案例三：基于LSTM的电影评论情感分析

1. 案例目的

• 掌握双向LSTM模型；

• 掌握文本预处理;

• 掌握使用双向LSTM模型进行情感预测。

2. 案例内容

情感分析是自然语言处理最基本的应用，情感分类较为简单，实用性也较强。常见的购物网站、电影网站都可以采集到相对高质量的数据集，也很容易给业务领域带来收益。例如，可以结合领域上下文，自动分析特定类型客户对当前产品的意见，可以分主题分用户类型对情感进行分析，以作针对性的处理，甚至基于此进一步推荐产品，提高转化率，带来更高的商业收益。

本案例主要基于双向LSTM进行“段落级”情感分析——IMDB影评情感分类。输入为一个段落，输出为影评是正面还是负面的信息。

3. 案例知识点

• Python语言编程

• 双向LSTM模型

• 文本预处理

• 使用双向LSTM模型进行情感预测

4. 案例时长

共4学时，具体安排如下：

• 文本预处理（1.5学时）

• 训练双向LSTM模型（1.5学时）

• 应用模型（1学时）

5. 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）Tensorflow 2.3.0

3）Keras 2.4.3

**•开发环境与工具：**

1）Spyder 4

6. 案例分析

本案例主要分为以下3部分：

1）数据预处理，为建立模型做准备；

2）建立双向LSTM模型对影评进行情感分类；

3）使用训练好的模型对新数据进行预测。

7. 案例实验过程

IMDB影评情感分类，可分为以下几个步骤：

1、 文本预处理

1.1导入库

1.2 去掉html标签

1.3 读取数据，并存储成list格式

1.4 建立字典

1.5 得到词向量

2、训练双向LSTM模型

2.1 导入库

2.2 设置全局变量

2.3 设置超参数

2.4 得到训练集和测试集

2.5 文本向量化

2.6 构建双向LSTM模型

2.7 编译模型

2.8 训练模型

2.9 评估模型

2.10 保存模型

3、应用模型

3.1 导入库

3.2 全局变量

3.3 全局常量

3.4 加载训练好的模型

3.5 需要预测的评论

3.6 将文本数据向量化

3.7 预测结果

7.1开发准备

在桌面上新建一个目录，名字叫LSTM\_IMDB。本案例中所有的数据集和代码都放在该目录下。

7.1.1 打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至LSTM\_IMDB目录下。

7.1.2 获取数据

本案例需要1个数据集，是aclImdb目录，该目录下存储了IMDB影评数据。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/b55a0b1cbdb74925a24708d6a9621d8b.zip下载数据集。下载后得到aclImdb.zip文件。将下载后的aclImdb.zip文件解压，得到aclImdb目录。将aclImdb目录复制到刚创建的LSTM\_IMDB目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为LSTM\_IMDB。使用Spyder新建3个Python文件，命名为data\_process.py、train.py、test.py。

7.1.4 新建目录

在LSTM\_IMDB目录下，新建一个model目录，用于存放训练好的模型。

至此，整个案例的目录结构图1所示。

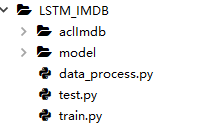


图1 案例的目录结构

7.1.5 了解数据

IMDB影评数据集，包含50000条评论，平均分为训练集（train目录）和测试集（test目录），即训练集（train目录）和测试集（test目录）各有25000条评论，且train和test目录下都分别包含两个子目录：pos目录和neg目录。pos目录下包含了12500条正面的评论，neg目录下包含了12500条负面的评论。每一条评论都存储在一个.txt文档中。数据集目录如下图所示：

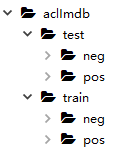


图2 IMDB影评数据集

其中一条影评信息如图3所示。



图3 其中一条影评信息

7.2文本预处理

在正式开始训练模型之前，需要对数据进行预处理，为建立模型做准备。该部分向data\_process.py文件中添加代码，定义所有对文本进行预处理的函数。

7.2.1 导入库

导入所有相关库。

【代码7-2-1】data\_process.py

# 导入库

import os

import re

from keras.preprocessing import sequence

from keras.preprocessing.text import Tokenizer

import numpy as np

import random

7.2.2 去掉html标签

该部分的目的是去掉html里面的特殊符号，将文本内的html标签替换为空字符串，因为数据集中一个文本为一条评论，故构建一个函数统一处理。

【代码7-2-2】data\_process.py

# 去掉html标签

def rm\_tags(text):

re\_tag=re.compile(r'<[^>]+>')

return re\_tag.sub('',text)

