第二次编程作业报告

热合玛·阿不力克木

1700017843

一、实验目标

实现一种基于 RNN 的模型以及一种基于 CNN 的模 型进行文本分类任务，具体为对英文篇章级文本进行分类。评估指标是 Accuracy。由自己实现相关的评估指标。

二、实验环境

tensorflow 1.13.1

keras 2.2.4

python 3.6

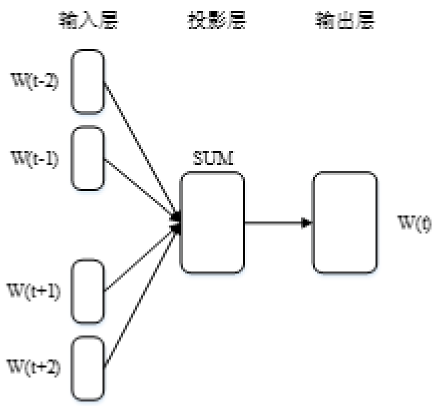
（jupyter notebook）

三、实验方法及模型介绍

对文本用jieba进行分词，用Word2Vec模型进行词向量转化，RNN模型用的是带有Attention机制的LSTM模型，CNN模型运用了TextCNN模型。

Word2Vec模型：

Word2vec模型是谷歌在2013年提出的，其能将词语转化成具有语义信息的空间词向量，从而能将一段文本转化成一段有语义的向量，进行多种自然语言处理任务。在同义词挖掘中，可以利用其将词语转化成语义信息的空间词向量这一特点，计算两两词语中的空间距离，计算向量空间距离的方法有很多，例如：欧氏距离，余弦距离，编辑距离等，本文采用欧氏距离计算词语空间距离。Word2Vec模型具有训练快速，内存消耗低的特点。Word2vec模型中包含CBOW模型和Skip-gram模型。本次实验中，我运用的是CBOW模型。



CBOW模型结构图

CBOW模型根据句子中前后若干个词语来预测中间词语是哪个词语。如上图所示，CBOW模型总共有三层，第一层为输入层，将词语转化成对应的向量，第二层为投影层，投影层将输入的词向量进行求和，第三层为输出层，输出最可能的预测的词语。CBOW模型目标在于最大化似然函数：

$$\tau = \sum_{w \epsilon C} logp(w|content(w))$$

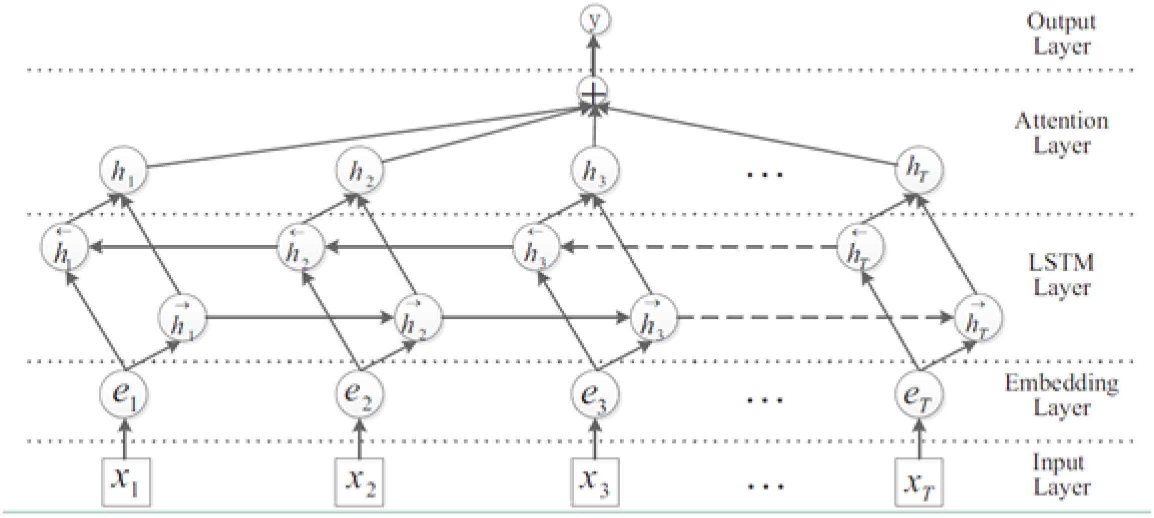
其中w为语料库C中的任意一个词语。

本文采用CBOW模型来训练词向量，训练时使用负采样训练方式，节省更多的空间，词向量维度为128维。通过将文本转化为词向量之后就可以输入模型进行训练，迭代次数为10。

