
        ——文学
            ——.txt  ...

        ——校园
            ——.txt  ...
    ——stopword.txt
```

2 数据集介绍

	样本数/.txt文件数	标签
训练集：	3305	[女性，体育，文学，校园]
测试集	200	[女性，体育，文学，校园]

3 数据来源

https://github.com/cystanford/text_classification (https://github.com/cystanford/text_classification)

目标定义

目标需求1：批量读取文档文件.txt
目标需求2：文本向量化使用词袋法或者TFIDF
目标需求3：比较各个算法的性能优劣，包括模型消耗用时、模型准确率

导入所需模块

In [2]:

```
import os
import shutil
import zipfile
import jieba
import time
import warnings
import xgboost
import lightgbm
import numpy as np
import pandas as pd
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.preprocessing.text import Tokenizer
from sklearn import svm
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

warnings.filterwarnings('ignore')
```

```
/opt/conda/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: numpy.dtype size changed, may indicate binary
  return f(*args, **kwds)
/opt/conda/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: numpy.dtype size changed, may indicate binary
```

```
/opt/conda/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: numpy.dtype size changed, may indicate binary
  return f(*args, **kwds)
Using TensorFlow backend.
```

目录

1. 解压文件并处理中文乱码

> 解压压缩包
>
> 解决目录或文件中文名字乱码的问题

In [3]:

```
# ----------------------------------------
# 路径
filepath   = '/home/kesci/work/xiaozhi/text_classification.zip'
savepath   = '/home/kesci/work/xiaozhi/'
delectpath = '/home/kesci/work/xiaozhi/text_classification'

# ----------------------------------------
# 检索并删除文件夹
if os.path.exists(delectpath):
    print('\n存在该文件夹，正在进行删除，防止解压重命名失败......\n')
    shutil.rmtree(delectpath)
else:
    print('\n不存在该文件夹，请放心处理......\n')

# ----------------------------------------
# 解压并处理中文名字乱码的问题
z = zipfile.ZipFile(filepath, 'r')
for file in z.namelist():
    # 中文乱码需处理
    filename = file.encode('cp437').decode('gbk') # 先使用 cp437 编码，然后再使用 gbk 解码
    z.extract(file, savepath)                     # 解压 ZIP 文件
    # 解决乱码问题
    os.chdir(savepath)                            # 切换到目标目录
    os.rename(file, filename)                     # 将乱码重命名文件
```

```
存在该文件夹，正在进行删除，防止解压重命名失败......
```

2. 批量读取和合并文本数据集

> 本案例注重编写函数来批量读取所有文档的文本数据，提高代码复用率
>
> 本案例注重函数格式的要求，添加参数注释，提高代码阅读率

In [4]:

```
def read_text(path, text_list):
    '''
    path: 必选参数，文件夹路径
    text_list: 必选参数，文件夹 path 下的所有 .txt 文件名列表
    return: 返回值
        features 文本(特征)数据，以列表形式返回;
        labels 分类标签，以列表形式返回
    '''

    features, labels = [], []
    for text in text_list:
        if text.split('.')[-1] == 'txt':
            try:
                with open(path + text, encoding='gbk') as fp:
                    features.append(fp.read())          # 特征
                    labels.append(path.split('/')[-2])  # 标签
            except Exception as erro:
                print('\n>>>发现错误，正在输出错误信息...\n', erro)

    return features, labels

def merge_text(train_or_test, label_name):
    '''
    train_or_test: 必选参数，train 训练数据集 or test 测试数据集
    label_name: 必选参数，分类标签的名字
    return: 返回值
        merge_features 合并好的所有特征数据，以列表形式返回;
        merge_labels   合并好的所有分类标签数据，以列表形式返回
    '''

    print('\n>>>文本读取和合并程序已经启动，请稍候...')
```

目录

```
    merge_features, merge_labels = [], []  # 函数全局变量
    for name in label_name:
        path = '/home/kesci/work/xiaozhi/text_classification/'+ train_or_test +'/'+ name +'/'
        text_list = os.listdir(path)
        features, labels = read_text(path=path, text_list=text_list)  # 调用函数
        merge_features += features  # 特征
        merge_labels   += labels    # 标签

    # 可以自定义添加一些想要知道的信息
    print('\n>>>你正在处理的数据类型是...\n', train_or_test)
    print('\n>>>[', train_or_test ,']数据具体情况如下...')
    print('样本数量\t', len(merge_features), '\t类别名称\t', set(merge_labels))
    print('\n>>>文本读取和合并工作已经处理完毕...\n')

    return merge_features, merge_labels
```

In [5]:

```
# 获取训练集

train_or_test = 'train'

label_name = ['女性', '体育', '校园', '文学']
X_train, y_train = merge_text(train_or_test, label_name)
```

```
>>>文本读取和合并程序已经启动，请稍候...

>>>发现错误，正在输出错误信息...
 'gbk' codec can't decode byte 0xaa in position 87: illegal multibyte sequence

>>>你正在处理的数据类型是...
 train

>>>[ train ]数据具体情况如下...
样本数量   3305    类别名称   {'女性', '文学', '体育', '校园'}

>>>文本读取和合并工作已经处理完毕...
```

In [6]:

```
# 获取测试集

train_or_test = 'test'

label_name = ['女性', '体育', '校园', '文学']
X_test, y_test = merge_text(train_or_test, label_name)
```

```
>>>文本读取和合并程序已经启动，请稍候...

