华东师范大学数据科学与工程学院实验报告

课程名称：当代人工智能　　**年级：**20 级　　**上机实践成绩：**

指导教师：李翔　　**姓名：**李思琪　　**学号：**10205501412

上机实践名称：多模态情感分析

上机实践编号：05　　**上机实践日期：**2023.6.9

实验 github 链接：https://github.com/lisiqi00/AI-lab5

一、实验目的

1. 设计一个多模态融合模型，根据数据集中的图片+文字描述，进行情感分类，分成 positive, neutral, negative 三类。

2. 划分数据集，在训练集上训练模型，在验证集上评估模型效果，在测试集上对位置情感的图+文，打标签。

3. 进行消融实验，即分别只输入文本或图像数据，在验证集上分析多模态融合模型的表现。

二、实验环境

语言：Python 3.8.8 keras 框架

编译器：vscode

GPU：AutoDL 租用

三、实验过程

1. 获取数据：根据 guid，将 guid、图片、对应的文字与 label 绑定在一起。

对于数据集中的所有文本和图片，读取后分别用词典表示，其中每个词典的 key 为 guid，value 为对应的文本数据或图片数据。

```
from PIL import Image
import numpy as np
import os

image_data={} #所有图片数据
text_data={} #所有文本数据

for i in range(1,5130):
    img_path='data/'+str(i)+'.jpg'
    text_path='data/'+str(i)+'.txt'
    if os.path.exists(img_path) and os.path.exists(text_path):
        #读图片数据
        img=Image.open(img_path).resize((224,224))
        img_array=np.array(img)
        image_data[i]=img_array
        #读文本数据
        with open(text_path,'r',encoding='utf-8',errors='replace') as file:
            text_str=file.read()
            text_data[i]=text_str
```

对于训练集和验证集，即 label 已知的 guid，获取其 label，以 guid 为 key,情感分类标签为 value 存入词典中。

```
label_data=dict()
with open('train.txt','r',encoding='utf-8') as label_file:
    next(label_file) #跳过第一行（标题行）
```

```
    for line in label_file:
        key, value = line.strip().split(',')
        label_data[int(key)]=value

#将 label_data 按照 guid 排序
sorted_label_data = dict(sorted(label_data.items(), key=lambda x: x[0]))
```

对于 label 未知的测试集 guid,将其 guid 保存到一个 list 中。

```
test_guid=list()
with open('test_without_label.txt','r',encoding='utf-8') as test_id_file:
    next(test_id_file)
    for line in test_id_file:
        key,value=line.strip().split(',')
        test_guid.append(int(key))
```

2. 数据预处理：对文本数据和图片数据的预处理。

(1) 对图片数据的预处理，即归一化处理；

```
train_image=np.array(train_image)/255.0
val_image=np.array(val_image)/255.0
test_image=np.array(test_image)/255.0
```

(2)对文本数据的预处理；

①大小写统一、去除停止词和多余符号；

```
#文本数据的预处理
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
sorted_pure_text_data=dict() #存储每个句子分词后的结果
#文本数据的预处理
for key in sorted_text_data:
    # print(sorted_text_data[key])
    #正则表达式匹配非字母、数字和空格的字符并删除
    pattern = r'[^a-zA-Z0-9\s]'
    cleaned_text = re.sub(pattern, '', sorted_text_data[key])
    #去除停止词，并都转化为小写字母
    stop_words=set(stopwords.words('english'))
    tokens=cleaned_text.split() #分词
    filtered_tokens = [word.lower() for word in tokens if word.lower() not in stop_words]  # 去除停止词
    # tokens=sorted_text_data[key].split() #分词
    # filtered_tokens = [word.lower() for word in tokens]  # 去除停止词
    sorted_pure_text_data[key]=filtered_tokens
    # print(sorted_pure_text_data[key])
```