RNN模型：BILSTM+Attention机制

模型总体结构：

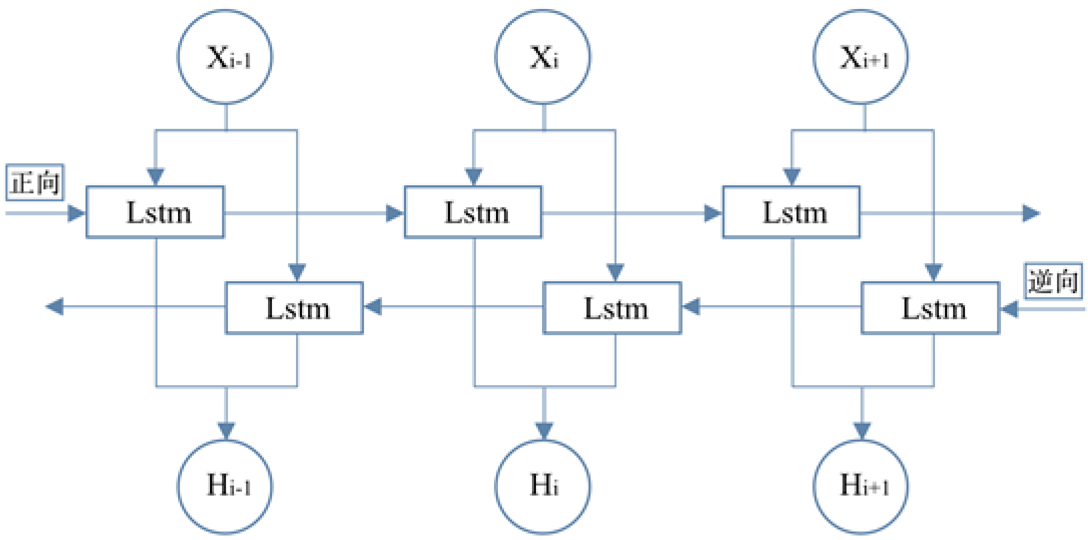
本文的模型结构如下图所示，将文本分词后输入到输入层，将训练好的word2vec词向量输入到词嵌入矩阵中,通过词嵌入矩阵将句子转换成带有语义的向量形式，通过双向LSTM层对句子进行编码，通过Attention层计算每个词语的相似度。最后通过softmax激活函数得出各个类别的概率。



模型总体结构

BILSTM：

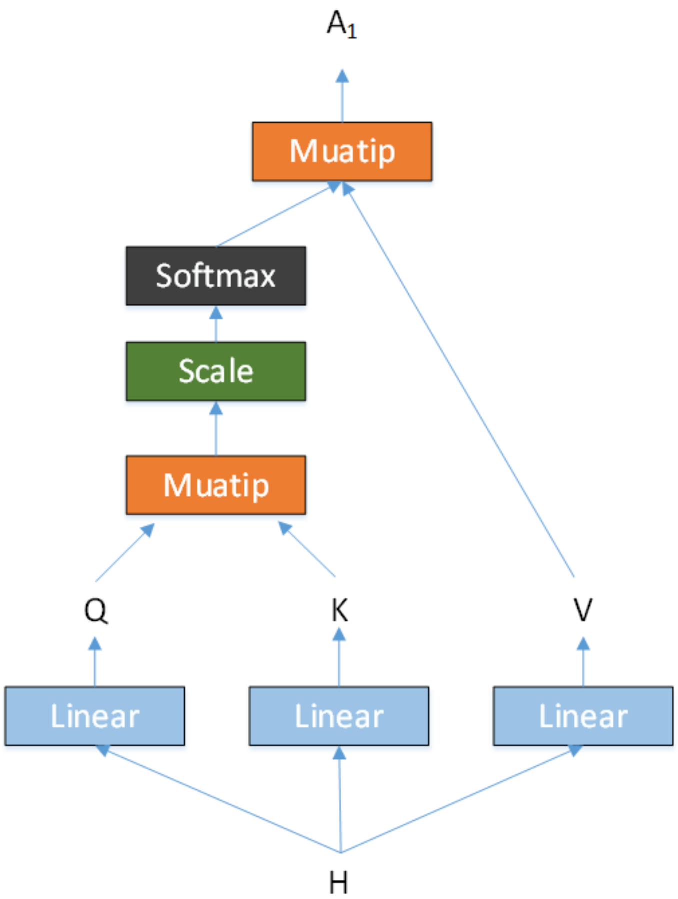
BILSTM模型[1]可以捕捉文本的双向信息流，本文将文本输入向量输入到BILSTM模型中得到特征向量。BILSTM模型结构如下图所示，Xi-1到Xi+1为输入向量，输入向量分别从正向和反向分别输入到模型中，得到特征向量Hi-1到Hi+1。



