homework4

数据集

数据集共包含34282个节点，每个节点为一段文本。除节点属性信息外，还提供了网络的结构信息(edges.csv)

- train.csv 包含id, name, text, label
- val.csv 包含id, name, text, label
- test.csv 没有label，提交预测文件pred.csv(格式详见submit_example.csv)
- edges.csv 图的结构信息，每行包含一条边连接的两个节点node1,node2
- submit_example.csv 需要提交的预测文件的示例，包含id, label(预测的类别标签)

模型搭建

对于节点数据的处理

利用hugging face 的bert模型进行基于文本的节点分类

在colab上运行

1.挂载数据集

```
from google.colab import drive

drive.mount('/content/drive', force_remount=True)
%cd ..
%cd content/drive/My Drive
```

2.数据读取和查看

```
import pandas as pd

train_data=pd.read_csv("data/train.csv")
val_data=pd.read_csv("data/val.csv")
test_data=pd.read_csv("data/test.csv")
train_data.head()
```

	id	name	text	label
0	21687	"The Perfect Furlough"	1958 films;American films;Comedy films;English...	3
1	15781	"Vicky Jenson"	American animators;Living people;Year of birth...	0
2	5205	"Billy Burke "	1966 births;American film actors;American tele...	0
3	14310	"Giovonnie Samuels"	1985 births;African American actors;American c...	0
4	21211	"To Walk with Lions"	1999 films;Canadian films;Kenyan films;English...	3

3.应用bert tokenizer并进行fine tuning

bert与原始transformers的主要区别是, BERT没有解码器, 但在基本版本中堆叠了12个编码器。

调用encode_examples函数对训练集以及验证集进行编码并划分batch

tokenizer使用预训练的bert-base-uncased

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# train dataset
ds_train_encoded =
encode_examples(train_data).shuffle(10000).batch(batch_size)
# val dataset
ds_val_encoded = encode_examples(val_data).batch(batch_size)
```

通过encode_plus将token映射到词嵌入,将文本分词后创建一个包含对应 id，token 类型及是否遮盖的词典。加入特殊token,如[CLS] token将插入序列的开头，[SEP] token位于末尾。

这里设置return_attention_mask为true,在词典中包含覆盖。

bert 输入格式 [CLS] + tokens + [SEP] + padding

```
def convert_example_to_feature(review):
    return tokenizer.encode_plus(review,
                                 add_special_tokens = True,
                                 max_length = max_length,
                                 pad_to_max_length = True,
                                 return_attention_mask = True,
                                )
```

构建tensor flow 数据集

取出每一行的text和label对应的值，利用convert_example_to_feature函数转化为bert的输入

保存通过tokenizer得到的input_ids_list,input_ids是token切片，用于构建将用作模型输入的序列。

attention mask在[0, 1] 中选择的掩码值：1 表示未屏蔽的标记，0 表示标记的标记。用0作为填充添加的标记。

```
def encode_examples(ds, limit=-1):
    # prepare list, so that we can build up final TensorFlow dataset from
slices.
    input_ids_list = []
    token_type_ids_list = []
    attention_mask_list = []
    label_list = []
    if (limit > 0):
        ds = ds.take(limit)
```

```
    for index, row in ds.iterrows():

        review = row["text"]
#         print(review)
        label = row["label"]
#         print(label)
        bert_input = convert_example_to_feature(review)

        input_ids_list.append(bert_input['input_ids'])
        token_type_ids_list.append(bert_input['token_type_ids'])
        attention_mask_list.append(bert_input['attention_mask'])
        label_list.append([label])
    return tf.data.Dataset.from_tensor_slices((input_ids_list,
attention_mask_list, token_type_ids_list,
label_list)).map(map_example_to_dict)
```

4.模型训练

学习率设为2e-5。

模型初始化，设置分类类别为4。优化器选择adam。