②word2vec 进行词的向量化表示并构造 word2id 和 id2word 的词典；

```
#word2vec 对文本数据进行词向量化
from gensim.models import Word2Vec
w2v_model=Word2Vec(list(sorted_pure_text_data.values()),min_count=5,vector_size=50)
word_vec_matrix=w2v_model.wv #词向量矩阵
#向词向量矩阵中添加<pad>和<unk>对应的词向量
#每个词对应的词向量的维度
embedding_dim=w2v_model.vector_size
unk_vector=np.random.uniform(-1,1,embedding_dim)
pad_vector=np.zeros(embedding_dim)
#添加新的词向量
word_vec_matrix.add_vectors(keys=['<unk>','<pad>'], weights=[unk_vector, pad_vector])
#生成的词向量矩阵的词典中含有的词的 List
w2v_word_vocab=word_vec_matrix.index_to_key、
#word2vec 对文本数据进行词向量化
from gensim.models import Word2Vec
w2v_model=Word2Vec(list(sorted_pure_text_data.values()),min_count=5,vector_size=50)
word_vec_matrix=w2v_model.wv #词向量矩阵
#向词向量矩阵中添加<pad>和<unk>对应的词向量
#每个词对应的词向量的维度
embedding_dim=w2v_model.vector_size
unk_vector=np.random.uniform(-1,1,embedding_dim)
pad_vector=np.zeros(embedding_dim)
```

```
#添加新的词向量
word_vec_matrix.add_vectors(keys=['<unk>','<pad>'], weights=[unk_vector, pad_vector])
#生成的词向量矩阵的词典中含有的词的 List
w2v_word_vocab=word_vec_matrix.index_to_key
```

③将出现频率低的词用<unk>代替；

```
train_text2id=list()
for sentence in train_text:
    sentence_id_list=list()
    for word in sentence:
        if word not in w2v_word_vocab:
            word='<unk>'
        text2id=word2index[word]
        sentence_id_list.append(text2id)
    train_text2id.append(sentence_id_list)
```

④将句子补全至定长；

```
#获取句子的最大长度
max_length=-1
for i in range(len(list(sorted_pure_text_data.values()))):
    if len(list(sorted_pure_text_data.values())[i])>=max_length:
        max_length=len(list(sorted_pure_text_data.values())[i])
print(max_length)

#将句子的长度都补到最长
from tensorflow.keras.preprocessing.sequence import pad_sequences
train_text_pad=pad_sequences(train_text2id,maxlen=max_length,padding='post',value=word2index['<pad>'])
val_text_pad=pad_sequences(val_text2id,maxlen=max_length,padding='post',value=word2index['<pad>'])
test_text_pad=pad_sequences(test_text2id,maxlen=max_length,padding='post',value=word2index['<pad>'])
```

⑤对标签 label 的处理：将'positive','negative','neutral'文本形成 label2idx 词典和 idx2label 词典，并进行独热编码；

```
#处理标签 label
from keras.utils import to_categorical
# 将情感标签转换为数字标签
labels=['positive','negative','neutral']
label_to_index = {'positive': 0, 'negative': 1, 'neutral': 2}
index_to_label = {0: 'positive', 1: 'negative', 2: 'neutral'}

train_label_idx = np.array([label_to_index[label] for label in train_label])
val_label_target_idx = np.array([label_to_index[label] for label in val_label_target])
#将标签进行独热编码
train_label_one_hot = to_categorical(train_label_idx)
val_label_one_hot = to_categorical(val_label_target_idx)
```

3. 文本数据的框架的搭建：

①LSTM（text_features 即为提取出来的文本特征）

```
text_input=Input(shape=(max_length,))
text_embed=Embedding(input_dim=len(w2v_word_vocab),output_dim=embedding_dim,weights=[word_vec_matrix.vectors],
trainable=False)(text_input)
text_features=LSTM(units=128)(text_embed)
```

②bi-LSTM（text_features_bilstm 即为提取出来的文本特征）

```
text_input_bilstm=Input(shape=(max_length,))
text_embed_bilstm=Embedding(input_dim=len(w2v_word_vocab),output_dim=embedding_dim,weights=[word_vec_matrix.ve
ctors],trainable=False)(text_input_bilstm)
lstm_layer=LSTM(units=128)
bidirectional_lstm = Bidirectional(lstm_layer)
text_features_bilstm = bidirectional_lstm(text_embed_bilstm)
```

4. 图片数据的框架的搭建：

①VGGNet19

```
base_model = VGG19(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
```

```
image_input = Input(shape=(224, 224,3))
image_features = base_model(image_input)
image_features = Flatten()(image_features)
```

②ResNet50

```
image_input_resnet = Input(shape=(224, 224,3))
resnet_model  = ResNet50(include_top=False, weights='imagenet', input_tensor=image_input_resnet)
image_features_resnet = resnet_model.output
image_features_resnet = Flatten()(image_features_resnet)
```