BILSTM模型结构图

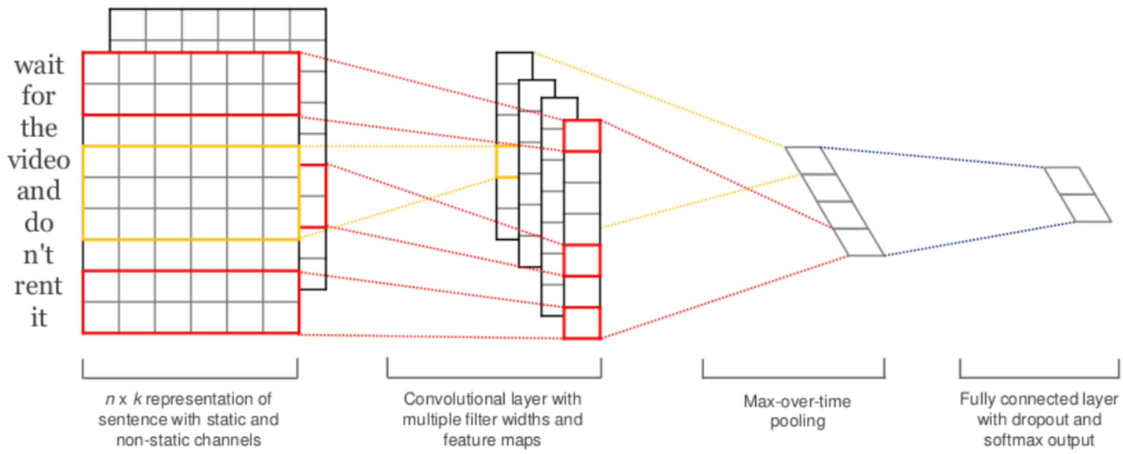
Self Attention机制：

Self Attention[2]机制如图3.3所示，Self Attention机制的具体操作如下：H向量经过三个不同的全连接层得到Q，K，V三个向量。Q和KT做矩阵乘法之后，得到向量Q*KT，其表示词语与其它词语的相关程度，对*QKT做标准化后，输入Softmax激活函数后得到词语之间的相关程度向量，将相关度程度向量与V做点乘得到向量A1。即模型在编码时每个词语都会考虑到句子中其他词语的语义比重，从而有着强大的编码能力。可以提高分类的效果。



CNN模型：TextCNN模型

TextCNN模型能够通过卷积神经网络的方式对向量进行局部抽取信息，进行编码。



网络结构

输入层为文本矩阵，卷积层使用不同的卷积核，卷积核的宽度和词向量的长度一致，每个卷积核获得一列feature map。每个feature map通过max-pooling都会得到一个特征值，这个操作也使得TextCNN能处理不同长度的文本。全连接层的输入为池化操作后形成的一维向量，经过激活函数输出，再加上Dropout层防止过拟合。并在全连接层上添加l2正则化参数。将全连接层的输出使用softmax函数，获取文本分到不同类别的概率。

四、实验步骤及代码

1.读取数据集

这里为了后续操作，对标题和内容进行了连接，生成了新的content列，并且为了方便处理数据，将dev.csv里读取的数据接在了train.csv中读取的数据后面。

```
import pandas as pd
df=pd.read_csv("train.csv")
df_val=pd.read_csv("dev.csv")
```

```
In [2]: df
```

Out[2]:

	Class Index	Title	Description
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...
108263	4	Mobile phones: An ear full of worms	They #39;re coming to mobile phones - those na...

108264 rows × 3 columns

```
# 连接标题和内容
df["content"]=df["Title"]+" "+df["Description"]
df_val["content"]=df_val["Title"]+" "+df_val["Description"]
df_val_num=df_val.shape[0] #dev的元素个数
# 拼接train和dev（但训练时还是用train，这里只是为了方便处理数据）
df=pd.concat([df,df_val],axis=0)
df=df[["Class Index","content"]]
df["category"]=df["Class Index"]
```

```
In [5]: df
```

Out[5]:

	Class Index	Title	Description	content
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...	Wall St. Bears Claw Back Into the Black (Reute...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...	Carlyle Looks Toward Commercial Aerospace (Reu...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...	Oil and Economy Cloud Stocks' Outlook (Reuters...