7.2.3 读取数据，并存储成list格式

IMDB影评数据集包含train和test两个文件夹，每个文件夹下包含pos和neg两个文件夹，因此可以根据train和test做标签来分别读取不同目录下的数据内容。步骤如下：

（1） 定义一个空列表file\_list，用于存储数据；

（2） 指定正面评价的数据存放路径，将正面评价的所有文件加入（1）定义的列表中；

（3） 指定负面评价的数据存放路径，将负面评价的所有文件加入（1）定义的列表中，因为正面评价和负面评价的文件列表各为12500，现在file\_list的长度为25000，前12500项为正面评价，其余为负面评价；

（4） 定义空列表all\_texts用来存储文本；

（5） 根据文件存储路径读取文件内容，用空字符将每行的数据连接起来，再调用7.2.2节中定义的rm\_tag函数去除html标签，存储到（4）定义的列表all\_texts中；

（6） 为数据集添加标签，1为正面评价，0为负面评价，生成与file\_list对应长度的标签列表all\_labels，并将其转换为网络需要的Numpy类型；

（7） 打乱数据顺序；

（8） 返回标签和文本内容，此时，all\_texts和label都是长度为25000的列表。

【代码7-2-3】data\_process.py

# 读取数据，并存储成list格式

def read\_files(path,filetype):

#定义一个空的列表，用于存放数据

file\_list=[]

# 读取正面评价

positive\_path=path+filetype+'/pos/'

#将正面评价的所有文件列表加入file\_list

for f in os.listdir(positive\_path):

file\_list+=[positive\_path+f]

# 读取负面评价

negative\_path=path+filetype+'/neg/'

for f in os.listdir(negative\_path):

file\_list+=[negative\_path+f]

# 定义空列表用来存储文本

all\_texts=[]

# 存储去除特殊符号之后的文本

for fi in file\_list:

with open(fi,encoding='utf8') as file\_input:

all\_texts+=[rm\_tags(' '.join(file\_input.readlines()))]

# 为数据集添加标签，1为正面评价，0为负面评价

all\_labels=([1]\*12500+[0]\*12500)

all\_labels = np.array(all\_labels)

# 打乱数据集顺序

random.seed(42)

random.shuffle(all\_texts)

random.seed(42)

random.shuffle(all\_labels)

return all\_texts, all\_labels

7.2.4 建立字典

神经网络无法直接处理非数字类型的数据，我们需要将单词转换为对应的数字类型，然后在输入的时候输入数字，进行运算，这一步会遍历所有文本内容，依照出现频率对单词进行排序，然后指定个数，对前2000或其他数字的单词建立一对一字典，进行数字转换，其余的则不在字典中。

【代码7-2-4】data\_process.py

# 建立字典

# @Param: num\_words: 构建字典使用的单词数

def get\_token(num\_words, x\_train):

token=Tokenizer(num\_words)

# 读取所有文章，按照单词出现的次数进行排序，排序前num\_words的单词会列到字典中

token.fit\_on\_texts(x\_train)

return token

7.2.5 得到词向量

该部分的作用是对文本内容进行处理，并转换为统一长度的数字列表。

循环神经网络可以接受不同长度的文本，但是，在我们模型训练环节，我们一般都会进行批量训练，这就要求我们将输入格式统一，即将文本的长度统一。文本长度统一的过程可以看作是“去长补短”的操作。

“去长”：是将文本前面多余的部分去掉；“补短”：是在文本前面添加元素0，循环神经网络具有一定的记忆能力，在文本信息提取的时候，是获取最后一个序列的输出作为文本的信息表达，此输出在理论上涵盖了文本所有的信息，但从记忆的衰减性角度分析，越接近最后序列的信息会被更多的保存，所以在“补短”处理过程中，将0元素添加到文本序列的前面，主要作用就是为了在符合记忆衰减规律前提下保持文本长度的统一。

【代码7-2-5】data\_process.py

# 得到词向量

# @Param: x\_data: x\_train或x\_test

# @Param: maxlen: 词向量维度

def get\_word\_vector(num\_words, maxlen, x\_train, x\_data):

token = get\_token(num\_words, x\_train)

# 将文本内容转化为数字列表

x\_data\_seq=token.texts\_to\_sequences(x\_data)