>>>你正在处理的数据类型是...
 test

>>>[ test ]数据具体情况如下...
样本数量   200     类别名称   {'女性', '文学', '体育', '校园'}

>>>文本读取和合并工作已经处理完毕...
```

3. 中文文本分词

In [7]:

```
# 训练集

X_train_word = [jieba.cut(words) for words in X_train]
X_train_cut = [' '.join(word) for word in X_train_word]

X_train_cut[:5]
```

```
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.837 seconds.
Prefix dict has been built succesfully.
```

Out[7]:

```
['冬季 饮食 减肥 小窍门 吃 ， 我 所欲 也 ； 瘦 ， 亦 我 所欲 也 。 难道 吃 和 瘦 真的 不可 兼得 吗 ？ 没这回事 宝迪沃 告诉 你 冬天 ，
 '优惠 信息 今日 拍拍网 母君宝康 母婴 专营店 乳钙 软胶囊 惊爆价 ： 39 元 含泪 上架 （ 优惠 还有 升级版 ： 今日 购买 一盒 该 软胶囊 任
 '老年斑 形成 的 原因 华美 段跃 民主 任 介绍 说 ， 人到 一定 年纪 后 ， 体内 细胞 功能 的 衰退 在 逐年 加速 ， 血液循环 也 趋向 缓慢
 '  | |   时尚 美人团 ：  各种 眼线 的 画法 ~ 转给 想画 出 漂亮 眼线 的 姐妹 们 吧 ~ $ LOTOzf $             女人 天生 爱美丽 ：
 '打造 小腰 精 ： 直立 ， 一臂 上举 ， 一臂 伸向 背后 。 左臂 后伸 ， 右臂 向上 伸 ， 身体 左转 ， 下肢 保持 不 动 ， 停 20 秒 ， 换
```

In [8]:

```
# 测试集

X_test_word = [jieba.cut(words) for words in X_test]
X_test_cut  = [' '.join(word) for word in X_test_word]

X_test_cut[:5]
```

Out[8]:

```
['【 祛痘 饮食 三多两 忌 】 一 ：  多吃含 锌 含钙 食物 。 锌 可 加速 蛋白质 合成 及 细胞 再生 ， 并 促进 伤口 愈合 ， 而 钙 能 安抚
 '【 12 星座 思春 表现 】 白羊 （ 发呆 沉思 ） 、 金牛 （ 闷 着 长毛 ） 、 双子 （ 狂过 嘴瘾 ） 、 巨蟹 （ 有心 没胆 ） 、 狮子 （
 '冬天 真是 万恶 的 季节 ， 食欲 越来越 大 ， 控制 不住 。 胖友 介绍 “ 瘦身 之颠 ” 这 款 减肥 产品 ， 连续 看 了 十几页 的 评价 ，
 '美丽 微 课堂 光子 嫩肤 项目 是 近 5 年 发展 起来 的 一种 带有 美容 性 治疗 的 一种 技术 。 可以 说 是 脱毛 的 孪生兄弟 。 真正 的
 '    | |    樱花 草 丶 忧伤 ：   【 事业 婚姻 两 得意 的 星座 女 】 第三名 ：天蝎座 $ LOTOzf $             星座 宝典 ： 【 事业
```

4. 停止词使用

1 停止词是为了过滤掉已经确认对训练模型没有实际价值的分词

2 停止词作用：提高模型的精确度和稳定性

In [9]:

```
# 加载停止词语料

stoplist = [word.strip() for word in open('/home/kesci/work/xiaozhi/text_classification/stopword.txt', \
                                           encoding='utf-8').readlines()]
stoplist[:10]
```

Out[9]:

```
['\ufeff,', '?', '、', '。', '“', '”', '《', '》', '！', '，']
```

5. 编码器处理文本标签

In [10]:

```
le = LabelEncoder()

y_train_le = le.fit_transform(y_train)
y_test_le  = le.fit_transform(y_test)

y_train_le, y_test_le
```

Out[10]:

```
(array([1, 1, 1, ..., 2, 2, 2]),
 array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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,
        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, 0, 0, 0, 0, 0, 0, 0, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2]))
```

文本数据转换成数据值数据矩阵

In [11]:

```
count = CountVectorizer(stop_words=stoplist)

'''注意:
这里要先 count.fit() 训练所有训练和测试集，保证特征数一致，
这样在算法建模时才不会报错
'''

count.fit(list(X_train_cut) + list(X_test_cut))
X_train_count = count.transform(X_train_cut)
X_test_count  = count.transform(X_test_cut)

X_train_count = X_train_count.toarray()
X_test_count  = X_test_count.toarray()

print(X_train_count.shape, X_test_count.shape)
X_train_count, X_test_count
```

```
(3305, 23732) (200, 23732)
```

Out[11]:

```
(array([[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]]), array([[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]]))
```

6. 算法模型

封装一个函数，提高复用率，使用时只需调用函数即可

In [12]:

