《实验报告》

实验题目 设计实现Hugging Face数据集用途分类

学院系别 信息学院

专业名称 人工智能

学生姓名 郑凯航

学生学号 37220222203885

任课教师 曹冬林

2024年 5 月 9 日

一、 实验目的

列举实验要达到的几个目的。

针对实验1和实验2构建的数据集信息分析以下内容：

- 设计实现通过数据简介进行大类分类的程序。
- 设计实现通过数据简介进行大类+小类分类的程序。

二、 实验环境

列举实验机器环境，配置环境。

os: Windows

platform Windows-10-10.0.22621-SP0

version: 10.0.22621

python: 3.8.18

beautifulsoup4==4.12.3

numpy==1.23.2

pandas==1.2.4

Requests==2.31.0

sentence_transformers==2.6.1

torch==2.0.1

transformers==4.39.3

三、 实验步骤

详细说明所做实验的基本实验步骤及流程图。

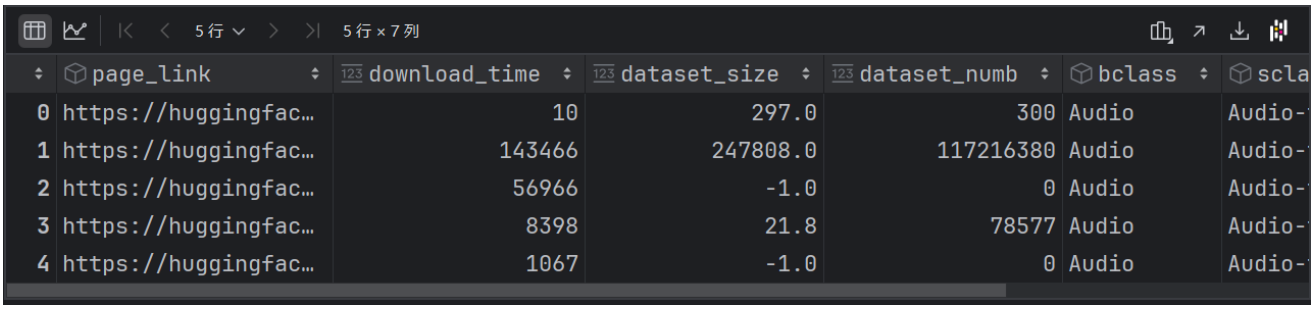
- 数据处理
- 词嵌入
- 标签编码
- 相关性评估

数据处理

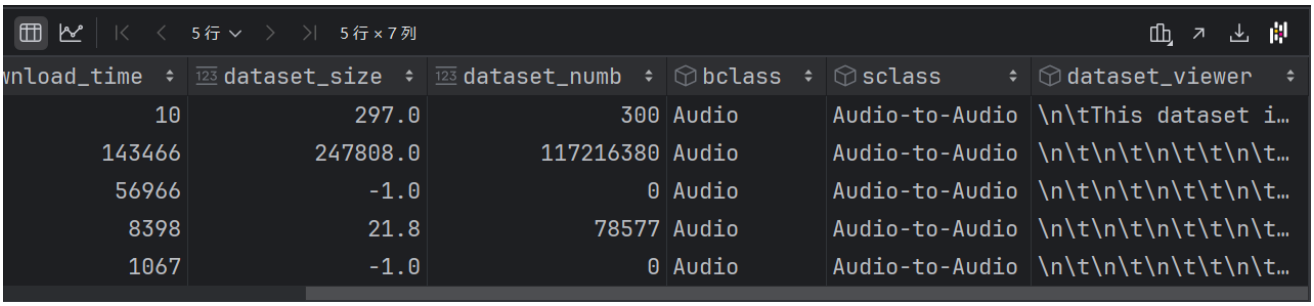
数据预处理

空缺值处理

- 对于数据简介，其中有许多无意义的换行符号，手动进行处理
- 对于url,前面的https://huggingface.co/datasets/无意义，手动删除
- 获取到的网页数据有部分缺失，使用-1替换