2.分词

采用jieba工具包对文本进行分词。

```
num_classes = len(df["category"].unique())# 获得标签数
#分词
import jieba
sentence=[[j.lower() for j in i.split(" ")] for i in df["content"]]
```

查看结果:

```
[12]: print (sentence[0])
```

```
['wall', 'st.', 'bears', 'claw', 'back', 'into', 'the', 'black', '(reuters)', 'reuters', '-', 'short-sellers,', 'wall', "street's", 'dwindli
ng\\band', 'of', 'ultra-cynics,', 'are', 'seeing', 'green', 'again.']
```

3.训练word2vec词向量 size为词向量长度，迭代次数为10

```
import pandas as pd
import gensim
w2v_model = gensim.models.Word2Vec(sentence, size=128, iter=10, min_count=0)
word_vectors = w2v_model.wv
w2v_model.save("w2v")
w2v_model=gensim.models.Word2Vec.load("w2v")
```

4.导入实现神经网络必要的包

```
from keras.layers import *
import numpy as np
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers.merge import concatenate
from keras.layers.embeddings import Embedding
from keras.layers.normalization import BatchNormalization
from keras.models import Model
from keras import backend as K
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import StratifiedKFold
from keras.utils.np_utils import to_categorical
from sklearn.metrics import f1_score, classification_report
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

5.将文本转化成向量，将标签onehot，对文本向量以最大长度补零

```
x_train=sentence
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train)  #统计每个词对应的数字，以便于将文本转化成向量
train_sequence = tokenizer.texts_to_sequences(x_train)#将所有的文本转化成向量
MAX_SEQUENCE_LENGTH=128 #最大长度
EMBEDDING_DIM = 128 #向量维度
y_train =df["category"]
y_train = to_categorical(y_train)  #将标签 one-hot
y_train = y_train.astype(np.int32)
word_index = tokenizer.word_index
#print('Found %s unique tokens.' % len(word_index))
train_pad = pad_sequences(train_sequence, maxlen=MAX_SEQUENCE_LENGTH) #将每条文本按
照最大长度补0
```

6.统计每个单词应该对应哪一条向量

```
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM),
dtype=np.float32)
not_in_model = 0
in_model = 0
embedding_max_value = 0
embedding_min_value = 1
not_words = []

for word, i in word_index.items():
    if word in w2v_model:
        in_model += 1
        embedding_matrix[i] = np.array(w2v_model[word])
        embedding_max_value = max(np.max(embedding_matrix[i]),
embedding_max_value)
        embedding_min_value = min(np.min(embedding_matrix[i]),
embedding_min_value)
    else:
        not_in_model += 1
        not_words.append(word)
```

7.用keras定义一个词嵌入层，并把刚才的矩阵加进去

```
embed = Embedding(len(word_index) + 1, EMBEDDING_DIM, weights=
[embedding_matrix], input_length=MAX_SEQUENCE_LENGTH,
                  trainable=True)  #定义一个词嵌入层,将句子转化成对应的向量
```

8.把前面合并的验证集和训练集再划分开

```
# train_data, val_data, train_y, val_y = train_test_split(train_pad, y_train,
test_size=0.2, random_state=43)
train_data=train_pad[:-df_val_num]
train_y=y_train[:-df_val_num]
val_data=train_pad[-df_val_num:]
val_y=y_train[-df_val_num:]
```

9.定义textcnn模型

```
def get_cnnmodel(embedding, class_num=5):
    inputs_sentence = Input(shape=(MAX_SEQUENCE_LENGTH,))#设置输入向量维度
    sentence =(embedding(inputs_sentence))#定义词嵌入层
    con=Conv1D(256, 5, padding='same')(sentence)
    maxp=MaxPooling1D(3, 3, padding='same')(con)
    con=Conv1D(128, 5, padding='same')(maxp)
    maxp=MaxPooling1D(3, 3, padding='same')(con)
    con=Conv1D(64, 3, padding='same')(maxp)
    fla=Flatten()(con)
    drop=Dropout(0.1)(fla)
    bn=BatchNormalization()(drop)
    ds=Dense(256, activation='relu')(bn)
    dp=Dropout(0.1)(ds)
    output = Dense(class_num, activation='softmax')(dp)#softmax层
    model = Model(inputs=[inputs_sentence], outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy'])#定义损失函数，优化器，评分标准
```

```
    model.summary()
    return model
# model = get_cnnmodel(embed)
```

10.训练cnn模型

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
model = get_cnnmodel(embed)
callbacks = [EarlyStopping(monitor='val_acc', min_delta=0.001, patience=10),
             ModelCheckpoint("textcnn.hdf5", monitor='val_acc',
                             mode='max', verbose=0, save_best_only=True)]
#设置模型提前停止,停止的条件是验证集val_acc两轮已经不增加,保存验证集val_acc最大的那个模型,名称为new_cnn.hdf5
history=model.fit(train_data,train_y, batch_size=16, epochs=5,
callbacks=callbacks,validation_data=(val_data,val_y))
```