```
# 用于存储所有算法的名字，准确率和所消耗的时间
estimator_list, score_list, time_list = [], [], []
```

In [13]:

```
def get_text_classification(estimator, X, y, X_test, y_test):
    '''
    estimator: 分类器，必选参数
            X: 特征训练数据，必选参数
            y: 标签训练数据，必选参数
       X_test: 特征测试数据，必选参数
        y_tes: 标签测试数据，必选参数
       return: 返回值
           y_pred_model: 预测值
             classifier: 分类器名字
                  score: 准确率
                      t: 消耗的时间
                 matrix: 混淆矩阵
                 report: 分类评价函数

    '''
    start = time.time()

    print('\n>>>算法正在启动，请稍候...')
    model = estimator

    print('\n>>>算法正在进行训练，请稍候...')
    model.fit(X, y)
    print(model)

    print('\n>>>算法正在进行预测，请稍候...')
    y_pred_model = model.predict(X_test)
    print(y_pred_model)

    print('\n>>>算法正在进行性能评估，请稍候...')
    score = metrics.accuracy_score(y_test, y_pred_model)
    matrix = metrics.confusion_matrix(y_test, y_pred_model)
    report = metrics.classification_report(y_test, y_pred_model)

    print('>>>准确率\n', score)
    print('\n>>>混淆矩阵\n', matrix)
    print('\n>>>召回率\n', report)
    print('>>>算法程序已经结束...')

    end = time.time()
    t = end - start
    print('\n>>>算法消耗时间为：', t, '秒\n')
    classifier = str(model).split('(')[0]

    return y_pred_model, classifier, score, round(t, 2), matrix, report
```

常规算法——方法1——k 近邻算法

In [14]:

```
knc = KNeighborsClassifier()

result = get_text_classification(knc, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])
```

```
>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候
```

```
>>>算法正在进行训练，请稍候...
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')

>>>算法正在进行预测，请稍候...
[0 0 2 0 1 0 0 0 0 1 0 1 0 1 0 0 2 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1
 1 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0
 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 2 2 2 0 0 0 0 0 0 0 0
 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 2 2 2 2 3 0 0 2 0 3 0 0 0 2 0 2 2 0 2 2 2 2 2 2 2 2 2 2 2 0 0 2 1
 2 0 2 2 2 2 2 2 0 2 0 0 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.67

>>>混淆矩阵
 [[99  0 16  0]
 [26 10  2  0]
 [ 7  1 23  0]
 [ 7  0  7  2]]

>>>召回率
               precision    recall  f1-score   support

           0       0.71      0.86      0.78       115
           1       0.91      0.26      0.41        38
           2       0.48      0.74      0.58        31
           3       1.00      0.12      0.22        16

   micro avg       0.67      0.67      0.67       200
   macro avg       0.78      0.50      0.50       200
weighted avg       0.74      0.67      0.63       200

>>>算法程序已经结束...

>>>算法消耗时间为： 32.66772508621216 秒


Out[14]:
(None, None, None)
```

常规算法——方法2——决策树

```
In [15]:
dtc = DecisionTreeClassifier()

result = get_text_classification(dtc, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])

>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 3 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 0 0 0 0 1 0 3 0 0 0 1 0 0 0 0 2 0 1 0 0 1 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0
 1 0 0 0 1 2 0 0 1 1 0 0 0 0 0 2 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 2 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 3 0 0 0 0 0 0 1 0 0 0 0 1 0 0
 0 1 0 1 0 2 3 3 3 3 3 3 2 1 3 3 0 3 2 1 2 2 1 2 1 2 2 2 2 3 2 1 3 2 3 2 1
 2 3 2 2 2 2 2 2 2 2 1 2 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.76

>>>混淆矩阵
 [[87 21  5  2]
 [ 1 34  2  1]
 [ 0  5 22  4]
 [ 1  2  4  9]]

>>>召回率
               precision    recall  f1-score   support
```

目录

```
              0       0.98      0.76      0.85       115
              1       0.55      0.89      0.68        38
              2       0.67      0.71      0.69        31
              3       0.56      0.56      0.56        16

    micro avg       0.76      0.76      0.76       200
    macro avg       0.69      0.73      0.70       200
 weighted avg       0.81      0.76      0.77       200

>>>算法程序已经结束...

>>>算法消耗时间为： 38.00881791114807 秒
```

Out[15]:

```
(None, None, None)
```

常规算法——方法3——多层感知器

In [16]:

```
mlpc = MLPClassifier()

result = get_text_classification(mlpc, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])
```

```
>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 2 1 1 1 1 1 1 1
 1 0 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 3 0 0 0 0 0 0 2 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 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 2 3 3 3 3 3 1 2 2 3 3 0 3 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.89

>>>混淆矩阵
 [[105   2   7   1]
 [  1  36   1   0]
 [  1   1  29   0]
 [  2   1   5   8]]

>>>召回率
               precision    recall  f1-score   support

           0       0.96      0.91      0.94       115
           1       0.90      0.95      0.92        38
           2       0.69      0.94      0.79        31
           3       0.89      0.50      0.64        16

   micro avg       0.89      0.89      0.89       200
   macro avg       0.86      0.82      0.82       200
weighted avg       0.90      0.89      0.89       200

>>>算法程序已经结束...

