Assignment 4

方煊杰

复旦大学

16307130335@fudan.edu.cn

准备

package requirements:

- torch
- numpy
- fastNLP

```
#install packages
$pip install torch
$pip install numpy
$pip install fastNLP
```

实现

Part1 文本预处理

1. 获得数据集

```
from sklearn.datasets import fetch_20newsgroups

def get_text_classification_datasets(n_class):
    categories = ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
                  'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
                  'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
                  'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
                  'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian',
                  'talk.politics.guns', 'talk.politics.mideast',
                  'talk.politics.misc', 'talk.religion.misc']
    dataset_train = fetch_20newsgroups(subset='train',
                                       categories=categories[0: n_class],
                                       data_home='../../..')
    return dataset_train
```

数据集来源于 fetch_20newsgroups, 在官网上看到总共有20个分类，由此可以通过设定 n_class 来返回所需要种类数量的数据

2. 去除特殊字符

```
def remove_punc(s):
    for i in s:
        if i in string.punctuation or i in string.whitespace:
            s = s.replace(i, " ")
    return s
```

- 对于每一个文本，使用`string.functuation`以及`string.replace`来忽略所有的标点符号

- 使用`string.whitespace`以及`string.replace`将所有特殊类符号（\n, \t...) 替换为空格符
- 示例：

```
>>> s = "Leo, Life is always hard like this. "
>>> s = remove_punc(s)
>>> s
"Leo Life is always hard like this"
```

3. 构建DataSet

根据fastNLP中DataSet以及Instance用法，构建DataSet数据集

```
dataset = DataSet()
for i in range(len(train_data.data)):
    ans = remove_punc(train_data.data[i])
    dataset.append((Instance(content=ans, target=int(train_data.target[i]))))
dataset.apply(lambda x: x['content'].lower().split(),
              new_field_name='words', is_input=True)
```

- 对于每一个文本，首先调用`remove_punc`去除特殊字符
- 再用`Instance类`封装文本，传入进**dataset**, 保存字段为**“content”**以及**“target”**
- 使用`split()`以及`lower()`函数，将content分为单个单词，并转为小写，以字段**"words"**的形式存入
- 示例：

```
>>> dataset = DataSet()
>>> ans = "Leo Life is always hard like this"
>>> t = 2
>>> dataset.append((Instance(content=ans, target=2)))
>>> dataset.apply(lambda x: x['content'].lower().split(),
                  new_field_name='words', is_input=True)
>>> dataset[0]
{'content': "Leo Life is always hard like this" type=str,
 "target": 2 type=int,
 "words": ["leo", "life", "is", "always", "hard", "like", "this"] type=list}
```

4. 构建单词表

使用fastNLP中的Vacabulary类自动统计词频并构建单词表

```
vocab = Vocabulary(min_freq=0, unknown='<unk>', padding='<pad>')
for txt in dataset:
    vocab.add_word_lst(txt['words'])
vocab.build_vocab()
```

- 查阅文档后使用`add_word_lst`函数将数据集中每个单词（频率）加入词典，构建词典

- 关于 vocabulary 中的一些用法示例：

```
>>> vocab.add_word_lst(["leo", "life", "is", "always", "hard", "like", "this"])
>>> vocab.build_vocab()
>>> vocab.word2idx   # 输出词典
{'<pad>': 0, '<unk>': 1, 'leo': 2, 'life': 3, 'is': 4, 'always': 5, 'hard': 6, 'like':
7, 'this': 8}
>>> vocab.to_index('leo')
2
>>> vocab.to_index('love') # 不存在的设为 'unk'
1
>>> vocab.to_word(2)
'leo'
```

5. 构建词向量

现在我们已经有了单词数据集以及单词表，所以可以轻松地构建出词向量

```
dataset.apply(lambda x: [vocab.to_index(word) for word in x['words']],
new_field_name='index')
dataset.set_input("index")
dataset.set_target("target")
```

- 将dataset中每一个单词都映射为整数，并作为字段**index**存入
- 为了之后的训练，将index和target字段置为**input**和**target**
- 示例：

```
>>> dataset[0]
{'content': "Leo Life is always hard like this" type=str,
 "target": 2 type=int,
 "words": ["leo", "life", "is", "always", "hard", "like", "this"] type=list}
>>> vocab.word2idx
{'<pad>': 0, '<unk>': 1, 'leo': 2, 'life': 3, 'is': 4, 'always': 5, 'hard': 6, 'like':
7, 'this': 8}
>>> dataset.apply(lambda x: [vocab.to_index(word) for word in x['words']],
                            new_field_name='index')
>>> dataset[0]
{'content': "Leo Life is always hard like this" type=str,
 "target": 2 type=int,
 "words": ["leo", "life", "is", "always", "hard", "like", "this"] type=list
 "index":[2, 3, 4, 5, 6, 7, 8] type = list}
```