	page_link	download_time	dataset_size	dataset_numb	bclass	scla
0	https://huggingfac…	10	297.0	300	Audio	Audio-
1	https://huggingfac…	143466	247808.0	117216380	Audio	Audio-
2	https://huggingfac…	56966	-1.0	0	Audio	Audio-
3	https://huggingfac…	8398	21.8	78577	Audio	Audio-
4	https://huggingfac…	1067	-1.0	0	Audio	Audio-



wnload_time	dataset_size	dataset_numb	bclass	sclass	dataset_viewer
10	297.0	300	Audio	Audio-to-Audio	\n\tThis dataset i…
143466	247808.0	117216380	Audio	Audio-to-Audio	\n\t\n\t\n\t\t\n\t…
56966	-1.0	0	Audio	Audio-to-Audio	\n\t\n\t\n\t\t\n\t…
8398	21.8	78577	Audio	Audio-to-Audio	\n\t\n\t\n\t\t\n\t…
1067	-1.0	0	Audio	Audio-to-Audio	\n\t\n\t\n\t\t\n\t…

处理前

10行 × 7列

	page_link	download_time	dataset_size	dataset_numb	bclass	scla
0	437aewuh/dog-dataset	10	297.0	300	Audio	Audio-
1	allenai/c4	143466	247808.0	117216380	Audio	Audio-
2	Anthropic/hh-rlhf	56966	-1.0	0	Audio	Audio-
3	b-mc2/sql-create-c…	8398	21.8	78577	Audio	Audio-
4	berkeley-nest/Nectar	1067	-1.0	0	Audio	Audio-
5	bigcode/starcoderd…	6337	-1.0	0	Audio	Audio-
6	cais/mmlu	3573730	104.0	231400	Audio	Audio-
7	Cnam-LMSSC/vibravox	1016	-1.0	0	Audio	Audio-
8	CohereForAI/aya_da…	1534	138.0	205568	Audio	Audio-
9	crystalai/autotrai…	-1	-1.0	0	Audio	Audio-

10行 × 7列

wnload_time	dataset_size	dataset_numb	bclass	sclass	dataset_viewer
10	297.0	300	Audio	Audio-to-Audio	This dataset is a …
143466	247808.0	117216380	Audio	Audio-to-Audio	C4Dataset SummaryA…
56966	-1.0	0	Audio	Audio-to-Audio	Dataset Card for H…
8398	21.8	78577	Audio	Audio-to-Audio	OverviewThis datas…
1067	-1.0	0	Audio	Audio-to-Audio	Dataset Card for N…
6337	-1.0	0	Audio	Audio-to-Audio	StarCoder Training…
3573730	104.0	231400	Audio	Audio-to-Audio	Dataset Card for M…
1016	-1.0	0	Audio	Audio-to-Audio	Dataset Card for V…
1534	138.0	205568	Audio	Audio-to-Audio	Dataset SummaryThe…
-1	-1.0	0	Audio	Audio-to-Audio	-1

0	1	2
10.0	297.0	300.0
143466.0	247808.0	117216380.0
56966.0	-1.0	-1.0
8398.0	21.8	78577.0
1067.0	-1.0	-1.0
6337.0	-1.0	-1.0
3573730.0	104.0	231400.0
1016.0	-1.0	-1.0
1534.0	138.0	205568.0
-1.0	-1.0	-1.0

处理后

- 文本向量化

  借助网络上预处理好的模型将数据简介和数据名向量化，从而可以被处理

在hugging face上找到可以将文本转为512维的向量的模型

`jina_embeddings_v2_small_en`

编码函数

```
def generate_embeddings():
        """
        文本转向量
        """         # 返回numpy数据
        TextEmbedding_model = AutoModel.from_pretrained('jinaai/jina-
embeddings-v2-small-en', trust_remote_code=True)
        return TextEmbedding_model
embedding_model = generate_embeddings()

def m_embedding(text):
    """
    文本转向量
:param text::return: tensor
    """     embeddings = embedding_model.encode(text,max_length=3487)
    return embeddings
```