```
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=4)
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate,epsilon=1e-
08, clipnorm=1)
```

利用SparseCategoricalCrossentropy计算损失以及准确率

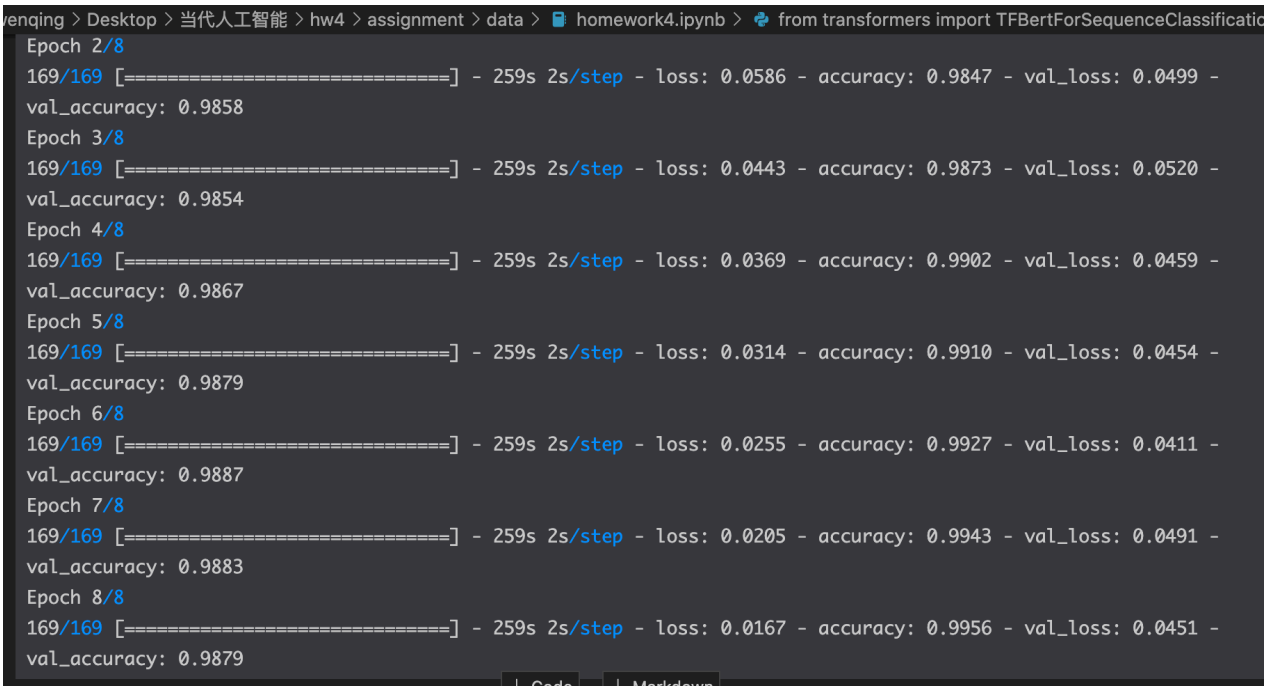
```
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

拟合模型,设置epoch为8

```
bert_history = model.fit(ds_train_encoded, epochs=number_of_epochs,
validation_data=ds_val_encoded)
```

5.训练效果

验证准确率可达98.87%



6.模型预测

复用之前的模型编码，将函数中的label一栏的传参和存储删掉。

```
def map_test_example_to_dict(input_ids, attention_masks, token_type_ids):
    return {
      "input_ids": input_ids,
      "token_type_ids": token_type_ids,
      "attention_mask": attention_masks,
  }
def encode_test_examples(ds, limit=-1):

    input_ids_list = []
    token_type_ids_list = []
    attention_mask_list = []
    label_list = []
    if (limit > 0):
        ds = ds.take(limit)

    for index, row in ds.iterrows():

        review = row["text"]
#         print(review)
#         print(label)
        bert_input = convert_example_to_feature(review)
        input_ids_list.append(bert_input['input_ids'])
        token_type_ids_list.append(bert_input['token_type_ids'])
        attention_mask_list.append(bert_input['attention_mask'])
    return tf.data.Dataset.from_tensor_slices((input_ids_list,
attention_mask_list, token_type_ids_list)).map(map_test_example_to_dict)
ds_test_encoded = encode_test_examples(test_data).batch(batch_size)
```

调用model.predict进行预测

```
out=model.predict(ds_test_encoded)
```

out.logits为最后输出的结果矩阵

通过np.argmax取每一行最大元素所在的标签作为这一组的标签值，并存储下来

```
import numpy as np
idx = np.argmax(output, axis=1)
idx.shape
df = pd.DataFrame(idx)
df.to_csv('myfile.csv')
```