# 统一文本长度，比maxlen短的在开头补0，比maxlen长的截掉开头的数据

x\_data\_to\_return =sequence.pad\_sequences(x\_data\_seq,maxlen)

return x\_data\_to\_return

7.3训练双向LSTM模型

该部分的作用是构建双向LSTM模型，训练模型，评估模型，保存训练好的模型。向train.py文件中添加代码。

7.3.1 导入库

导入所有相关库。

【代码7-3-1】train.py

# 导入库

from keras.models import Sequential

from keras.layers.core import Dense,Dropout

from keras.layers.embeddings import Embedding

from keras.layers.recurrent import LSTM

from keras.layers.wrappers import Bidirectional

from data\_process import read\_files

from data\_process import get\_word\_vector

执行代码块。

7.3.2 设置全局变量

这些变量用于指定数据源的路径、模型保存的路径。设置为全局变量比较方便使用。

【代码7-3-2】train.py

# 全局变量

outputpath = 'model/final.model' # 模型保存路径

path='aclImdb/' # 指定数据集存放目录

执行代码块。

7.3.3 设置超参数

这些超参数都是训练模型的一些指定参数，如指定词向量的维度、数据字典中单词的数量，一批加载多少数据，训练多少个EPOCH。如果模型性能不好或者出现其他问题，我们需要调整这些参数。在案例中，EPOCH的数量不足，会导致模型欠拟合。

【代码7-3-3】train.py

# 超参数

maxlen=128 # 词向量维度

num\_words=2000 # 数据字典中单词的数量

BATCH\_SIZE=64

EPOCHS=20

执行代码块。

7.3.4 得到训练集和测试集

调用read\_files函数，得到训练集和测试集。

【代码7-3-4】train.py

# 得到训练集和测试集

x\_train\_temp, y\_train = read\_files(path,'train')

x\_test\_temp, y\_test = read\_files(path,'test')

执行代码块。

从现在开始，建议打开Spyder的Variable explorer面板，实时查看变量的信息。

此时，x\_test\_temp和x\_train\_temp两个变量中分别存储了测试集和训练集的各25000条文本数据，y\_test和y\_train变量中存储了测试集和训练集对应的标签信息。

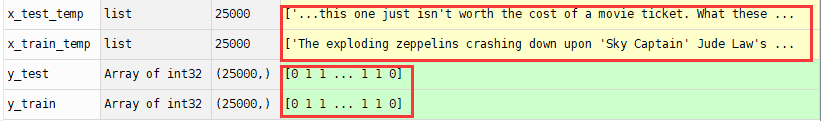


图4 x\_test\_temp、x\_train\_temp、y\_test、y\_train变量

7.3.5 文本向量化

调用7.2.5中定义的get\_word\_vector函数将x\_train和x\_test文本向量化。

【代码7-3-5】train.py

# x\_train和x\_test文本向量化

x\_train = get\_word\_vector(num\_words, maxlen, x\_train\_temp, x\_train\_temp)

x\_test = get\_word\_vector(num\_words, maxlen, x\_train\_temp, x\_test\_temp)

执行代码块。

运行代码后，将文本向量化后的信息分别保存在x\_train和x\_test变量中，以x\_train变量为例，效果如图5所示，每一条评论信息对应128个数字（由128个单词数字化得到）。

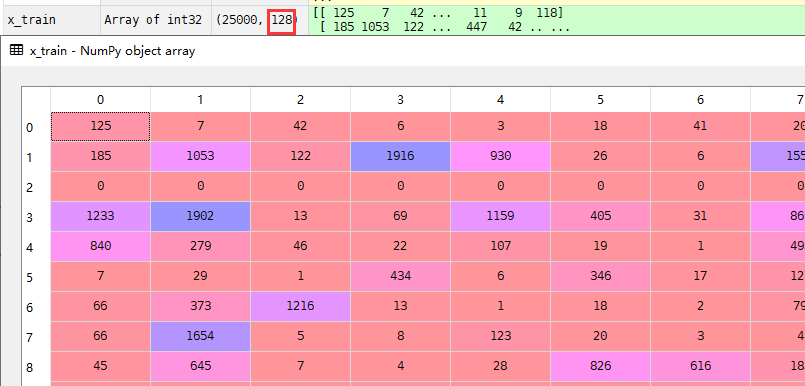


图5 x\_train变量

7.3.6 构建双向LSTM模型

该部分的作用是构建双向LSTM模型。其中，Embedding层的主要作用是将每个token（字）嵌入成为固定长度的词向量。

【代码7-3-6】train.py

# 构建一个空模型

model=Sequential()