之后将文本向量化后的数据与下载信息等结合在一起最终得到的数据

```
x = np.concatenate((x2,x1), axis=1)
print(type(x),x.shape,x.dtype)
x
```

在 2024.05.09 15:02:54 于 14ms内执行

```
<class 'numpy.ndarray'> (30, 1027) float64
```

1-10 30 行 × 1,027 列

	0	1	2	3	4	5	6	7	8
0	10.0	297.000000	300.0	-0.157304	-0.332159	0.328267	0.516192	0.046530	-0.18
1	143466.0	247808.000000	117216380.0	-0.362347	-0.606690	-0.083577	0.916346	-0.402184	-0.76
2	56966.0	-1.000000	0.0	-0.111131	-0.599770	0.126193	0.431490	-0.231033	-0.35
3	8398.0	21.800000	78577.0	-0.385664	-0.361137	-0.171430	0.836958	-0.430386	-0.05
4	1067.0	-1.000000	0.0	-0.175287	-0.541404	-0.238136	0.394209	-0.029357	0.07
5	6337.0	-1.000000	0.0	-0.264410	-0.305974	0.094566	0.709095	-0.257594	-0.31
6	3573730.0	104.000000	231400.0	-0.505673	-0.582827	-0.172755	0.172171	-0.195341	-0.36
7	1016.0	-1.000000	0.0	-0.207851	-0.546666	-0.362612	0.313032	-0.311423	-0.38
8	1534.0	138.000000	205568.0	-0.187412	-0.809365	0.201911	0.625714	0.140776	-0.35
9	-1.0	-1.000000	0.0	-0.248444	-0.959300	0.419456	-0.021442	-0.237106	-0.31

标签

one_hot编码

基于大小类制作one_hot编码

发现小类一共46个，则可以用维度46的向量作为标签
通过得到的映射关系构建c2h字典，将小类名映射到标签

```
c2h = dict()
label = np.eye(46,46,dtype=np.float64)
k = 0
for _,v in cls.items():
    for c in v :
        # print(v)
        c2h[c.replace(' ','_')] = label[k]
        k += 1
y = df.loc[:,'sclass'].apply(lambda x:np.array(c2h[x]))
y = np.vstack(y)
```

相关性评估

由于发现直接编码后的数据用来训练网络效果很差，所以继续将46个小类标签也编码，随后计算他们相关度，将512*2维转为46*2维

本质是将文本信息和标签信息映射到函数空间上，然后计算两个向量的相关度

最后可以得到98维的向量，其中数据简介，数据标签与小类的相关性占46*2=92维，剩下的6维表示三位下载次数，数据大小和数据条数和三位代表他们是不是缺失值的标志位

	96	97
04, dtype=torch.float64)	tensor(0.7105, dtype=torch.float64)	tensor(0.6921, dtype=torch.float64)
74, dtype=torch.float64)	tensor(0.7579, dtype=torch.float64)	tensor(0.7216, dtype=torch.float64)
62, dtype=torch.float64)	tensor(0.7243, dtype=torch.float64)	tensor(0.6858, dtype=torch.float64)
02, dtype=torch.float64)	tensor(0.6598, dtype=torch.float64)	tensor(0.7021, dtype=torch.float64)
61, dtype=torch.float64)	tensor(0.7409, dtype=torch.float64)	tensor(0.6995, dtype=torch.float64)
63, dtype=torch.float64)	tensor(0.7434, dtype=torch.float64)	tensor(0.7376, dtype=torch.float64)
83, dtype=torch.float64)	tensor(0.7583, dtype=torch.float64)	tensor(0.7249, dtype=torch.float64)
84, dtype=torch.float64)	tensor(0.7713, dtype=torch.float64)	tensor(0.7104, dtype=torch.float64)
78, dtype=torch.float64)	tensor(0.7186, dtype=torch.float64)	tensor(0.7356, dtype=torch.float64)
60, dtype=torch.float64)	tensor(0.7633, dtype=torch.float64)	tensor(0.7304, dtype=torch.float64)

