读取数据

In [1]:

```
import json

text=[]

filepaths=['./data/classify_json/体育.json',
           './data/classify_json/从政.json',
           './data/classify_json/国际.json',
           './data/classify_json/经济.json',

]
raw_text=[]
labels=[]

for i in range(len(filepaths)):

    filename=filepaths[i]
    label=i

    f_read=open(filename, 'r',encoding='utf8',errors='ignore')
    for line in f_read:
        line=line.replace('\\u0009','').replace('\\n','')
        obj=json.loads(line)
        sent=obj['contentClean']
        raw_text.append(sent)
        labels.append(label)
print(len(raw_text))
```

```
2000
```

数据预处理

将文本用数字化的序列进行表示

In [8]:

```
import jieba
from tensorflow.keras.preprocessing.text import Tokenizer
text=[]

for sent in raw_text:
    tokens=list(jieba.cut(sent))
    text.append(tokens)
tokenizer = Tokenizer() # 创建一个Tokenizer对象，将一个词转换为正整数
tokenizer.fit_on_texts(text)  #将词编号，词频越大，编号越小
vocab = tokenizer.word_index
#print(vocab[:100],len(vocab))
x_word_ids = tokenizer.texts_to_sequences(text) #将句子中的每个词转换为数字
#print(x_word_ids[1])
from tensorflow.keras.preprocessing.sequence import pad_sequences
x_padded_seqs = pad_sequences(x_word_ids, maxlen=50,truncating='post',padding='pre')#
#print(x_padded_seqs[2])
```

也可以采用word2vec进行词的向量化表示

w2v_model=Word2Vec.load('sentiment_analysis/w2v_model.pkl')

预训练的词向量中没有出现的词用0向量表示

embedding_matrix = np.zeros((len(vocab) + 1, 300)) for word, i in vocab.items(): try: embedding_vector = w2v_model[str(word)] embedding_matrix[i] = embedding_vector except KeyError: continue

#

TextCNN模型的构建与训练

划分数据集

In [3]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_padded_seqs,labels,shuffle=True,test
```

TextCNN模型的训练

In [4]:

```
import tensorflow as tf
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dropout, Dense, I
from tensorflow.keras.models import Model
from tensorflow.keras.layers import concatenate

main_input = Input(shape=(50,), dtype='float64')
# 嵌入层（使用预训练的词向量）
embedder = Embedding(len(vocab) + 1, 300, input_length=50, trainable=False)
#@input_dim     : 词汇表的大小，即最大整数索引+1。
#@output_dim: 稠密嵌入的维数

embed = embedder(main_input)
# 卷积层和池化层，设置卷积核大小分别为3,4,5
cnn1 = Conv1D(256, 3, padding='same', strides=1, activation='relu')(embed)
cnn1 = MaxPooling1D(pool_size=48)(cnn1)
cnn2 = Conv1D(256, 4, padding='same', strides=1, activation='relu')(embed)
cnn2 = MaxPooling1D(pool_size=47)(cnn2)
cnn3 = Conv1D(256, 5, padding='same', strides=1, activation='relu')(embed)
cnn3 = MaxPooling1D(pool_size=46)(cnn3)
# 合并三个模型的输出向量
cnn = concatenate([cnn1, cnn2, cnn3], axis=-1)
flat = Flatten()(cnn)
drop = Dropout(0.2)(flat) #在池化层到全连接层之前可以加上dropout防止过拟合
main_output = Dense(4, activation='softmax')(drop)
model = Model(inputs=main_input, outputs=main_output)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']
one_hot_labels = tf.keras.utils.to_categorical(y_train, num_classes=4)  # 将标签转换为
model.fit(x_train, one_hot_labels, batch_size=8, epochs=10)
```

```
Epoch 1/10
225/225 [==============================] - 2s 10ms/step - loss: 1.0906 - accuracy: 0.5
517
Epoch 2/10
225/225 [==============================] - 2s 10ms/step - loss: 0.6379 - accuracy: 0.7
972
Epoch 3/10
225/225 [==============================] - 2s 10ms/step - loss: 0.3536 - accuracy: 0.9
122
Epoch 4/10
```