# 添加嵌入层，这里输入形状为[None,128]，输出形状为[None,128,32]

model.add(Embedding(output\_dim=32,

input\_dim=num\_words,

input\_length=maxlen))

# 加入正则化层,不影响数据形状

model.add(Dropout(0.2))

# LSTM中return\_sequences默认为False，不返回一个序列，只返回最后一个输出；

# 这里使用Bidirectional进行包装，则为双向LSTM，因为是双向的，所以输出为[None,64]；如果return\_sequences为true，则输出为[None, 128, 64]

model.add(Bidirectional(LSTM(units=32,return\_sequences=False)))

# 因为输出已经算一维了，无需平坦层，这一层输出为[None,256]

model.add(Dense(units=256,

activation='relu'))

# 加入正则化层

model.add(Dropout(0.2))

# 输出形状为[None,1]，因为最终是二分类问题，所以用sigmoid激活函数

model.add(Dense(units=1,activation='sigmoid'))

model.summary()

执行代码块。

网络信息输出如图6所示。

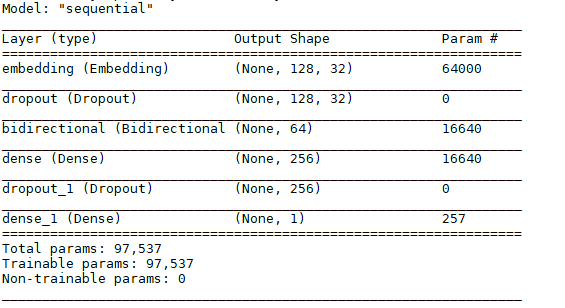


图6 双向LSTM模型

7.3.7 编译模型

编译模型，优化算法选择Adam，loss值选择二元交叉熵，评估指标选择accuracy。

【代码7-3-7】train.py

# 编译模型

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

执行代码块。

7.3.8 训练模型

该部分的目的是训练双向LSTM模型。根据代码中设置的超参数EPOCHS，我们需要训练20个epoch。且训练过程中使用了验证集，因此可以实时评估模型性能。模型训练信息存储到train\_history变量中。

在Intel(R) Core(TM) i7-8750H [CPU@2.20GHz](mailto:CPU@2.20GHz)CPU，8GBRAM的计算机上，每个epoch的运行时间大约为14s，20个epoch大约需要5分钟。不需要GPU。

由于机器性能不同，运行时间有所不同，若训练速度过慢，建议同学们将减小EPOCHS参数的设置，仅用于查看案例效果即可。

【代码7-3-8】train.py

# 训练模型

train\_history=model.fit(x\_train,y\_train,batch\_size=BATCH\_SIZE,

epochs=EPOCHS,verbose=1,

validation\_split=0.2)

执行代码块。

训练过程如图7所示。训练完成后，在验证集上的accuracy为83.60%左右，性能还是不错的。

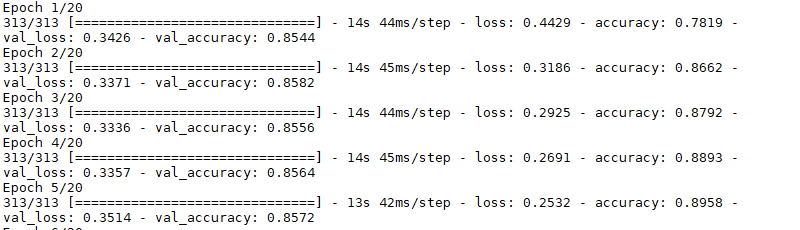


图7 训练过程图

7.3.9 评估模型

模型训练完成后，在测试集上评估模型性能。

【代码7-3-9】train.py

#评估模型

scores=model.evaluate(x\_test,y\_test,verbose=1)

scores[1]

执行代码块。

测试评估结果如图8所示，accuracy为83.42%，性能还是比较令人满意的。



图8 测试评估结果

注意：由于初始权重会随机初始化，所以同学们的训练结果不会和本案例一模一样。

7.3.10 保存模型

如果同学们没有完成模型的训练，请不要执行这个步骤。

保存模型是非常重要的一个步骤。下次使用该模型时，只需用1个函数加载进来即可。

【代码7-3-10】train.py

#保存模型

model.save(outputpath)