>>>算法消耗时间为： 213.85095739364624 秒
```

Out[16]:

```
(None, None, None)
```

常规算法——方法4——伯努力贝叶斯算法

In [17]:

```
bnb = BernoulliNB()
```

```
result = get_text_classification(bnb, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])
```

```
>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1
 1 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 2 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 0 0 0 0 0 0 2 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 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 2 0 0 2 0 0 2 0 2 0 0 0 0 0
 0 2 0 0 0 0 0 2 2 2 0 2 2 2 0]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.785

>>>混淆矩阵
 [[113   0   2   0]
 [  6  32   0   0]
 [ 19   0  12   0]
 [ 15   0   1   0]]

>>>召回率
              precision    recall  f1-score   support

           0       0.74      0.98      0.84       115
           1       1.00      0.84      0.91        38
           2       0.80      0.39      0.52        31
           3       0.00      0.00      0.00        16

   micro avg       0.79      0.79      0.79       200
   macro avg       0.63      0.55      0.57       200
weighted avg       0.74      0.79      0.74       200

>>>算法程序已经结束...

>>>算法消耗时间为： 1.267916202545166 秒
```

Out[17]:

```
(None, None, None)
```

常规算法——方法5——高斯贝叶斯

In [18]:

```
gnb = GaussianNB()

result = get_text_classification(gnb, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])
```

```
>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
GaussianNB(priors=None, var_smoothing=1e-09)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 2 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 3 0 0 0 0 0 0 2 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 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 3 3 3 3 3 3 1 3 1 3 3 2 2 3 0 3 0 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.89

>>>混淆矩阵
 [[105   1   8   1]
 [  2  34   2   0]
 [  1   1  28   1]
 [  1   2   2  11]]

>>>召回率
              precision    recall  f1-score   support

           0       0.96      0.91      0.94       115
```

目录

```
              1       0.89      0.89      0.89        38
              2       0.70      0.90      0.79        31
              3       0.85      0.69      0.76        16

    micro avg       0.89      0.89      0.89       200
    macro avg       0.85      0.85      0.84       200
 weighted avg       0.90      0.89      0.89       200

 >>>算法程序已经结束...

 >>>算法消耗时间为： 2.20556640625 秒
```

Out[18]:

```
(None, None, None)
```

常规算法——方法6——多项式朴素贝叶斯

In [19]:

```
mnb = MultinomialNB()

result = get_text_classification(mnb, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])

>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 3 1 1 1 1 2 1 1 1 1 1 1 1
 1 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 0 0 0 0 0 0
 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 3 0 0 0 0 0 0 0 0 0
 0 0 0 0 2 0 0 0 0 0 0 0 2 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 0 0 3 3 3 3 3 2 0 2 3 3 3 3 0 3 0 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.905

>>>混淆矩阵
 [[108   1   2   4]
 [  0  33   3   2]
 [  1   0  30   0]
 [  4   0   2  10]]

>>>召回率
               precision    recall  f1-score   support

           0       0.96      0.94      0.95       115
           1       0.97      0.87      0.92        38
           2       0.81      0.97      0.88        31
           3       0.62      0.62      0.62        16

   micro avg       0.91      0.91      0.91       200
   macro avg       0.84      0.85      0.84       200
weighted avg       0.91      0.91      0.91       200

>>>算法程序已经结束...

>>>算法消耗时间为： 0.6552143096923828 秒
```

Out[19]:

```
(None, None, None)
```

常规算法——方法7——逻辑回归算法

In [20]:

```
lgr = LogisticRegression()

result = get_text_classification(lgr, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])

>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn'
```

目
录

```
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 2 1 1 1 1 1 1 1
 1 0 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0
 1 0 0 0 2 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
 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 3 3 3 3 3 3 2 2 3 3 0 3 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 1
 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.89

>>>混淆矩阵
 [[105   3   7   0]
 [  0  36   1   1]
 [  1   2  28   0]
 [  5   0   2   9]]

>>>召回率
               precision    recall  f1-score   support

           0       0.95      0.91      0.93       115
           1       0.88      0.95      0.91        38
           2       0.74      0.90      0.81        31
           3       0.90      0.56      0.69        16

   micro avg       0.89      0.89      0.89       200
   macro avg       0.87      0.83      0.84       200
weighted avg       0.90      0.89      0.89       200

>>>算法程序已经结束...

>>>算法消耗时间为： 0.7971396446228027 秒
```

```
Out[20]:
```

```
(None, None, None)
```

常规算法——方法8——支持向量机算法

```
In [21]:
```

```
svc = svm.SVC()

result = get_text_classification(svc, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])
```

```
>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)

>>>算法正在进行预测，请稍候...
[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 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 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 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 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 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.575

>>>混淆矩阵
 [[115   0   0   0]
 [ 38   0   0   0]
 [ 31   0   0   0]
 [ 16   0   0   0]]

>>>召回率
               precision    recall  f1-score   support

           0       0.57      1.00      0.73       115
           1       0.00      0.00      0.00        38
           2       0.00      0.00      0.00        31
```

```
              3       0.00      0.00      0.00        16

   micro avg       0.57      0.57      0.57       200
   macro avg       0.14      0.25      0.18       200
weighted avg       0.33      0.57      0.42       200

>>>算法程序已经结束...