Part2 模型

1. 普通RNN

```
class rnn(nn.Module):
    def __init__(self, input_size=1000):
        super().__init__()
        self.rnn = nn.RNN(
            input_size=100,
            hidden_size=32,
            num_layers=1,
            batch_first=True
        )
        self.embedding = nn.Embedding(input_size, 100)
        self.out = nn.Linear(32, 4)

    def forward(self, index):
        data = self.embedding(index)
        output, hidden = self.rnn(data)
        output = self.out(output)
        # 仅仅获取 time seq 维度中的最后一个向量
        return {"pred": output[:,-1,:]}
```

- 为了使用 fastNLP 中的 trainer 进行训练，研究格式之后要取出序列中的最后一个向量，并返回一个 dict:

  `return {"pred": output[:,-1,:]}`

2. LSTM

ps:由于在 Assignment3 中我已经实现了手动numpy构造网络，所以在这里使用 pytorch 封装好的模型

```
class myLSTM(nn.Module):
    def __init__(self, input_size=1000):
        super().__init__()
        self.myLSTM = torch.nn.LSTM(
            input_size=100,
            hidden_size=32,
            num_layers=1,
            batch_first=True
        )
        self.embedding = nn.Embedding(input_size, 100)
        self.out = nn.Linear(32, 4)

    def forward(self, index):
        x = self.embedding(index)
        output, hidden = self.myLSTM(x)
        output = output[:,-1,:] # 取序列的最后一个
        x = self.out(output)
        return {"pred": x}
```

- 与RNN几乎同样

3. CNN

```
class cnn(nn.Module):
    def __init__(self, input_dim, max_len):
        super(cnn, self).__init__()
        vocb_size = input_dim
        dim = 100
        n_class = 4
        self.max_len = max_len
        self.embeding = nn.Embedding(vocb_size, dim)
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=100, out_channels=16, kernel_size=5,
                      stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_len)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=100, out_channels=16, kernel_size=4,
                      stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_len)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=100, out_channels=16, kernel_size=3,
                      stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_len)
        )
        self.out = nn.Linear(48, n_class)

    def forward(self, index):
        x = self.embeding(index)
        x = x.unsqueeze(3)
        x = x.permute(0, 2, 1, 3)
        x1 = self.conv1(x)
        x2 = self.conv2(x)
        x3 = self.conv3(x)
        x = torch.cat((x1, x2, x3), dim=1)
        # 转换为 (N, OUT*1*1)
        x = x.view(-1, x.size(1))
        output = self.out(x)
        return {"pred": output}
```

- 使用 `nn.Sequential` 来对网络进行封装（这样看起来比较清爽）
- nn.Conv2d 对输入的文本向量进行卷积，输入参数需要为四维: (batch, C_in, H, W)

  因此在forward中，我们要对输入文本向量进行扩展:

  `x = x.unsqueeze(3)`： 将原本的三维向量扩展为四维
- 同时由于conv2d针对的是第一维和第二维，所以要把x中的维度换个位置:

  `x = x.permute(0, 2, 1, 3)` ：将第二维和第三维换个位置
- 采用不同的卷积核对同一个向量进行卷积操作（提取不同的特征）：

`x = torch.cat((x1, x2, x3), dim=1)`

所以最后的out层要把三次输出的 out_channels 维度相加作为输入, 即 out_channels * 3

Part3 训练

1. Trainer of fastNLP

```
trainer = Trainer(tra, model,
                  loss=CrossEntropyLoss(pred="pred",target='target'),
                  optimizer=SGD(model.parameters(), lr=0.001),
                  n_epochs=5, dev_data=dev,
                  metrics=AccuracyMetric(pred="pred",target='target'), batch_size=1)
trainer.train()
```

直接调用fastNLP中封装好的trainer进行训练，大大减少代码数量

- 损失函数使用交叉熵（也来自fastNLP）， 计算输入为 forward返回的**"pred"**和数据集中的**'target'**， 对应字段名相符合
- 优化使用SGD，当然也可以轻易换成别的
- 设定准确度函数，同样输入为forward返回的**"pred"**和测试集中的**'target'**

2. 模型保存与载入

```
model = torch.load("checkpoint/model.pth")
torch.save(model, "checkpoint/model.pth")
```