执行代码块。

模型会保存在model目录下。

7.4应用模型

加载训练好的模型，对新数据进行预测。向test.py文件中添加代码。

7.4.1 导入库

把所有用到的库导入。

【代码7-4-1】test.py

# 导入包

from data\_process import read\_files

from keras.models import load\_model

from data\_process import get\_word\_vector

执行代码块。

7.4.2 全局变量

这些变量用于指定数据源的路径、加载模型的路径。设置为全局变量比较方便使用。

【代码7-4-2】test.py

# 全局变量

outputpath = 'model/final.model'# 模型路径

path='aclImdb/' # 指定数据集存放目录

执行代码块。

7.4.3 全局常量

因为下面还需要将文本数据向量化，仍然需要传入这两个参数。

【代码7-4-3】test.py

# 全局常量

maxlen=128 # 词向量维度

num\_words=2000 # 数据字典中单词的数量

执行代码块。

7.4.4 加载训练好的模型

将训练好的模型加载进来，保存在model变量中。

【代码7-4-4】test.py

# 加载训练好的模型

model = load\_model(outputpath)

执行代码块。

7.4.5 需要预测的评论

定义空列表，并将需要预测的评论加入到列表中。

【代码7-4-5】test.py

#需要预测的评论

test\_text=[]

test\_text.append('This movie is very good,i like it very much')

test\_text.append('This movie is bad')

test\_text.append('I do not like this movie')

test\_text.append('This movie is so fantastic')

test\_text.append('This movie is bad, but the movie theater is very bad')

执行代码块。

同学们也可以自己添加评论，进行预测。

7.4.6 将文本数据向量化

接下来，调用get\_word\_vector函数，将文本数据向量化。

【代码7-4-6】test.py

# 将文本数据向量化

x\_train\_temp, y\_train = read\_files(path,'train')

x\_test = get\_word\_vector(num\_words, maxlen, x\_train\_temp, test\_text)

执行代码块。

7.4.7 预测结果

使用训练好的模型对新数据进行预测。

# 预测结果

pred\_prob = model.predict(x\_test)

pred = ['positive' if i >=0.5 else 'negative' for i in pred\_prob]

for i, item in enumerate(test\_text):

print('%s： %s\n\n' %(item, pred[i]))

执行代码块。

预测结果如图9所示，训练的模型对5条评论的情感预测非常准确。

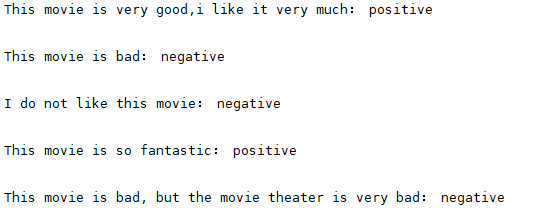


图9 预测结果

8. 案例代码

【案例代码】data\_process.py

# -\*- coding: utf-8 -\*-

import os

import re

from keras.preprocessing import sequence

from keras.preprocessing.text import Tokenizer

import numpy as np

import random

# 去掉html标签

def rm\_tags(text):

re\_tag=re.compile(r'<[^>]+>')

return re\_tag.sub('',text)

# 读取数据，并存储成list格式

def read\_files(path,filetype):

#定义一个空的列表，用于存放数据

file\_list=[]

# 读取正面评价

positive\_path=path+filetype+'/pos/'

#将正面评价的所有文件列表加入file\_list

for f in os.listdir(positive\_path):

file\_list+=[positive\_path+f]

# 读取负面评价

negative\_path=path+filetype+'/neg/'

for f in os.listdir(negative\_path):

file\_list+=[negative\_path+f]

# 定义空列表用来存储文本

all\_texts=[]

# 存储去除特殊符号之后的文本

for fi in file\_list:

with open(fi,encoding='utf8') as file\_input:

all\_texts+=[rm\_tags(' '.join(file\_input.readlines()))]

# 为数据集添加标签，1为正面评价，0为负面评价

all\_labels=([1]\*12500+[0]\*12500)

all\_labels = np.array(all\_labels)

# 打乱数据集顺序

random.seed(42)

random.shuffle(all\_texts)

random.seed(42)

random.shuffle(all\_labels)

return all\_texts, all\_labels

# 建立字典

# @Param: num\_words: 构建字典使用的单词数

def get\_token(num\_words, x\_train):

token=Tokenizer(num\_words)