>>>算法消耗时间为： 398.7027745246887 秒
```

Out[21]:

```
(None, None, None)
```

集成学习算法——方法1——随机森林算法

In [22]:

```
rfc = RandomForestClassifier()

result = get_text_classification(rfc, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])

>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 2 3 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1
 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0 0 2 0 0 0 0 2 0 0 0 0
 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 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 2 0 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0
 0 0 0 0 0 2 3 2 2 3 3 1 2 0 3 3 0 3 2 0 2 0 0 2 2 2 2 2 2 2 2 2 2 0 0 2 3
 2 3 2 2 2 2 2 2 2 2 2 1 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.82

>>>混淆矩阵
 [[103   3   8   1]
 [  5  31   1   1]
 [  4   1  24   2]
 [  3   1   6   6]]

>>>召回率
               precision    recall  f1-score   support

           0       0.90      0.90      0.90       115
           1       0.86      0.82      0.84        38
           2       0.62      0.77      0.69        31
           3       0.60      0.38      0.46        16

   micro avg       0.82      0.82      0.82       200
   macro avg       0.74      0.72      0.72       200
weighted avg       0.82      0.82      0.82       200

>>>算法程序已经结束...

>>>算法消耗时间为： 6.421279668807983 秒
```

Out[22]:

```
(None, None, None)
```

集成学习算法——方法2——自增强算法

In [23]:

```
abc = AdaBoostClassifier()

result = get_text_classification(abc, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])
```

```
>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0
 1 0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 1 1 0
 1 1 0 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 0 1
 1 0 0 1 0 1 3 3 3 3 3 3 1 1 3 3 1 3 1 1 1 0 1 2 1 1 1 1 2 3 2 1 3 1 0 1 1
 1 3 2 2 1 2 1 1 2 2 1 1 2 1 1]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.57

>>>混淆矩阵
 [[59 56  0  0]
 [ 1 37  0  0]
 [ 2 17  9  3]
 [ 0  7  0  9]]

>>>召回率
              precision    recall  f1-score   support

           0       0.95      0.51      0.67       115
           1       0.32      0.97      0.48        38
           2       1.00      0.29      0.45        31
           3       0.75      0.56      0.64        16

   micro avg       0.57      0.57      0.57       200
   macro avg       0.75      0.58      0.56       200
weighted avg       0.82      0.57      0.60       200

>>>算法程序已经结束...

>>>算法消耗时间为： 149.1696276664734 秒


Out[23]:

(None, None, None)
```

集成学习算法——方法3——lightgbm算法

```
In [24]:

gbm = lightgbm.LGBMClassifier()

result = get_text_classification(gbm, X_train_count, y_train_le, X_test_count, y_test_le)
estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])

>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
```



```
        n_jobs=-1, num_leaves=31, objective=None, random_state=None,
        reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=1.0,
        subsample_for_bin=200000, subsample_freq=1)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1
 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0
 1 0 0 0 2 2 0 0 3 0 0 0 0 0 0 2 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0
 2 0 0 0 0 0 0 0 0 0 1 0 2 0 3 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 1 0 2 3 2 2 3 3 3 2 2 3 3 0 3 2 1 2 0 2 2 2 2 2 2 2 2 2 2 3 3 2 0 1
 2 3 2 2 2 2 2 1 2 2 2 1 2 2 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.795

>>>混淆矩阵
 [[94  6 13  2]
 [ 0 35  1  2]
 [ 2  3 23  3]
 [ 1  1  7  7]]

>>>召回率
```

目录

```
                precision    recall  f1-score   support

             0       0.97      0.82      0.89       115
             1       0.78      0.92      0.84        38
             2       0.52      0.74      0.61        31
             3       0.50      0.44      0.47        16

    micro avg       0.80      0.80      0.80       200
    macro avg       0.69      0.73      0.70       200
weighted avg       0.83      0.80      0.80       200

>>>算法程序已经结束...

>>>算法消耗时间为： 2.091001510620117 秒

Out[24]:

(None, None, None)
```

集成学习算法——方法4——xgboost算法

xgboost 模型运行有点慢，这里需要等待一阵子

```
In [25]:

 xgb = xgboost.XGBClassifier()

 result = get_text_classification(xgb, X_train_count, y_train_le, X_test_count, y_test_le)
 estimator_list.append(result[1]), score_list.append(result[2]), time_list.append(result[3])

>>>算法正在启动，请稍候...

>>>算法正在进行训练，请稍候...
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)

>>>算法正在进行预测，请稍候...
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0
 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0
 0 1 0 1 0 1 3 3 3 3 3 3 0 1 3 3 0 3 1 1 1 2 1 2 1 1 2 2 2 3 2 1 1 0 2 1 1
 1 3 2 2 2 2 2 1 2 2 2 1 2 1 2]

>>>算法正在进行性能评估，请稍候...
>>>准确率
 0.735

>>>混淆矩阵
 [[83 32  0  0]
 [ 0 38  0  0]
 [ 1 11 17  2]
 [ 2  5  0  9]]

>>>召回率
                precision    recall  f1-score   support

             0       0.97      0.72      0.83       115
             1       0.44      1.00      0.61        38
             2       1.00      0.55      0.71        31
             3       0.82      0.56      0.67        16

    micro avg       0.73      0.73      0.73       200
    macro avg       0.81      0.71      0.70       200
weighted avg       0.86      0.73      0.75       200

>>>算法程序已经结束...