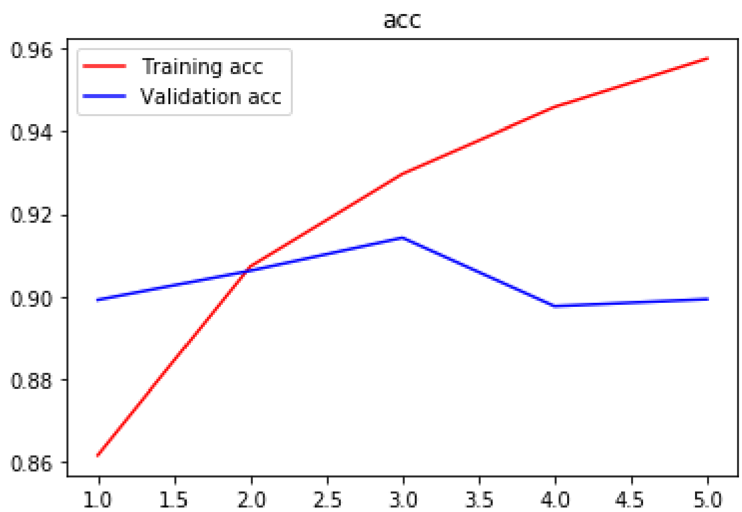
```
Total params: 20,919,621
Trainable params: 20,917,701
Non-trainable params: 1,920
_________________________________________________________________
Train on 108264 samples, validate on 11736 samples
Epoch 1/5
108264/108264 [==============================] - 102s 945us/step - loss: 0.4105 - acc: 0.8615 - val_loss: 0.2995 - val_acc: 0.8991
Epoch 2/5
108264/108264 [==============================] - 98s 909us/step - loss: 0.2786 - acc: 0.9072 - val_loss: 0.2715 - val_acc: 0.9062
Epoch 3/5
108264/108264 [==============================] - 99s 911us/step - loss: 0.2133 - acc: 0.9297 - val_loss: 0.2601 - val_acc: 0.9142
Epoch 4/5
108264/108264 [==============================] - 98s 909us/step - loss: 0.1659 - acc: 0.9458 - val_loss: 0.3044 - val_acc: 0.8976
Epoch 5/5
108264/108264 [==============================] - 99s 911us/step - loss: 0.1285 - acc: 0.9576 - val_loss: 0.3303 - val_acc: 0.8993
```

11.加载最优的模型并且测试验证集

```
from tensorflow.keras.models import load_model
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
test_data=val_data
test_y=val_y
lstm_attention= load_model("textcnn.hdf5")
testpre=lstm_attention.predict([test_data])
tpre=np.argmax(testpre,axis=1)
testy=np.argmax(test_y,axis=1)


from sklearn.metrics import classification_report
print (classification_report(testy,tpre,digits=4))
```

```
              precision    recall  f1-score   support

           1     0.9287    0.8965    0.9123      2802
           2     0.9592    0.9866    0.9727      2980
           3     0.9003    0.8701    0.8850      3103
           4     0.8686    0.9039    0.8859      2851

   micro avg     0.9142    0.9142    0.9142     11736
   macro avg     0.9142    0.9143    0.9140     11736
weighted avg     0.9143    0.9142    0.9140     11736
```

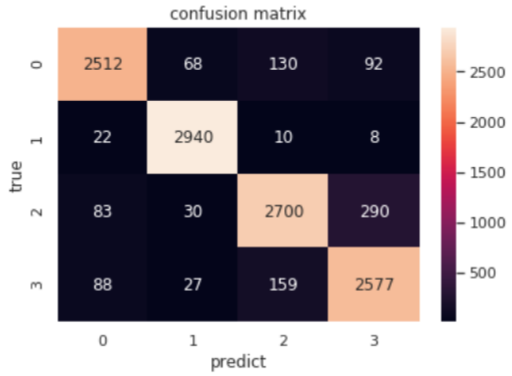
```python
import matplotlib.pyplot as plt
val_loss = history.history['val_loss']
loss = history.history['loss']
epochs = range(1, len(loss ) + 1)
plt.title('Loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')
plt.legend()
plt.show()

plt.cla()
val_loss = history.history['val_acc']
loss = history.history['acc']
epochs = range(1, len(loss ) + 1)

plt.title('acc')
plt.plot(epochs, loss, 'red', label='Training acc')
plt.plot(epochs, val_loss, 'blue', label='Validation acc')
plt.legend()
plt.show()
```