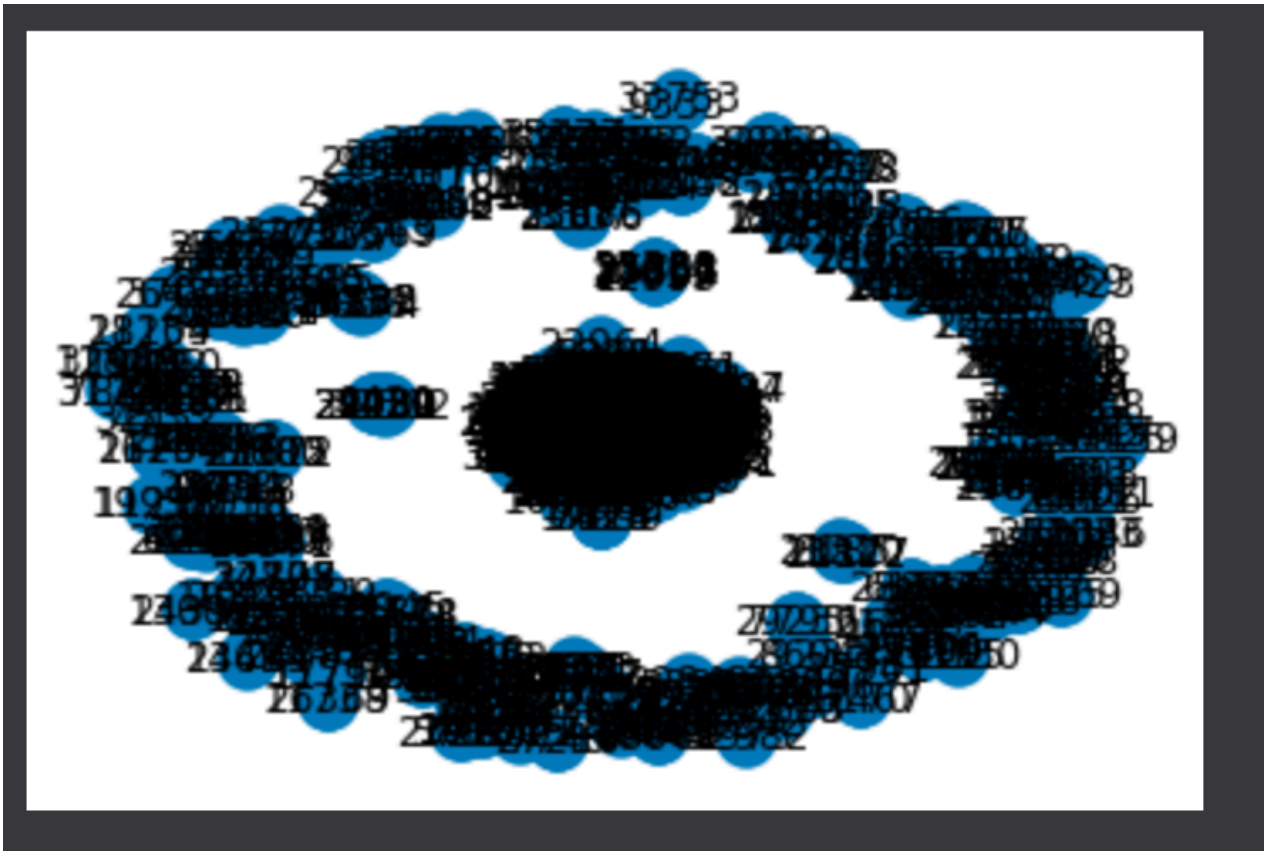
对于图结构信息的研究

尝试画图看一下整体结构,利用networkx库

```
import networkx as nx
import matplotlib.pyplot as plt

for row in edge.itertuples():
    g.add_edge(getattr(row,'node1'),getattr(row,'node2'))

fig, ax = plt.subplots()
nx.draw(g, ax=ax, with_labels=True) # show node label
plt.show()
```



可以看到网络的连通性较差。

图的基本信息

```
Node size: 27312
Edge size: 122706
==================================================================
```

模型效果

单纯使用文本分类效果已经非常好了，由于图的结构非连通，考虑到加入后反而会使得模型训练结果变差，最终使用的模型不包含图结构信息，仅基于文本分类。val_acc约为99%。

错误处理

1.本地tensorflow环境不匹配

```
[Running] python -u "/Users/wangwenqing/Desktop/当代人工智能/hw4/data/homework4.py"
RuntimeError: module compiled against API version 0xe but this version of numpy is 0xd
Traceback (most recent call last):
  File "/Users/wangwenqing/Desktop/当代人工智能/hw4/data/homework4.py", line 5, in <module>
    import tensorflow as tf
  File "/Users/wangwenqing/opt/anaconda3/lib/python3.7/site-packages/tensorflow/__init__.py", line 37, in <module>
    from tensorflow.python.tools import module_util as _module_util
  File "/Users/wangwenqing/opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/__init__.py", line 37, in <module>
    from tensorflow.python.eager import context
  File "/Users/wangwenqing/opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/eager/context.py", line 35, in <module>
    from tensorflow.python.client import pywrap_tf_session
  File "/Users/wangwenqing/opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/client/pywrap_tf_session.py", line 19, in <mod
    from tensorflow.python.client._pywrap_tf_session import *
ImportError: SystemError: <built-in method __contains__ of dict object at 0x7fd6d73660f0> returned a result with an error set

[Done] exited with code=1 in 3.751 seconds
```

决定放到colab上训练

2.colab上由于挂载线程连接故障报错

```
OSError: [Errno 107] Transport endpoint is not connected
```

从新创建一个副本来再次尝试挂载

报错即可消失

Reference

https://discuss.huggingface.co/t/getting-outputs-of-mode-predict-per-sentence-input/7037

https://huggingface.co/docs/transformers/internal/tokenization_utils

https://swatimeena989.medium.com/bert-text-classification-using-keras-903671e0207d#c685

https://www.jiqizhixin.com/articles/2019-05-16-14