>>>算法消耗时间为： 602.9425418376923 秒

Out[25]:

(None, None, None)
```

深度学习算法——方法1——多分类前馈神经网络

虽然 Keras 也是一个高级封装的接口，但对初学者来说也会很容易混淆一些地方，所以小知同学来说一些概念。

1 算法流程：

创建神经网络—添加神经层—编译神经网络—训练神经网络—预测—性能评估—保存模型

目录

2 添加神经层

至少要有两层神经层，第一层必须是输入神经层，最后一层必须是输出层；

输入神经层主要设置输入的维度，而最后一层主要是设置激活函数的类型来指明是分类还是回归问题

3 编译神经网络

分类问题的 metrics，一般以 accuracy 准确率来衡量

回归问题的 metrics，一般以 mae 平均绝对误差来衡量

暂时就说这些比较容易混淆的知识点

In [26]:

```
start = time.time()
# ---------------------------------
# np.random.seed(0)     # 设置随机数种子
feature_num = X_train_count.shape[1]     # 设置所希望的特征数量

# ---------------------------------
# 独热编码目标向量来创建目标矩阵
y_train_cate = to_categorical(y_train_le)
y_test_cate = to_categorical(y_test_le)
print(y_train_cate)
```

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

In [27]:

```
# ------------------------------------------------------
# 1 创建神经网络
network = models.Sequential()

# ------------------------------------------------------
# 2 添加神经连接层
# 第一层必须有并且一定是 [输入层]，必选
network.add(layers.Dense(     # 添加带有 relu 激活函数的全连接层
                         units=128,
                         activation='relu',
                         input_shape=(feature_num, )
                         ))

# 介于第一层和最后一层之间的称为 [隐藏层]，可选
network.add(layers.Dense(     # 添加带有 relu 激活函数的全连接层
                         units=128,
                         activation='relu'
                         ))
network.add(layers.Dropout(0.8))
# 最后一层必须有并且一定是 [输出层]，必选
network.add(layers.Dense(     # 添加带有 softmax 激活函数的全连接层
                         units=4,
                         activation='sigmoid'
                         ))

# ------------------------------------------------------
# 3 编译神经网络
network.compile(loss='categorical_crossentropy',  # 分类交叉熵损失函数
                optimizer='rmsprop',
                metrics=['accuracy']              # 准确率度量
                )

# ------------------------------------------------------
# 4 开始训练神经网络
history = network.fit(X_train_count,     # 训练集特征
            y_train_cate,        # 训练集标签
            epochs=20,          # 迭代次数
            batch_size=300,    # 每个批量的观测数  可做优化
            validation_data=(X_test_count, y_test_cate)  # 验证测试集数据
            )
network.summary()
```

```
Train on 3305 samples, validate on 200 samples
Epoch 1/20
```

目录

```
3305/3305 [==============================] - 2s 587us/step - loss: 1.2444 - acc: 0.5425 - val_loss: 1.0093 - val_acc
Epoch 2/20
3305/3305 [==============================] - 2s 490us/step - loss: 0.8650 - acc: 0.7256 - val_loss: 0.6828 - val_acc
Epoch 3/20
3305/3305 [==============================] - 2s 489us/step - loss: 0.5740 - acc: 0.8551 - val_loss: 0.5079 - val_acc
Epoch 4/20
3305/3305 [==============================] - 2s 489us/step - loss: 0.3937 - acc: 0.9147 - val_loss: 0.3795 - val_acc
Epoch 5/20
3305/3305 [==============================] - 2s 494us/step - loss: 0.2743 - acc: 0.9467 - val_loss: 0.2993 - val_acc
Epoch 6/20
3305/3305 [==============================] - 2s 493us/step - loss: 0.1883 - acc: 0.9728 - val_loss: 0.2564 - val_acc
Epoch 7/20
3305/3305 [==============================] - 2s 491us/step - loss: 0.1326 - acc: 0.9770 - val_loss: 0.2308 - val_acc
Epoch 8/20
3305/3305 [==============================] - 2s 492us/step - loss: 0.0978 - acc: 0.9846 - val_loss: 0.2241 - val_acc
Epoch 9/20
3305/3305 [==============================] - 3s 932us/step - loss: 0.0788 - acc: 0.9879 - val_loss: 0.2210 - val_acc
Epoch 10/20
3305/3305 [==============================] - 3s 962us/step - loss: 0.0564 - acc: 0.9888 - val_loss: 0.2041 - val_acc
Epoch 11/20
3305/3305 [==============================] - 3s 981us/step - loss: 0.0479 - acc: 0.9906 - val_loss: 0.2118 - val_acc
Epoch 12/20
3305/3305 [==============================] - 3s 1ms/step - loss: 0.0376 - acc: 0.9936 - val_loss: 0.2109 - val_acc:
Epoch 13/20
3305/3305 [==============================] - 3s 1ms/step - loss: 0.0363 - acc: 0.9930 - val_loss: 0.1821 - val_acc:
Epoch 14/20
3305/3305 [==============================] - 3s 980us/step - loss: 0.0241 - acc: 0.9964 - val_loss: 0.2104 - val_acc
Epoch 15/20
3305/3305 [==============================] - 3s 996us/step - loss: 0.0225 - acc: 0.9958 - val_loss: 0.2092 - val_acc
Epoch 16/20
3305/3305 [==============================] - 3s 1ms/step - loss: 0.0236 - acc: 0.9955 - val_loss: 0.2074 - val_acc:
Epoch 17/20
3305/3305 [==============================] - 2s 665us/step - loss: 0.0195 - acc: 0.9943 - val_loss: 0.1952 - val_acc
Epoch 18/20
3305/3305 [==============================] - 2s 492us/step - loss: 0.0156 - acc: 0.9976 - val_loss: 0.2131 - val_acc
Epoch 19/20
3305/3305 [==============================] - 2s 491us/step - loss: 0.0149 - acc: 0.9970 - val_loss: 0.2556 - val_acc
Epoch 20/20
3305/3305 [==============================] - 2s 487us/step - loss: 0.0128 - acc: 0.9970 - val_loss: 0.2760 - val_acc
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               3037824
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 516
=================================================================
Total params: 3,054,852
Trainable params: 3,054,852
Non-trainable params: 0
_________________________________________________________________
```

In [28]:

```
# ------------------------------------------------------
# 5 模型预测

y_pred_keras = network.predict(X_test_count)

# y_pred_keras[:20]
```

In [29]:

```
# ------------------------------------------------------
# 6 性能评估
print('>>>多分类前馈神经网络性能评估如下...\n')
score = network.evaluate(X_test_count,
                         y_test_cate,
                         batch_size=32)
print('\n>>>评分\n', score)
print()
end = time.time()

estimator_list.append('前馈网络')
score_list.append(score[1])
time_list.append(round(end-start, 2))
```

```
>>>多分类前馈神经网络性能评估如下...

200/200 [==============================] - 0s 298us/step
```

```
>>>评分
 [0.27604406617581845, 0.9]
```

目录

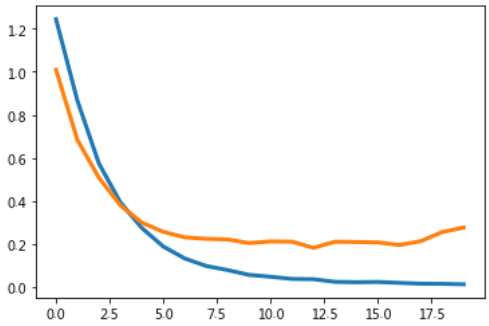
In [30]:

```
# 损失函数情况
import matplotlib.pyplot as plt
%matplotlib inline

train_loss = history.history["loss"]
valid_loss = history.history["val_loss"]
epochs = [i for i in range(len(train_loss))]
plt.plot(epochs, train_loss,linewidth=3.0)
plt.plot(epochs, valid_loss,linewidth=3.0)
```

Out[30]:

```
[<matplotlib.lines.Line2D at 0x7fbd5c305780>]
```

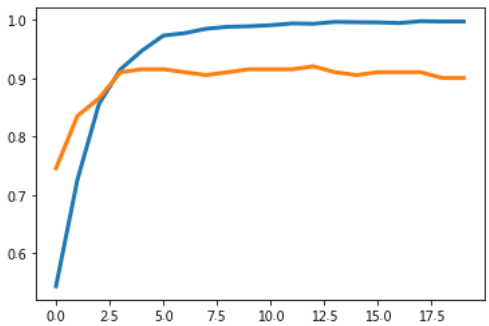


In [31]:

```
# 准确率情况
train_loss = history.history["acc"]
valid_loss = history.history["val_acc"]
epochs = [i for i in range(len(train_loss))]
plt.plot(epochs, train_loss,linewidth=3.0)
plt.plot(epochs, valid_loss,linewidth=3.0)
```

Out[31]:

```
[<matplotlib.lines.Line2D at 0x7fbd0247ee10>]
```



In [32]:

```
# ----------------------------------------------------
# 7 保存/加载模型

# 保存
print('\n>>>你正在进行保存模型操作，请稍候...\n')

network.save('/home/kesci/work/xiaozhi/my_network_model.h5')

print('>>>保存工作已完成...\n')


# 加载和使用
print('>>>你正在加载已经训练好的模型，请稍候...\n')

my_load_model = models.load_model('/home/kesci/work/xiaozhi/my_network_model.h5')

print('>>>你正在使用加载的现成模型进行预测，请稍候...\n')
print('>>>预测部分结果如下...')
```

```
print(">>>预测部分结果如下...")

my_load_model.predict(X_test_count)[:20]
```

```
>>>你正在进行保存模型操作，请稍候...

>>>保存工作已完成...

>>>你正在加载已经训练好的模型，请稍候...

>>>你正在使用加载的现成模型进行预测，请稍候...