12.打印混淆矩阵和各个类别的准确率

```
def 给测试集输出指标(val_data):
    testpre=lstm_attention.predict([test_data])
    ypre=np.argmax(testpre,axis=1)
    ytrue=np.argmax(test_y,axis=1)
    def leibie_acc(test_pre,y_test):
        test_pre=np.array(test_pre)
        y_test=np.array(y_test)

        print ("准确率:",sum((test_pre==y_test).astype(int))/len(y_test))
        for i in set(list(y_test)):

            tmp=
(test_pre[np.where(y_test==i)]==y_test[np.where(y_test==i)]).astype(int)
            acc=sum(tmp)/len(tmp)
            print (i,"类别准确率为:",acc,"数量为: ",len(tmp))
        return


    import seaborn as sns
    from sklearn.metrics import confusion_matrix
    import matplotlib.pyplot as plt

    def 混洗矩阵(valp,valy):
        sns.set()
        np.set_printoptions(suppress=True)
        f,ax=plt.subplots()
        valp=np.array(valp)
        valy=np.array(valy)
        C2= confusion_matrix(valy , valp,
labels=list(range(1,len(set(df["category"]))+1)))
        print(C2) #打印出来看看
        sns.heatmap(C2,annot=True,ax=ax,fmt='.20g') #画热力图

        ax.set_title('confusion matrix') #标题
        ax.set_xlabel('predict') #x轴
        ax.set_ylabel('true') #y轴

    def 评价指标(val_data,val_y):
    #    valpre=model_load.predict(val_data)
        leibie_acc(val_data,val_y)
        混洗矩阵(val_data,val_y)

    评价指标(ypre,ytrue)
给测试集输出指标(test_data)
```

```
准确率: 0.9141956373551465
1 类别准确率为: 0.8965024982155603 数量为:  2802
2 类别准确率为: 0.9865771812080537 数量为:  2980
3 类别准确率为: 0.8701256848211408 数量为:  3103
4 类别准确率为: 0.9038933707471063 数量为:  2851
[[2512   68  130   92]
 [  22 2940   10    8]
 [  83   30 2700  290]
 [  88   27  159 2577]]
```



13.LSTM+Attention模型

```
def attention_3d_block(inputs):

    input_dim = int(inputs.shape[2])

    a = Permute((2, 1))(inputs)
    a = Reshape((input_dim, train_data.shape[1]))(a) # this line is not useful.
It's just to know which dimension is what.
    a = Dense(train_data.shape[1], activation='softmax')(a)
    a_probs = Permute((2, 1), name='attention_vec')(a)
    output_attention_mul = Multiply()([inputs, a_probs])
    return output_attention_mul

def get_lstmmodel(embedding, class_num=len(set(df["category"]))):
    inputs_sentence = Input(shape=(MAX_SEQUENCE_LENGTH,))  #设置输入向量维度
    sentence =(embedding(inputs_sentence))  #定义词嵌入层
    context1 = Bidirectional(LSTM(64, return_sequences=True))(sentence)  # 双向
lstm层,lstm神经元维度为64
    atten = attention_3d_block(context1) #给lstm1层加上注意力机制
    atten = Flatten()(atten)
    x = Dense(100, activation='relu')(atten)# 全连接层,全连接层神经元维度为100
#     x = Dense(100, activation='relu')(x)# 全连接层,全连接层神经元维度为100
    output = Dense(5, activation='softmax')(atten)  #softmax层
    model = Model(inputs=[inputs_sentence], outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy']) #定义损失函数，优化器，评分标准
    model.summary()
    return model

model = get_lstmmodel(embed)
```

```
___________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
===================================================================================================
input_6 (InputLayer)            (None, 128)          0
___________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 128, 128)     20315776    input_6[0][0]
___________________________________________________________________________________________________
bidirectional_5 (Bidirectional) (None, 128, 128)     98816       embedding_1[5][0]
___________________________________________________________________________________________________
permute_5 (Permute)             (None, 128, 128)     0           bidirectional_5[0][0]
___________________________________________________________________________________________________
reshape_5 (Reshape)             (None, 128, 128)     0           permute_5[0][0]
___________________________________________________________________________________________________
dense_15 (Dense)                (None, 128, 128)     16512       reshape_5[0][0]
___________________________________________________________________________________________________
attention_vec (Permute)         (None, 128, 128)     0           dense_15[0][0]
___________________________________________________________________________________________________
multiply_5 (Multiply)           (None, 128, 128)     0           bidirectional_5[0][0]
                                                                 attention_vec[0][0]
___________________________________________________________________________________________________
flatten_6 (Flatten)             (None, 16384)        0           multiply_5[0][0]
___________________________________________________________________________________________________
dense_17 (Dense)                (None, 5)            81925       flatten_6[0][0]
===================================================================================================
Total params: 20,513,029
Trainable params: 20,513,029
Non-trainable params: 0
```