类型转换

由于在pytorch模型上进行计算的数据必须有一致类型和设备，因此在开始就将其基于平台定义

```
def try_gpu(i=0):
    if torch.cuda.device_count() >= i + 1:
        return torch.device(f'cuda:{i}')
    return torch.device('cpu')
```

```
device = try_gpu()
example = True
typ = torch.float32
```

在数据处理完成后将其进行转换，避免不必要的麻烦

```
x,y = torch.load('data.data')
x = x.to(typ).to(device)
y = y.to(typ).to(device)
```

网络搭建

参考网上资料，使用pytorch框架

```
# 大类网络
class Go_net(nn.Module):
    def __init__(self,input_size=98):
        super(Go_net,self).__init__()
        self.inception1 = inception(input_size,128,128,128,128)
        self.sblock5 = sblock(512,256)

        self.sblock7 = sblock(256,7)

        self.sequential = nn.Sequential(
            self.inception_re1,
            self.inception1,
            self.sblock5,
            self.sblock7
        )

    def forward(self,x):
        """
        前向传播
        :param X:
        :return:
        """
        return self.sequential(x)
```

初始化参数，避免梯度丢失，爆炸

```
# 初始化参数
def init_xavier(m):
    """
    xavier初始化可以避免梯度爆炸、梯度消失
```

```
    :param m::return:
    """
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
```

训练

在进行调参后找到比较好的模型参数

```
# 参数及网络初始化
batch_size = 1024
lr = 0.0001
epoch = 20000
m_Go_net = Go_net().to(torch.float32).to(device)

m_net = m_Go_net
m_net_name = 'm_Go_net'
m_net.apply(init_xavier)

x = x.to(torch.float32).to(device)
y = y.to(torch.float32).to(device)
m_net.to(device)
dataset = load_array((x,y),batch_size)

# 优化器
optimizer = torch.optim.Adam(m_net.parameters(),lr=lr,weight_decay=1e-4)
scheduler = lr_scheduler.StepLR(optimizer,step_size=100,gamma=0.9)

# 训练
ok_model =
train(data_loader=dataset,lr=lr,net=m_net,epochs=epoch,optimizer=optimizer,
scheduler=scheduler,device=device,flag=m_net_name)
```