>>>预测部分结果如下...
```

```
Out[32]:

array([[5.1700755e-11, 9.9822301e-01, 5.9817724e-12, 1.0217827e-12],
       [5.7793606e-08, 9.7230035e-01, 3.4339887e-08, 4.5136148e-09],
       [9.5060422e-09, 5.1215082e-01, 8.4493550e-08, 9.5746200e-10],
       [1.9871244e-13, 9.9930143e-01, 2.1562925e-12, 1.3205597e-14],
       [1.8191915e-12, 9.9938619e-01, 2.1153847e-12, 4.9101162e-14],
       [7.0857258e-11, 6.6743273e-01, 1.3755715e-10, 1.9664211e-13],
       [1.4808219e-11, 2.6674886e-04, 1.9547663e-08, 2.4381385e-16],
       [2.1480148e-04, 6.2760770e-02, 5.8339193e-04, 1.9946802e-04],
       [1.0268595e-10, 9.2095643e-01, 5.2838917e-08, 1.5940900e-11],
       [7.5692771e-11, 9.8575795e-01, 7.2382700e-10, 6.0663430e-10],
       [3.5654655e-06, 2.7780209e-02, 6.7675983e-06, 6.2545593e-07],
       [5.6649534e-15, 9.9979669e-01, 1.9059852e-14, 6.8455198e-17],
       [2.3525736e-04, 1.1769304e-01, 2.6649828e-03, 2.0936482e-04],
       [3.6169077e-08, 9.7102821e-01, 5.3637911e-10, 1.0601772e-10],
       [1.1997248e-12, 9.8428184e-01, 6.2197027e-12, 1.8859871e-14],
       [9.4651825e-14, 9.9954802e-01, 4.3595604e-13, 6.6097509e-15],
       [4.8878224e-04, 4.5537679e-03, 1.2824073e-03, 1.9483171e-04],
       [6.1444093e-06, 8.1935581e-03, 1.8014919e-03, 9.0278033e-04],
       [4.6505798e-07, 2.8149030e-01, 1.4581896e-06, 4.8493359e-08],
       [2.6251234e-07, 2.9730025e-01, 3.8841034e-05, 1.4241238e-07]],
      dtype=float32)
```

深度学习算法——方法2——LSTM 神经网络

本案例的数据使用 LSTM 跑模型

```
In [33]:

# ----------------------------------------------

# 设置所希望的特征数
feature_num = X_train_count.shape[1]

# 使用单热编码目标向量对标签进行处理

y_train_cate = to_categorical(y_train_le)
y_test_cate = to_categorical(y_test_le)

print(y_train_cate)
```

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

```
In [34]:

# # ----------------------------------------------
# # 1 创建神经网络
# lstm_network = models.Sequential()

# # ----------------------------------------------
# # 2 添加神经层
# lstm_network.add(layers.Embedding(input_dim=feature_num,  # 添加嵌入层
#                                   output_dim=4))

# lstm_network.add(layers.LSTM(units=128))                 # 添加 128 个单元的 LSTM 神经层

# lstm_network.add(layers.Dense(units=4,
#                               activation='sigmoid'))     # 添加 sigmoid 分类激活函数的全连接层

# # ----------------------------------------------
# # 3 编译神经网络
# lstm_network.compile(loss='binary_crossentropy',
#                      optimizer='Adam',
```

```
#                     metrics=['accuracy']
#                     )

# # ------------------------------------------------
# # 4 开始训练模型
# lstm_network.fit(X_train_count,
#                  y_train_cate,
#                  epochs=5,
#                  batch_size=128,
#                  validation_data=(X_test_count, y_test_cate)
#                  )
```

目录

7.算法之间性能比较

In [35]:

```
df = pd.DataFrame()
df['分类器'] = estimator_list
df['准确率'] = score_list
df['消耗时间/s'] = time_list
df
```

Out[35]:

	分类器	准确率	消耗时间/s
0	KNeighborsClassifier	0.670	32.67
1	DecisionTreeClassifier	0.760	38.01
2	MLPClassifier	0.890	213.85
3	BernoulliNB	0.785	1.27
4	GaussianNB	0.890	2.21
5	MultinomialNB	0.905	0.66
6	LogisticRegression	0.890	0.80
7	SVC	0.575	398.70
8	RandomForestClassifier	0.820	6.42
9	AdaBoostClassifier	0.570	149.17
10	LGBMClassifier	0.795	2.09
11	XGBClassifier	0.735	602.94
12	前馈网络	0.900	47.08

综上 DataFrame 展示，结合消耗时间和准确率来看，可以得出以下结论：

在同一训练集和测试集、分类器默认参数设置（都未进行调参）的情况下：

- 综合效果最好的是:

  `MultinomialNB` 多项式朴素贝叶斯分类算法:
  其准确率达到了 `90.5%` 并且所消耗的的时间才 `0.55 s`

- 综合效果最差的是:

  `SVC` 支持向量机
  其准确率才 `0.575` 并且消耗时间高达 `380.72s`

- 准确率最低的是：0.570

  `AdaBoostClassifier` 自适应增强集成学习算法

- 消耗时间最高的是：566.59s

  `XGBClassifier` 集成学习算法

In [ ]:

目录

目
录

目录

和鲸社区公众号