14.LSTM模型训练

由于设备问题，这里训练只进行了一轮，如果能训练地比较充分，准确率还能提高。

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
model = get_lstmmodel(embed)
callbacks = [EarlyStopping(monitor='val_acc', min_delta=0.001, patience=10),
             ModelCheckpoint("LSTM.hdf5", monitor='val_acc',
                             mode='max', verbose=0, save_best_only=True)]
#设置模型提前停止,停止的条件是验证集val_acc两轮已经不增加,保存验证集val_acc最大的那个模型,名称为new_cnn.hdf5
history=model.fit(train_data,train_y, batch_size=16, epochs=5,
callbacks=callbacks,validation_data=(val_data,val_y))
```

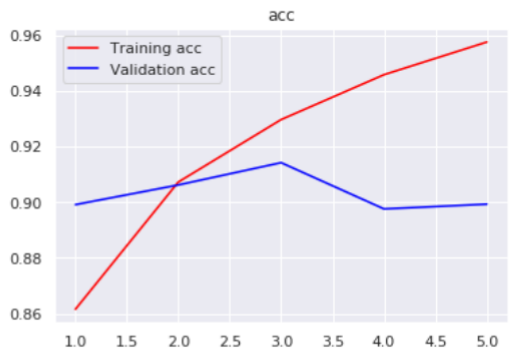
```
Non-trainable params: 0
__________________________________________________________________________________________________
Train on 108264 samples, validate on 11736 samples
Epoch 1/5
108264/108264 [==============================] - 3257s 30ms/step - loss: 0.1083 - acc: 0.9637 - val_loss: 0.3056 - val_acc: 0.9009
Epoch 2/5
  5632/108264 [>.............................] - ETA: 49:25 - loss: 0.0418 - acc: 0.9860

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-45-8405bdcd54ab> in <module>
      6                                  mode='max', verbose=0, save_best_only=True)]
      7 #设置模型提前停止,停止的条件是验证集val_acc两轮已经不增加,保存验证集val_acc最大的那个模型,名称为new_cnn.hdf5
----> 8 history=model.fit(train_data,train_y, batch_size=16, epochs=5, callbacks=callbacks,validation_data=(val_data,val_y))
      9 #每次训练批次大小为64,遍历整整10次训练集才完成

/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, **kwargs)
   1037                                         initial_epoch=initial_epoch,
   1038                                         steps_per_epoch=steps_per_epoch,
-> 1039                                         validation_steps=validation_steps)
```

15.加载最优模型，并测试验证集

```
from tensorflow.keras.models import load_model
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
lstm_attention= load_model("LSTM.hdf5")
testpre=lstm_attention.predict([test_data])

# pre=[]
tpre=np.argmax(testpre,axis=1)
testy=np.argmax(test_y,axis=1)
```

```
from sklearn.metrics import classification_report
print (classification_report(testy,tpre,digits=4))
import matplotlib.pyplot as plt
val_loss = history.history['val_loss']
loss = history.history['loss']
epochs = range(1, len(loss ) + 1)
plt.title('Loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')
plt.legend()
plt.show()

plt.cla()
val_loss = history.history['val_acc']
loss = history.history['acc']
epochs = range(1, len(loss ) + 1)

plt.title('acc')
plt.plot(epochs, loss, 'red', label='Training acc')
plt.plot(epochs, val_loss, 'blue', label='Validation acc')
plt.legend()
plt.show()
```

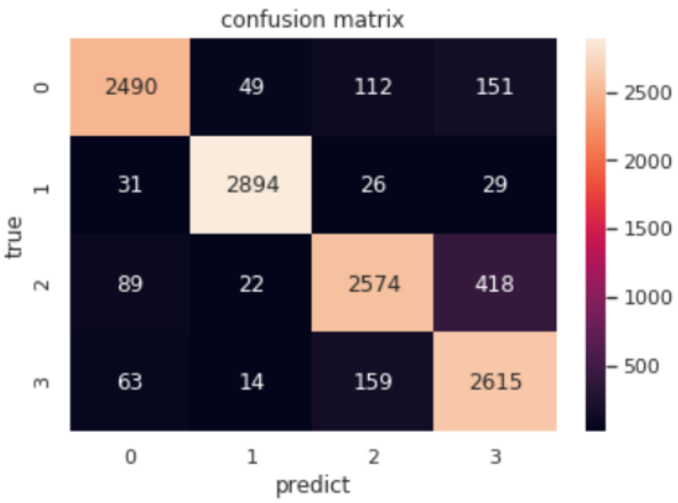
	precision	recall	f1-score	support
1	0.9315	0.8887	0.9096	2802
2	0.9715	0.9711	0.9713	2980
3	0.8966	0.8295	0.8617	3103
4	0.8139	0.9172	0.8625	2851
micro avg	0.9009	0.9009	0.9009	11736
macro avg	0.9034	0.9016	0.9013	11736
weighted avg	0.9038	0.9009	0.9012	11736



16.打印混淆矩阵和各个类别的准确率