5. 两个模型进行融合，并计算在验证集上的表现：

在这里，我对两个模态进行融合的方法是晚期融合、联合训练，即将它们视为两个不同的分支，单独处理，在得到特征之后，对提取的特征进行融合。这里采用 Concatenate() 层将两种特征通过直接连接的方法进行融合。

①LSTM+VGGNet19

```
#融合文本和图像特征
combined_features = Concatenate()([text_features, image_features])
predictions = Dense(3, activation='softmax')(combined_features)
#构建多模态融合模型
model = Model(inputs=[text_input, image_input], outputs=predictions)
#编译模型
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

model.fit(x=[train_text_pad,train_image],y=train_label_one_hot,epochs=10,batch_size=32,validation_data=([val_t
ext_pad,val_image],val_label_one_hot))

# 使用模型进行预测
lstm_vgg_predictions = model.predict([test_text_pad, test_image])

# 将预测结果转换为标签
predicted_labels_1 = np.argmax(lstm_vgg_predictions, axis=1)
predicted_labels_1 = [index_to_label[label] for label in predicted_labels_1]

# 打印预测结果
for i, label in enumerate(predicted_labels_1):
    print(f"Sample {i+1}: Predicted Label = {label}")
```

10 轮迭代运算后的正确率达到 62%左右。

```
Epoch 1/10

2023-07-14 12:51:55.979604: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8101
2023-07-14 12:51:59.111221: I tensorflow/stream_executor/cuda/cuda_blas.cc:1786] TensorFloat-32 will be used for the matrix multiplication. This will only b

100/100 [==============================] - 35s 264ms/step - loss: 2.6652 - accuracy: 0.5747 - val_loss: 0.8504 - val_accuracy: 0.6250
Epoch 2/10
100/100 [==============================] - 26s 256ms/step - loss: 0.9137 - accuracy: 0.5872 - val_loss: 0.8634 - val_accuracy: 0.6250
Epoch 3/10
100/100 [==============================] - 26s 256ms/step - loss: 0.9164 - accuracy: 0.5881 - val_loss: 0.8859 - val_accuracy: 0.6237
Epoch 4/10
100/100 [==============================] - 26s 256ms/step - loss: 0.9165 - accuracy: 0.5888 - val_loss: 0.8473 - val_accuracy: 0.6250
Epoch 5/10
100/100 [==============================] - 26s 255ms/step - loss: 0.9109 - accuracy: 0.5900 - val_loss: 0.9104 - val_accuracy: 0.6100
Epoch 6/10
100/100 [==============================] - 25s 255ms/step - loss: 0.9122 - accuracy: 0.5897 - val_loss: 0.8590 - val_accuracy: 0.6250
Epoch 7/10
100/100 [==============================] - 26s 256ms/step - loss: 0.9068 - accuracy: 0.5900 - val_loss: 0.8506 - val_accuracy: 0.6250
Epoch 8/10
100/100 [==============================] - 25s 251ms/step - loss: 0.9025 - accuracy: 0.5900 - val_loss: 0.8597 - val_accuracy: 0.6250
Epoch 9/10
100/100 [==============================] - 25s 255ms/step - loss: 0.9032 - accuracy: 0.5900 - val_loss: 0.8581 - val_accuracy: 0.6250
Epoch 10/10
100/100 [==============================] - 25s 254ms/step - loss: 0.9041 - accuracy: 0.5900 - val_loss: 0.8503 - val_accuracy: 0.6250

<keras.callbacks.History at 0x7f0fec0812b0>
```

②bi-LSTM+ResNet

```
#构建多模态模型(lstm+resnet)
#融合文本和图像特征
combined_features_2 = Concatenate()([text_features_bilstm, image_features_resnet])
combined_features = Dropout(rate=0.1)(combined_features_2)
```

```
predictions_2 = Dense(3, activation='softmax')(combined_features_2)
#构建多模态融合模型
lstm_resnet_model = Model(inputs=[text_input_bilstm, image_input_resnet], outputs=predictions_2)
#编译模型
learning_rate = 0.0025  # 设置所需的学习率
optimizer = Adam(learning_rate=learning_rate)
lstm_resnet_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
# lstm_resnet_model.summary()
lstm_resnet_model.fit(x=[train_text_pad,train_image],y=train_label_one_hot,epochs=10,batch_size=32,validation_
data=([val_text_pad,val_image],val_label_one_hot))

# 使用模型进行预测
lstm_resnet_predictions = lstm_resnet_model.predict([test_text_pad, test_image])

# 将预测结果转换为标签
predicted_labels = np.argmax(lstm_resnet_predictions, axis=1)
predicted_labels = [index_to_label[label] for label in predicted_labels]

# 打印预测结果
for i, label in enumerate(predicted_labels):
    print(f"Sample {i+1}: Predicted Label = {label}")
```