使用

1. train.py

1.1 使用方法

使用`argparse`工具来封装整个训练过程，便于改变训练参数

```
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--methods", "-m", default="lstm", choices=["rnn", "lstm", "cnn"])
    parser.add_argument("--n_epochs", "-n", default=5, type=int)
    parser.add_argument("--embedding", "-e", default=100, type=int)
    parser.add_argument("--category", "-c", default=4, type=int)
    parser.add_argument("--batch", "-b", default=4, type=int)
    parser.add_argument("--learning_rate", "-l", default=0.005, type=float)
```

```
    args = parser.parse_args()
    if args.category > 20 or args.category < 1:
        raise Exception("the number of category must be between 1 and 20")
    train_data, test_data, dic_size= handle_data(args.category)
    if args.methods == "rnn":
        model = rnn(dic_size)
        output = "rnn_model.pth"
    elif args.methods == "lstm":
        model = myLSTM(dic_size)
        output = "lstm_model.pth"
    else:
        model = cnn(dic_size)
        output = "cnn_model.pth"
```

1.2 示例输入

```
# 默认输入维度(即embedding维度):100, 隐藏层维度: 100, 迭代次数: 5,  学习率: 0.005,  batch:4
# 生成模型的名字为"model_lstm.json"
$ python train.py

# 更改参数
$ python test.py -m cnn -n 10 -e 50 -c 8 -l 0.001
# 代表使用cnn, embeding维度设为50,  迭代10次,  选择8个分类,  学习率设为0.001
```

1.3 示例结果

```
fang@fang-HP-Pavilion-Notebook:~/桌面/PRML-Spring19-Fudan/assignment-4/16307130335$ python main.py -m cnn
input fields after batch(if batch size is 2):
        words: (1)type:numpy.ndarray (2)dtype:object, (3)shape:(2,)
        index: (1)type:torch.Tensor (2)dtype:torch.int64, (3)shape:torch.Size([2, 218])
target fields after batch(if batch size is 2):
        target: (1)type:torch.Tensor (2)dtype:torch.int64, (3)shape:torch.Size([2])

training epochs started 2019-05-28-19-49-25
Evaluation at Epoch 1/5. Step:599/2995. AccuracyMetric: acc=0.66147
```

2. test.py

1.1 使用方法

- 使用`argparse`工具获得模型路径

```
parser = argparse.ArgumentParser()
parser.add_argument("--file", "-f", default="model_cnn.pth")
parser.add_argument("--category", "-c", default=4, type=int)
args = parser.parse_args()
model = torch.load(args.file)
train_data, test_data, dic_size = handle_data(args.category)
t = Tester(test_data, model, metrics=AccuracyMetric(pred="pred",target='target'))
t.test()
```

1.2 示例结果

只是用来自己看看效果，因为这个数据集种类啥的要定下来，和之前训练的不能不同

```
/usr/bin/python3.6 /home/fang/桌面/PRML-Spring19-Fudan/assignment-4/16307130335/test.py
cnn_model.pth
[tester]
AccuracyMetric: acc=0.993318
```

结果

1. CNN

cnn牛逼！ 所以另外两个只跑了20分类的看一看，这个都跑了一遍，确实厉害

1.1 分类：4

```
/usr/bin/python3.6 /home/fang/桌面/PRML-Spring19-Fudan/assignment-4/16307130335/test.py
cnn_model.pth
[tester]
AccuracyMetric: acc=0.993318
```

1.2 分类： 8

```
fang@fang-HP-Pavilion-Notebook:~/桌面/PRML-Spring19-Fudan/assignment-4/16307130335$ python train.py -m cnn -n 3 -c 8
input fields after batch(if batch size is 2):
        words: (1)type:numpy.ndarray (2)dtype:object, (3)shape:(2,)
        index: (1)type:torch.Tensor (2)dtype:torch.int64, (3)shape:torch.Size([2, 273])
target fields after batch(if batch size is 2):
        target: (1)type:torch.Tensor (2)dtype:torch.int64, (3)shape:torch.Size([2])

training epochs started 2019-05-28-23-29-27
Evaluation at Epoch 1/3. Step:919/2757. AccuracyMetric: acc=0.940152
```

1.3 分类：20

```
fang@fang-HP-Pavilion-Notebook:~/桌面/PRML-Spring19-Fudan/assignment-4/16307130335$ python train.py -m cnn -n 10 -c 20 -b 1
input fields after batch(if batch size is 1):
        words: (1)type:numpy.ndarray (2)dtype:<U14, (3)shape:(1, 117)
        index: (1)type:torch.Tensor (2)dtype:torch.int64, (3)shape:torch.Size([1, 117])
target fields after batch(if batch size is 1):
        target: (1)type:torch.Tensor (2)dtype:torch.int64, (3)shape:torch.Size([1])

training epochs started 2019-05-28-23-45-56
Evaluation at Epoch 1/10. Step:9052/90520. AccuracyMetric: acc=0.477454

Evaluation at Epoch 2/10. Step:18104/90520. AccuracyMetric: acc=0.515473

Evaluation at Epoch 3/10. Step:27156/90520. AccuracyMetric: acc=0.566313

Evaluation at Epoch 4/10. Step:36208/90520. AccuracyMetric: acc=0.551282

Evaluation at Epoch 5/10. Step:45260/90520. AccuracyMetric: acc=0.56145

Evaluation at Epoch 6/10. Step:54312/90520. AccuracyMetric: acc=0.60389

Evaluation at Epoch 7/10. Step:63364/90520. AccuracyMetric: acc=0.6145

Evaluation at Epoch 8/10. Step:72416/90520. AccuracyMetric: acc=0.619363

Evaluation at Epoch 9/10. Step:81468/90520. AccuracyMetric: acc=0.649867

Evaluation at Epoch 10/10. Step:90520/90520. AccuracyMetric: acc=0.637931


In Epoch:9/Step:81468, got best dev performance:AccuracyMetric: acc=0.649867
Reloaded the best model.
```