```
def 给测试集输出指标(val_data):
    testpre=lstm_attention.predict([test_data])
    ypre=np.argmax(testpre,axis=1)
    ytrue=np.argmax(test_y,axis=1)
    def leibie_acc(test_pre,y_test):
        test_pre=np.array(test_pre)
        y_test=np.array(y_test)

        print ("准确率:",sum((test_pre==y_test).astype(int))/len(y_test))
        for i in set(list(y_test)):

            tmp=
(test_pre[np.where(y_test==i)]==y_test[np.where(y_test==i)]).astype(int)
            acc=sum(tmp)/len(tmp)
            print (i,"类别准确率为:",acc,"数量为: ",len(tmp))
        return


    import seaborn as sns
    from sklearn.metrics import confusion_matrix
    import matplotlib.pyplot as plt

    def 混洗矩阵(valp,valy):
        sns.set()
        np.set_printoptions(suppress=True)
        f,ax=plt.subplots()
        valp=np.array(valp)
        valy=np.array(valy)
        C2= confusion_matrix(valy , valp,
labels=list(range(1,len(set(df["category"]))+1)))
        print(C2) #打印出来看看
        sns.heatmap(C2,annot=True,ax=ax,fmt='.20g') #画热力图

        ax.set_title('confusion matrix') #标题
        ax.set_xlabel('predict') #x轴
        ax.set_ylabel('true') #y轴

    def 评价指标(val_data,val_y):
    #     valpre=model_load.predict(val_data)
        leibie_acc(val_data,val_y)
        混洗矩阵(val_data,val_y)

    评价指标(ypre,ytrue)
给测试集输出指标(test_data)
```

```
准确率: 0.9009032038173143
1 类别准确率为: 0.8886509635974305 数量为:  2802
2 类别准确率为: 0.9711409395973154 数量为:  2980
3 类别准确率为: 0.8295198195294876 数量为:  3103
4 类别准确率为: 0.9172220273588214 数量为:  2851
[[2490   49  112  151]
 [  31 2894   26   29]
 [  89   22 2574  418]
 [  63   14  159 2615]]
```



17.分别用cnn模型和rnn模型预测test并生成csv文件

```
# import jieba
cnn_model= load_model("textcnn.hdf5")
rnn_model= load_model("LSTM.hdf5")
def getleibie(text):
#     d=dict(zip(d_category_to_number.values(),d_category_to_number.keys()))
#     print ([i for i in list(jieba.cut(text)) if i!=" "])
    x=tokenizer.texts_to_sequences([[i.lower() for i in text.split(" ")]])
    pre_X = pad_sequences(x, maxlen=MAX_SEQUENCE_LENGTH) #将每条文本按照最大长度补0
    return np.argmax(cnn_model.predict(pre_X),axis=-1)[0]
    # LSTM模型预测
    # return np.argmax(rnn_model.predict(pre_X),axis=-1)[0]
#getleibie("i love you too")

df_test=pd.read_csv("test.csv")

from tqdm import tqdm
tmppre=[]
for i,row in tqdm(df_test.iterrows()):
    tmppre.append(getleibie(row["Title"]+" "+row["Description"]))

df_test["Class Index"]=tmppre
df_test.to_csv("submit.csv",header=True,index=False,encoding="utf-8-sig")
```

五、验证集实验结果

TextCNN：

```
准确率: 0.9141956373551465
1 类别准确率为: 0.8965024982155603 数量为:  2802
2 类别准确率为: 0.9865771812080537 数量为:  2980
3 类别准确率为: 0.8701256848211408 数量为:  3103
4 类别准确率为: 0.9038933707471063 数量为:  2851
[[2512   68  130   92]
 [  22 2940   10    8]
 [  83   30 2700  290]
 [  88   27  159 2577]]
```

BILSTM+Attention：

```
准确率: 0.9009032038173143
1 类别准确率为: 0.8886509635974305 数量为:  2802
2 类别准确率为: 0.9711409395973154 数量为:  2980
3 类别准确率为: 0.8295198195294876 数量为:  3103
4 类别准确率为: 0.9172220273588214 数量为:  2851
[[2490   49  112  151]
 [  31 2894   26   29]
 [  89   22 2574  418]
 [  63   14  159 2615]]
```

从结果中得到，本文使用的CNN模型准确率可以达到约91.42，RNN模型准确率可以达到90.09。按道理来说，本文用的BILSTM+Attention机制的准确率应该比TextCNN高才对，但是由于设备原因，BILSTM+Attention模型训练的不够充分，只进行了一轮训练，因此准确率才比较低，若能再训练充分一些，该模型准确率应该会比TextCNN高。

参考文献

[1] 李健龙,王盼卿,韩琪羽.基于双向 LSTM 的军事命名实体识别[J].计算机工程与科学,2019 (4): 20.

[2] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in neural information processing systems. 2017: 5998-6008.

[3] Guo B, Zhang C, Liu J, et al. Improving text classification with weighted word embeddings via a multi-channel TextCNN model[J]. Neurocomputing, 2019, 363: 366-374.