10 轮迭代计算最后正确率能够达到 60%左右。

```
Epoch 1/10
100/100 [==============================] - 25s 192ms/step - loss: 3.6875 - accuracy: 0.5619 - val_loss: 146.5488 - val_accuracy: 0.4963
Epoch 2/10
100/100 [==============================] - 17s 168ms/step - loss: 2.2020 - accuracy: 0.5691 - val_loss: 1691.7037 - val_accuracy: 0.6025
Epoch 3/10
100/100 [==============================] - 17s 167ms/step - loss: 2.0340 - accuracy: 0.5753 - val_loss: 0.8616 - val_accuracy: 0.6250
Epoch 4/10
100/100 [==============================] - 17s 167ms/step - loss: 1.5393 - accuracy: 0.5753 - val_loss: 0.8529 - val_accuracy: 0.6162
Epoch 5/10
100/100 [==============================] - 17s 168ms/step - loss: 1.3882 - accuracy: 0.5678 - val_loss: 0.8775 - val_accuracy: 0.6237
Epoch 6/10
100/100 [==============================] - 17s 168ms/step - loss: 1.3140 - accuracy: 0.5750 - val_loss: 0.8651 - val_accuracy: 0.6225
Epoch 7/10
100/100 [==============================] - 17s 169ms/step - loss: 1.2697 - accuracy: 0.5688 - val_loss: 0.8539 - val_accuracy: 0.6212
Epoch 8/10
100/100 [==============================] - 17s 170ms/step - loss: 1.3603 - accuracy: 0.5734 - val_loss: 229.9778 - val_accuracy: 0.6200
Epoch 9/10
100/100 [==============================] - 17s 169ms/step - loss: 1.2976 - accuracy: 0.5806 - val_loss: 87.2703 - val_accuracy: 0.5525
Epoch 10/10
100/100 [==============================] - 17s 172ms/step - loss: 1.2789 - accuracy: 0.5769 - val_loss: 12.0128 - val_accuracy: 0.6012

<keras.callbacks.History at 0x7f0db8466b80>
```

6. 在测试集上做预测

Model.predict()得到分别属于三种类别的概率，取最大的一项为最终的预测结果。

```
# 使用模型进行预测
lstm_resnet_predictions = lstm_resnet_model.predict([test_text_pad, test_image])

# 将预测结果转换为标签
predicted_labels = np.argmax(lstm_resnet_predictions, axis=1)
predicted_labels = [index_to_label[label] for label in predicted_labels]

# 打印预测结果
for i, label in enumerate(predicted_labels):
    print(f"Sample {i+1}: Predicted Label = {label}")
```

将预测结果写入文件中。其中 test_lstm_vgg19 表示用 lstm 和 vgg19 建模得到的标签；test_bilstm_resnet 表示用 lstm 和 resnet 建模得到的标签。

7. 消融实验：对于文本模型和图片模型分别测试其在验证集上的表现。

(1)对于第一个多模态融合模型：lstm+vggnet_19:

①lstm:

```
predictions_text = Dense(3, activation='softmax')(text_features)
# 构建仅文本输入的模型
text_model = Model(inputs=text_input, outputs=predictions_text)
text_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 仅使用文本数据进行训练
text_model.fit(train_text_pad, train_label_one_hot, epochs=10, batch_size=32, validation_data=(val_text_pad,
val_label_one_hot))

# 在验证集上评估模型
text_model.evaluate(val_text_pad, val_label_one_hot)
```

在验证集上测试的结果为[0.8920366168022156, 0.6012499928474426]，分别表示 val_loss 和 val_acc。

②VggNet_19:

```
predictions_images = Dense(3, activation='softmax')(image_features)
# 构建仅图像输入的模型
image_model = Model(inputs=image_input, outputs=predictions_images)
image_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 仅使用图像数据进行训练
image_model.fit(train_image, train_label_one_hot, epochs=10, batch_size=32, validation_data=(val_image,
val_label_one_hot))

# 在验证集上评估模型
image_model.evaluate(val_image, val_label_one_hot)
```