可以在在线平台上训练，免费使用TPU

测试

小类

```
---best_model_release  epoch 1,train_loss 42.280,train_acc 0.183,valid_acc 0.015
  epoch 1,train_loss 42.280,train_acc 0.183,valid_acc 0.015
---best_model_release  epoch 3,train_loss 31.010,train_acc 0.379,valid_acc 0.235
---best_model_release  epoch 4,train_loss 28.653,train_acc 0.427,valid_acc 0.340
---best_model_release  epoch 5,train_loss 26.969,train_acc 0.463,valid_acc 0.416
---best_model_release  epoch 6,train_loss 25.665,train_acc 0.488,valid_acc 0.459
---best_model_release  epoch 7,train_loss 24.620,train_acc 0.507,valid_acc 0.483
---best_model_release  epoch 8,train_loss 23.722,train_acc 0.526,valid_acc 0.496
---best_model_release  epoch 9,train_loss 22.925,train_acc 0.538,valid_acc 0.506
---best_model_release  epoch 10,train_loss 22.212,train_acc 0.550,valid_acc 0.514
---best_model_release  epoch 11,train_loss 21.566,train_acc 0.558,valid_acc 0.521
---best_model_release  epoch 12,train_loss 20.953,train_acc 0.572,valid_acc 0.527
---best_model_release  epoch 13,train_loss 20.383,train_acc 0.582,valid_acc 0.533
---best_model_release  epoch 14,train_loss 19.843,train_acc 0.592,valid_acc 0.539
---best_model_release  epoch 15,train_loss 19.347,train_acc 0.600,valid_acc 0.540
---best_model_release  epoch 16,train_loss 18.896,train_acc 0.607,valid_acc 0.546
---best_model_release  epoch 17,train_loss 18.469,train_acc 0.616,valid_acc 0.550
---best_model_release  epoch 18,train_loss 18.022,train_acc 0.623,valid_acc 0.552
---best_model_release  epoch 19,train_loss 17.613,train_acc 0.632,valid_acc 0.557
---best_model_release  epoch 20,train_loss 17.204,train_acc 0.641,valid_acc 0.562
---best_model_release  epoch 21,train_loss 16.840,train_acc 0.647,valid_acc 0.565
---best_model_release  epoch 22,train_loss 16.500,train_acc 0.653,valid_acc 0.569
---best_model_release  epoch 23,train_loss 16.150,train_acc 0.661,valid_acc 0.573
---best_model_release  epoch 24,train_loss 15.808,train_acc 0.670,valid_acc 0.573
---best_model_release  epoch 25,train_loss 15.495,train_acc 0.676,valid_acc 0.576
---best_model_release  epoch 26,train_loss 15.160,train_acc 0.683,valid_acc 0.577
---best_model_release  epoch 27,train_loss 14.849,train_acc 0.692,valid_acc 0.578
---best_model_release  epoch 28,train_loss 14.538,train_acc 0.696,valid_acc 0.580
---best_model_release  epoch 29,train_loss 14.241,train_acc 0.704,valid_acc 0.580
---best_model_release  epoch 31,train_loss 13.667,train_acc 0.717,valid_acc 0.584
---best_model_release  epoch 32,train_loss 13.403,train_acc 0.722,valid_acc 0.585
---best_model_release  epoch 34,train_loss 12.872,train_acc 0.736,valid_acc 0.590
---best_model_release  epoch 38,train_loss 11.850,train_acc 0.763,valid_acc 0.590
---best_model_release  epoch 40,train_loss 11.382,train_acc 0.776,valid_acc 0.593
---best_model_release  epoch 42,train_loss 10.898,train_acc 0.786,valid_acc 0.594
---best_model_release  epoch 44,train_loss 10.470,train_acc 0.800,valid_acc 0.595
---best_model_release  epoch 46,train_loss 10.029,train_acc 0.811,valid_acc 0.596
  epoch 101,train_loss 3.341,train_acc 0.953,valid_acc 0.590
  epoch 201,train_loss 1.470,train_acc 0.965,valid_acc 0.592
---best_model_release  epoch 289,train_loss 1.233,train_acc 0.966,valid_acc 0.597
  epoch 301,train_loss 1.223,train_acc 0.965,valid_acc 0.593
```