```
225/225 [==============================] - 2s 10ms/step - loss: 0.1836 - accuracy: 0.9
783
Epoch 5/10
225/225 [==============================] - 2s 10ms/step - loss: 0.1024 - accuracy: 0.9
861
Epoch 6/10
225/225 [==============================] - 2s 10ms/step - loss: 0.0747 - accuracy: 0.9
861
Epoch 7/10
225/225 [==============================] - 2s 11ms/step - loss: 0.0627 - accuracy: 0.9
861
Epoch 8/10
225/225 [==============================] - 2s 10ms/step - loss: 0.0559 - accuracy: 0.9
856
Epoch 9/10
225/225 [==============================] - 2s 10ms/step - loss: 0.0549 - accuracy: 0.9
856
Epoch 10/10
225/225 [==============================] - 2s 10ms/step - loss: 0.0488 - accuracy: 0.9
872
```

Out[4]: <tensorflow.python.keras.callbacks.History at 0x1ce0611a860>

TextCNN模型的测试

In [5]:

```
import numpy as np
from sklearn import metrics

result = model.predict(x_test)  # 预测样本属于每个类别的概率

result_labels = np.argmax(result, axis=-1)  # 获得最大概率对应的标签

print(result_labels)
print('准确率', metrics.accuracy_score(y_test, result_labels))
```

```
[3 3 2 3 2 2 2 0 1 0 0 0 2 3 1 3 1 2 0 2 1 3 3 2 1 0 3 2 0 1 3 1 2 1 2 1 2
 3 2 3 3 2 2 0 0 3 0 1 3 0 3 3 1 2 2 3 0 1 2 0 0 2 2 2 3 3 3 2 3 0 3 3 0 2
 3 0 1 1 0 1 0 3 2 0 0 1 0 0 1 3 1 2 3 2 0 3 0 0 3 0 0 3 1 0 3 0 3 0 2 3 3
 0 3 3 1 0 1 1 1 1 0 1 2 2 1 2 3 0 1 3 1 0 0 1 0 1 2 3 1 1 0 3 2 0 2 0 2 3
 3 0 3 0 2 2 1 1 1 3 3 0 2 1 1 1 1 0 0 1 0 2 0 1 3 3 0 0 3 3 1 3 1 0 1 2 2
 3 3 0 2 3 2 3 0 1 0 0 2 3 2 3]
准确率 0.805
```

对新的文本进行类别预测

In [6]:

```
sentence='今年11月29日是这位中国女排当家球星的22岁生日。尽管在国外，但朱婷还是感受到了

sent_tokens=list(jieba.cut(sentence))
print(sent_tokens)

sent_word_ids = tokenizer.texts_to_sequences([sent_tokens]) #将句子中的每个词转换为数

print('数字化序列后：',sent_word_ids)
sent_padded_seq = pad_sequences(sent_word_ids, maxlen=50,truncating='post',padding='p
print('padding后',sent_padded_seq)
sent_result = model.predict(sent_padded_seq)  # 预测样本属于每个类别的概率

result_label = np.argmax(sent_result, axis=1)  # 获得最大概率对应的标签
print(result_label)
```

```
['今年', '11', '月', '29', '日', '是', '这位', '中国女排', '当家', '球星', '的', '22',
'岁', '生日', '。', '尽管', '在', '国外', '，', '，', '但', '朱婷', '还是', '感受', '到',
'了', '家乡', '的', '温暖', '，', '，', '因为', '29', '日', '她', '有', '一场', '特别',
```

'的', '生日会', ', ', '腾讯', '体育', '也', '对', '这场', '生日会', '进行', '了', '全程', '直播', '。', '远', '在', '土耳其', '打球', '的', '朱婷', '迎来', '自己', '的', '大', '日子', ', ']
数字化序列后： [[123, 131, 23, 991, 38, 10, 1126, 2268, 8146, 1640, 3, 717, 201, 4016, 4, 651, 6, 1960, 1, 39, 2231, 154, 1627, 44, 11, 3652, 3, 5140, 1, 127, 991, 38, 146, 13, 692, 298, 3, 11208, 1, 1656, 351, 14, 20, 1686, 11208, 77, 11, 4017, 3435, 4, 2267, 6, 525, 4337, 3, 2231, 1562, 43, 3, 91, 2907, 1]]
padding后 [[123   131   23   991   38   10 1126 2268 8146 1640   3   717
    201 4016   4   651   6 1960   1   39 2231   154 1627   44
     11 3652   3 5140   1   127   991   38   146   13   692   298
      3 11208   1 1656   351   14   20 1686 11208   77   11 4017
   3435   4]]
[0]