继续训练准确度还在继续上升～～～～～

```
/usr/bin/python3.6 /home/fang/桌面/PRML-Spring19-Fudan/assignment-4/16307130335/test.py
cnn_model.pth
[tester]
AccuracyMetric: acc=0.658709
```

2. RNN

1.1 分类：20

```
Evaluation at Epoch 19/50. Step:171988/452600. AccuracyMetric: acc=0.366048

Evaluation at Epoch 20/50. Step:181040/452600. AccuracyMetric: acc=0.392573

Evaluation at Epoch 21/50. Step:190092/452600. AccuracyMetric: acc=0.360301

Evaluation at Epoch 22/50. Step:199144/452600. AccuracyMetric: acc=0.380637

Evaluation at Epoch 23/50. Step:208196/452600. AccuracyMetric: acc=0.407162

Epoch 24/50:  48%|                                                                                | 215650/452600 [8:00:40<9:55:44,  6.63it/s, loss:1.53699
Epoch 24/50:  48%|                                                                                | 216750/452600 [8:03:13<9:48:31,  6.68it/s, loss:1.85680

In Epoch:23/Step:208196, got best dev performance:AccuracyMetric: acc=0.407162
Reloaded the best model.
```

一夜过去了，只跑了24个batch，一怒之下把它停止，准确率当前为41%，心情沉重

3 LSTM

1.1 分类： 20

LSTM并没有好到哪里去，慢的一匹，让人难受。

```
[tester]
AccuracyMetric: acc=0.65252
```

2. 总结

玄学调参

真的是完全不知道什么样的参数能够比较好的训练出高的准确度。

CNN还可以，不管怎么说训练速度快而且准确度会比较高，但是RNN简直是 **扶不起的阿斗**。。。。

之前小规模测试发现准确率不高以为是迭代次数不够，于是决定跑久一点。

训练一晚上，早上起床一看， 45% ， 我：？？？？？

后来和同学交流讨论之后，用了一个奇怪的trick:

```
    def forward(self, index):
        data = self.embedding(index)
        output, hidden = self.rnn(data)
        output = self.out(output)
        # 仅仅获取 time seq 维度中的最后一个向量
        output = torch.mean(output, dim=1, keepdim=True)
        return {"pred": output[:,-1,:]}
```

在forward函数中把output中序列的向量求平均，然后收敛就加速了很多！！！！ 而且准确率也大幅度提高！！

关于fastNLP

1. 关于使用

在这次的作业中， 我使用了fastNLP中的Instance, Vacabulary, DataSet 等类对数据进行了预处理， 具体如何时候用可以见我在part1 文本预处理中的步骤。

之后使用了pytorch构造模型，然后通过对forward返回值以及dataset中字段的定义，使用Trainer 进行训练，大大降低了代码量。

对于已经生成好的模型，还使用了Tester ,AccuracyMetric 进行测试, 自动输出准确度，一行搞定。

2. 我的建议

fastNLP作为基于pytorch上的框架，给了我很大的便利。

在写第三次作业的时候，文档好像没有那么完善，于是就主要看那个**“一分钟/十分钟快速开始”**，但是在这一次作业框架更新后我发现文档突然变得十分详细（老哥辛苦了！）

然后在浏览完文档之后，我想着如果给某些函数加上具体一点的，十分简单的，甚至是无脑的 **输入输出** 示例，会不会更加有助于一些初学者学习使用。

所以我自己尝试**写了点示例**，在报告中 part1 文本预处理里， 用了一个具体的句子，把构建dataset到单词表所有的过程实现了一遍，**并且输出了每一步的变量**。的确非常蠢，我自己也觉得好傻.....不过我想对于完全初学者，可以帮助理解函数用法，毕竟自己写的东西会觉得很简单，但是看别人的代码会容易暴躁.....

以我的水平可能并不能够对这个框架进行改进（QAQ），只能以使用者的角度来建议，比如在进行训练的时候如果能够同时输出loss曲线可能会让人很爽～～～

最后，希望fastNLP能够越来越完善，成为进行nlp研究的学者中最常用的框架。