# 读取所有文章，按照单词出现的次数进行排序，排序前2000的单词会列到字典中

token.fit\_on\_texts(x\_train)

return token

# 得到词向量

# @Param: x\_data: x\_train或x\_test

# @Param: maxlen: 词向量维度

def get\_word\_vector(num\_words, maxlen, x\_train, x\_data):

token = get\_token(num\_words, x\_train)

# 将文本内容转化为数字列表

x\_data\_seq=token.texts\_to\_sequences(x\_data)

# 统一文本长度，比maxlen短的在开头补0，比maxlen长的截掉开头的数据

x\_data\_to\_return =sequence.pad\_sequences(x\_data\_seq,maxlen)

return x\_data\_to\_return

【案例代码】train.py

# -\*- coding: utf-8 -\*-

# 导入库

from keras.models import Sequential

from keras.layers.core import Dense,Dropout

from keras.layers.embeddings import Embedding

from keras.layers.recurrent import LSTM

from keras.layers.wrappers import Bidirectional

from data\_process import read\_files

from data\_process import get\_word\_vector

# 全局变量

outputpath = 'model/final.model' # 模型保存路径

path='aclImdb/' # 指定数据集存放目录

# 超参数

maxlen=128 # 词向量维度

num\_words=2000 # 数据字典中单词的数量

BATCH\_SIZE=64

EPOCHS=20

# 得到训练集和测试集

x\_train\_temp, y\_train = read\_files(path,'train')

x\_test\_temp, y\_test = read\_files(path,'test')

# x\_train和x\_test文本向量化

x\_train = get\_word\_vector(num\_words, maxlen, x\_train\_temp, x\_train\_temp)

x\_test = get\_word\_vector(num\_words, maxlen, x\_train\_temp, x\_test\_temp)

# 构建一个空模型

model=Sequential()

# 添加嵌入层，这里输入形状为[None,128]，输出形状为[None,128,32]

model.add(Embedding(output\_dim=32,

input\_dim=num\_words,

input\_length=maxlen))

# 加入正则化层,不影响数据形状

model.add(Dropout(0.2))

# LSTM中return\_sequences默认为False，不返回一个序列，只返回最后一个输出；

# 这里使用Bidirectional进行包装，则为双向LSTM，因为是双向的，所以输出为[None,64]；如果return\_sequences为true，则输出为[None, 128, 64]

model.add(Bidirectional(LSTM(units=32,return\_sequences=False)))

# 因为输出已经算一维了，无需平坦层，这一层输出为[None,256]

model.add(Dense(units=256,

activation='relu'))

# 加入正则化层

model.add(Dropout(0.2))

# 输出形状为[None,1]，因为最终是二分类问题，所以用sigmoid激活函数

model.add(Dense(units=1,activation='sigmoid'))

model.summary()

# 编译模型

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# 训练模型

train\_history=model.fit(x\_train,y\_train,batch\_size=BATCH\_SIZE,

epochs=EPOCHS,verbose=1,

validation\_split=0.2)

#评估模型

scores=model.evaluate(x\_test,y\_test,verbose=1)

scores[1]

#保存模型

model.save(outputpath)

【案例代码】test.py

# -\*- coding: utf-8 -\*-

# 导入包

from data\_process import read\_files

from keras.models import load\_model

from data\_process import get\_word\_vector

# 全局变量

outputpath = 'model/final.model'# 模型路径

path='aclImdb/' # 指定数据集存放目录

# 全局常量

maxlen=128 # 词向量维度

num\_words=2000 # 数据字典中单词的数量

# 加载预训练模型

model = load\_model(outputpath)

#需要预测的评论

test\_text=[]

test\_text.append('This movie is very good,i like it very much')

test\_text.append('This movie is bad')

test\_text.append('I do not like this movie')

test\_text.append('This movie is so fantastic')

test\_text.append('This movie is bad, but the movie theater is very bad')

# 将文本数据向量化

x\_train\_temp, y\_train = read\_files(path,'train')

x\_test = get\_word\_vector(num\_words, maxlen, x\_train\_temp, test\_text)

# 预测结果

pred\_prob = model.predict(x\_test)

pred = ['positive' if i >=0.5 else 'negative' for i in pred\_prob]

for i, item in enumerate(test\_text):

print('%s： %s\n\n' %(item, pred[i]))