大类

```
---best_model_release  epoch 19,train_loss 5.134,train_acc 0.874,valid_acc 0.893
---best_model_release  epoch 21,train_loss 5.068,train_acc 0.874,valid_acc 0.894
---best_model_release  epoch 24,train_loss 4.893,train_acc 0.875,valid_acc 0.895
---best_model_release  epoch 25,train_loss 4.865,train_acc 0.879,valid_acc 0.896
---best_model_release  epoch 29,train_loss 4.816,train_acc 0.882,valid_acc 0.899
---best_model_release  epoch 33,train_loss 4.705,train_acc 0.882,valid_acc 0.899
---best_model_release  epoch 35,train_loss 4.641,train_acc 0.882,valid_acc 0.900
---best_model_release  epoch 36,train_loss 4.556,train_acc 0.882,valid_acc 0.901
---best_model_release  epoch 38,train_loss 4.532,train_acc 0.885,valid_acc 0.902
---best_model_release  epoch 40,train_loss 4.367,train_acc 0.889,valid_acc 0.903
---best_model_release  epoch 41,train_loss 4.398,train_acc 0.885,valid_acc 0.903
---best_model_release  epoch 42,train_loss 4.431,train_acc 0.888,valid_acc 0.905
---best_model_release  epoch 43,train_loss 4.353,train_acc 0.890,valid_acc 0.906
---best_model_release  epoch 44,train_loss 4.389,train_acc 0.887,valid_acc 0.909
---best_model_release  epoch 45,train_loss 4.277,train_acc 0.895,valid_acc 0.910
---best_model_release  epoch 46,train_loss 4.358,train_acc 0.892,valid_acc 0.912
---best_model_release  epoch 47,train_loss 4.237,train_acc 0.895,valid_acc 0.913
---best_model_release  epoch 50,train_loss 4.069,train_acc 0.899,valid_acc 0.914
---best_model_release  epoch 52,train_loss 4.161,train_acc 0.894,valid_acc 0.914
---best_model_release  epoch 53,train_loss 4.042,train_acc 0.896,valid_acc 0.915
---best_model_release  epoch 56,train_loss 4.018,train_acc 0.899,valid_acc 0.915
---best_model_release  epoch 73,train_loss 3.689,train_acc 0.903,valid_acc 0.916
---best_model_release  epoch 78,train_loss 3.617,train_acc 0.906,valid_acc 0.917
---best_model_release  epoch 79,train_loss 3.624,train_acc 0.905,valid_acc 0.917
---best_model_release  epoch 82,train_loss 3.653,train_acc 0.905,valid_acc 0.918
---best_model_release  epoch 86,train_loss 3.537,train_acc 0.909,valid_acc 0.919
---best_model_release  epoch 87,train_loss 3.558,train_acc 0.907,valid_acc 0.919
---best_model_release  epoch 88,train_loss 3.508,train_acc 0.909,valid_acc 0.920
---best_model_release  epoch 90,train_loss 3.530,train_acc 0.909,valid_acc 0.920
  epoch 101,train_loss 3.321,train_acc 0.910,valid_acc 0.918
---best_model_release  epoch 105,train_loss 3.310,train_acc 0.913,valid_acc 0.922
---best_model_release  epoch 147,train_loss 3.044,train_acc 0.918,valid_acc 0.923
---best_model_release  epoch 160,train_loss 2.851,train_acc 0.924,valid_acc 0.923
---best_model_release  epoch 192,train_loss 2.719,train_acc 0.924,valid_acc 0.924
  epoch 201,train_loss 2.702,train_acc 0.923,valid_acc 0.919
---best_model_release  epoch 236,train_loss 2.573,train_acc 0.927,valid_acc 0.925
---best_model_release  epoch 241,train_loss 2.538,train_acc 0.929,valid_acc 0.926
  epoch 301,train_loss 2.324,train_acc 0.935,valid_acc 0.924
---best_model_release  epoch 361,train_loss 2.237,train_acc 0.937,valid_acc 0.927
  epoch 401,train_loss 2.193,train_acc 0.937,valid_acc 0.924
  epoch 501,train_loss 2.040,train_acc 0.940,valid_acc 0.924
  epoch 601,train_loss 1.938,train_acc 0.944,valid_acc 0.922
  epoch 701,train_loss 1.934,train_acc 0.943,valid_acc 0.925
  epoch 801,train_loss 1.861,train_acc 0.946,valid_acc 0.922
```

测试结果

```
---best_model_release  epoch 656,train_loss 0.321,train_acc 0.983,valid_acc
0.620
```

```
  epoch 1201,train_loss 0.256,train_acc 0.985,valid_acc 0.611
```

```
---best_model_release  epoch 361,train_loss 2.237,train_acc 0.937,valid_acc
0.927
```

```
epoch 901,train_loss 1.887,train_acc 0.944,valid_acc 0.923
```

四、 实验分析

详细分析实验结果，包括程序中使用数据的基本情况、程序的正确性验证、实验的效果对比。

数据的基本情况

使用上次实验的两万条数据，经过清洗后用于训练模型

其中文本数据在编码和计算相关性后转为向量用于训练，缺失值用0补全并且设置额外的3个维度用于表示缺失

程序的正确性验证

模型在训练集，测试集上的准确率

```
---best_model_release  epoch 656,train_loss 0.321,train_acc 0.983,valid_acc 0.620

---best_model_release  epoch 361,train_loss 2.237,train_acc 0.937,valid_acc 0.927
```