在验证集上测试的结果为[0.8997756242752075, 0.6050000190734863]，分别表示 val_loss 和 val_acc。

(2)对于第二个多模态融合模型：bi-lstm+resnet_50:

①bi-lstm:

```
predictions_text_bilstm = Dense(3, activation='softmax')(text_features_bilstm)

# 构建文本模型
text_model = Model(inputs=text_input_bilstm, outputs=predictions_text_bilstm)
text_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
text_model.fit(train_text_pad, train_label_one_hot, epochs=10, batch_size=32, validation_data=(val_text_pad,
val_label_one_hot))

# 在验证集上评估模型
text_model.evaluate(val_text_pad, val_label_one_hot)
```

在验证集上测试的结果为[0.8765424489974976, 0.6037499904632568]，分别表示 val_loss 和 val_acc。

②ResNet_50:

```
predictions_images_resnet = Dense(3, activation='softmax')(image_features_resnet)
# 构建图像模型
image_model = Model(inputs=image_input_resnet, outputs=predictions_images_resnet)
image_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
image_model.fit(train_image, train_label_one_hot, epochs=10, batch_size=32, validation_data=(val_image,
val_label_one_hot))

# 在验证集上评估模型
image_model.evaluate(val_image, val_label_one_hot)
```

在验证集上测试的结果为[1.0930933952331543, 0.59375]，分别表示 val_loss 和 val_acc。

四、模型设计思路

我分别设计了两个多模态融合模型，分别是 LSTM+VggNet_19 融合模型，bi-LSTM+ResNet_50 模型。LSTM 和 bi-LSTM 分别用来处理文本数据，VggNet_19 和 ResNet_50 分别用来处理图像数据。LSTM 模型与普通的 CNN 相比，更擅长处理文本这种序列型的数据，它可以保存记忆状态，使其能够捕捉文本中的上下文信息。而之所以选择 VggNet_19 和 ResNet_50,是因为这是两个用于图像分类的经典的卷积神经网络。VggNet_19 具有小卷积核和多个卷积层的特点，使其能够对图像的特征捕获得更加丰富且准确，在 lab3 的实验中，我使用 vggNet_19 得到的准确率是最高的，所以先尝试了这一种。ResNet_50 则加深了训练的深度，对于图像分类也能产生很好的效果。

五、实验过程中遇到的问题以及我的解决方法

1. 在进行消融实验的时候，遇到报错如下：ValueError: Shapes (32, 3) and (32, 128) are incompatible。

这个报错是因为模型的输出形状与标签的形状不匹配导致的，在消融实验中，只是用文本数据进行训练时，模型的输出应该与文本的特征维度一致。

改正：在消融实验前添加一句，将模型的输出层修改为适应文本特征的维度即可。

```
predictions_text = Dense(num_classes, activation='softmax')(text_features)
```

2. 注意图像分类问题，要将标签映射到数字表示后，再映射到 one-hot 向量表示，否则会出现报错。

3. 在本次和上次实验中，经常出现 GPU 无法再使用的问题，报错如下：

```
UnknownError: Graph execution error: Fail to find the dnn implementation.
[[{{node CudnnRNN}}]] [[model_train/train_encoder_lstm/PartitionedCall]]
[Op:__inference_train_function_9821]
```

因为是租用的服务器，所以我没办法修改它的配置，最后解决的方法只能是关机创建镜像，然后利用镜像重新开一个 gpu。

4. 在实验过程中出现了，训练集上表现效果好，但是验证集上表现效果较差的问题，说明此时出现了过拟合的问题，在这里我的解决方法是：调整学习率，引入 dropout 层做正则化处理。

六、总结

1. 实验 github 链接：https://github.com/lisiqi00/AI-lab5

2. 通过本次实验设计了多模态融合模型结构，对于神经网络、机器学习的应用在文本和图片上的交叉混合运用有了更好的掌握，动手实现了两个多模态融合结构，实现了情感分类的功能，效果能够达到 60%以上的正确率，因为题给的数据集中三个种类的情感分布得不均匀，绝大多数都是 Positive 类型，所以从数据集中随机选择数据进行情感分类时，结果就会绝大多数倾向于 positive，因为 neutral 和 negative 的情感对应的特征不如 positive 的明确。

3. 本学期实验圆满结束，后续还要多加练习。辛苦老师和助教一学期的辛苦教学与帮助~