实验结果检测

在全体数据上的准确率：

小类

```
val_loader = data.DataLoader(dataset, batch_size=1, shuffle=True)


def test(data):
    with torch.no_grad():
        m_Go_net_test.eval()
        # 确保数据转换为BFloat16
        data = data.to(torch.bfloat16)
        return m_Go_net_test(data)


t = 0
acc = 0
for x,y in val_loader:
    t+=1
    if c2i[int(test(x).argmax(1))] == c2i[int(y.argmax(1))]:
        acc += 1
print(acc/t)
```

在 2024.05.09 16:49:01 于 21s 320ms内执行

```
0.8751547533719946
```

大类:

```
        return m_Go_net_test(data)


t = 0
acc = 0
for x,y in val_loader:
    t+=1
    if int(test(x).argmax(1)) == int(y.argmax(1)):
        acc += 1
print(acc/t)
```

在 2024.05.09 20:16:45 于 19s 836ms内执行

```
0.9631198279794096
```

五、 问题和解决方案

详细列举实验中出现的问题以及自己所给出的解决方法。

数据清洗

文本

> 上次实验获得的数据中有文本信息，对此先使用网络上的模型将其转为向量，但是在训练过程中发现效果较差，推测可能是网络难以拟合词嵌入后的向量

解决：
手动将词嵌入后的向量与类别进行计算，然后计算他们之间的相关性
在这样处理后，模型的测试集准确率由0.4提升到0.6

网络搭建

参考：《动手学深度学习》搭建svm和并行网络

训练调参

尝试使用svm和并行网络，初始时svm表现较好，但是在手动计算相关性后并行网络的效果优于svm

最终在svm上调出了小类0.604的测试集准确率，并行网络上调出了小类0.620，大类0.927的测试集准确率
且svm更容易出现过拟合

```
svm -- train_acc 0.983,valid_acc 0.620
并行 -- train_acc 0.702,valid_acc 0.604
```

调参时网络层数过多容易导致过拟合，过少会导致欠拟合

初始的网络参数设置过少，导致训练集，测试集准确率都偏低，且`dropout`层参数设置过大影响最终结果

多次改变网络层架构和`lr`，`batch_size`，`weight_decay=1e-4`等参数后发现对于svm，三层网络可以得到比较好的结果，而对并行神经网络4层较好，还发现并行神经网络的每个输出大小最好不要低于输入大小

结果验证

编写代码在全部数据上进行了验证，准确率约为0.88

```python

def test(data):
    with torch.no_grad():
        m_Go_net_test.eval()
        # 确保数据转换为BFloat16
        data = data.to(torch.bfloat16)
        return m_Go_net_test(data)


t = 0
acc = 0
for x,y in val_loader:
    t+=1
    if c2i[int(test(x).argmax(1))] == c2i[int(y.argmax(1))]:
        acc += 1
print(acc/t)
```

在 2024.05.09 16:49:01 于 21s 320ms内执行

```
0.8751547533719946
```

六、 实验总结

总结实验效果，还有哪些地方可以继续改进。

实验效果

训练的模型在验证集上分别达到了0.62，0.927的准确率

数据

还可以基于网页提取更多特征信息，仅靠过去收集的五个信息对46类结果预测较为困难例如可以记录网页中的图片信息并基于此获得新的输入

基于文本信息获得更多维度的特征，例如可以计算数据简介中关键字出现的次数或者图片数量，特定标签数量等

此外，从网站上抽取的信息中包含上个月下载次数等与时间相关的信息，在检查时发现这一数据变化很大，可以考虑以此建立时序或者采用其平均值来进行计算

网站上还有很多数据没有大小类标签，可以考虑将他们也加入训练集

网络

可以尝试构建结构更好的网络，尽管使用了正则化，`dropout`等手段，但实验中所选的网络在训练后期均出现了严重的过拟合现象，