电影评论情感分类

1 实验概述

本实验目的是设计一个模型来对电影评论进行二元情感分类，判断该评论的内容是积极还是消极。

2 实验环境

2.1 硬件环境

- CPU：Intel i5-12500H
- GPU：NVIDIA Geforce RTX 2050（4GB）

2.2 软件环境

- Windows 11
- Anaconda

2.3 Conda环境

- python=3.9
- pytorch=2.0.0
- pytorch-cuda=11.8
- gensim=4.3.1

3 实验思路

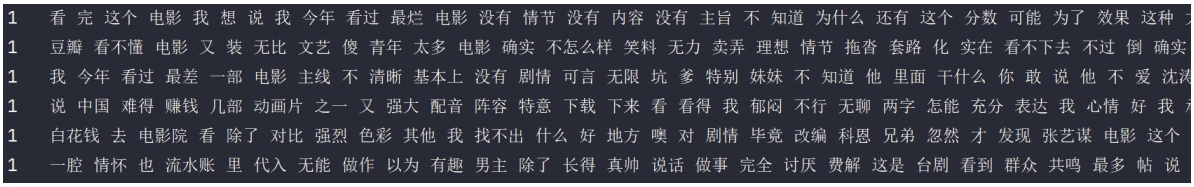
3.1 数据来源

本次实验的数据来自实验提供，包含4个文件：

- train.txt：训练集
- validation.txt：验证集
- test.txt：测试集
- wiki_word2vec_50.bin：词向量

3.2 数据预处理

原始样本集内容如图所示：



其中，第一个数字表示label，1是negative，0是positive。后续是文本内容，已经被分词过。

因此，所需要进行的预处理步骤为：

1. 统计样本集中所用到的词汇，整理出一个词汇索引表。

   对应函数实现：

```python
def build_word2id(file):
    word2id = {'_PAD_': 0}
    path = ['./data/train.txt', './data/validation.txt']
    for _path in path:
        with open(_path, encoding='utf-8') as f:
            for line in f.readlines():
                sp = line.strip().split()
                for word in sp[1:]:
                    if word not in word2id.keys():
                        word2id[word] = len(word2id)
    return word2id

word2id = build_word2id('./data/word2id.txt')
print(len(word2id))
```

2. 使用 `wiki_word2vec_50.bin` 文件来将词汇转化为向量。

   对应函数实现：

```python
def build_word2vec(filename, word2id, save_to_path=None):
    n_words = max(word2id.values()) + 1
    model = gensim.models.KeyedVectors.load_word2vec_format(filename, binary=True)
    word_vecs = np.array(np.random.uniform(-1., 1., [n_words, model.vector_size]))
    for word in word2id.keys():
        try:
            word_vecs[word2id[word]] = model[word]
        except KeyError:
            pass
    if save_to_path:
        with open(save_to_path, 'w', encoding='utf-8') as f:
            for vec in word_vecs:
                vec = [str(w) for w in vec]
```

```
                    f.write(' '.join(vec))
                    f.write('\n')
        return word_vecs

    def cat_to_id(classes=None):
        if not classes:
            classes = ['0', '1']
        cat2id = {cat: idx for (idx, cat) in enumerate(classes)}
        return classes, cat2id


    def load_corpus(path, word2id, max_sen_len=50):
        _, cat2id = cat_to_id()
        contents, labels = [], []
        with open(path, encoding='utf-8') as f:
            for line in f.readlines():
                sp = line.strip().split()
                if not sp:
                    continue
                label = sp[0]
                content = [word2id.get(w, 0) for w in sp[1:]]
                content = content[:max_sen_len]
                if len(content) < max_sen_len:
                    content += [word2id['_PAD_']] * (max_sen_len -
    len(content))
                labels.append(label)
                contents.append(content)
        counter = Counter(labels)
        print('\tSamples num: %d' % (len(labels)))
        for w in counter:
            print('\tLabel= {}: {}'.format(w, counter[w]))

        contents = np.asarray(contents)
        labels = np.array([cat2id[l] for l in labels])
        return contents, labels
```

3.3 模型选择与参数设置

模型网络选择 `TextCNN`

参数设置如下：

```
# 设置超参数
LR = 0.001          # 学习率
BATCH_SIZE = 32     # 训练批量
EPOCH_NUM = 9       # 训练轮数
NUM_WORKERS = 2
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# 设置模型参数
```

```
class CONFIG:
    update_w2v = True                   # 是否在训练中更新w2v
    vocab_size = word2vec.shape[0]      # 词汇量，与word2id中的词汇量一致
    n_class = 2                         # 分类数：分别为pos和neg
    embedding_dim = word2vec.shape[1]   # 词向量维度
    drop_prob = 0.5                     # dropout层，参数keep的比例
    num_filters = 128                   # 卷积层filter的数量
    kernel_size = 5                    # 卷积核的尺寸
    pretrained_embed = word2vec         # 预训练的词嵌入模型
```

4 实验代码

4.1 数据载入部分

```
# 加载训练集与验证集
print('Loading Train Dataset ... ')
train_contents, train_labels = load_corpus('./data/train.txt', word2id, max_sen_len=50)
print('Loading Valid Dataset ... ')
val_contents, val_labels = load_corpus('./data/validation.txt', word2id, max_sen_len=50)

train_dataset = TensorDataset(torch.from_numpy(train_contents).type(torch.float),

 torch.from_numpy(train_labels).type(torch.long))
valid_dataset = TensorDataset(torch.from_numpy(val_contents).type(torch.float),
                              torch.from_numpy(val_labels).type(torch.long))
train_dataloader = DataLoader(dataset = train_dataset, batch_size = BATCH_SIZE,
                              shuffle = True, num_workers = NUM_WORKERS)
valid_dataloader = DataLoader(dataset = valid_dataset, batch_size = BATCH_SIZE,
                              shuffle = True, num_workers = NUM_WORKERS)
```

4.2 模型部分

```
class TextCNN(nn.Module):
    def __init__(self, config):
        super(TextCNN, self).__init__()
        update_w2v = config.update_w2v
        vocab_size = config.vocab_size
        n_class = config.n_class
        embedding_dim = config.embedding_dim
        num_filters = config.num_filters
```

```
        kernel_size = config.kernel_size
        drop_prob = config.drop_prob
        pretrained_embed = config.pretrained_embed

        # 使用预训练的词向量
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.embedding.weight.data.copy_(torch.from_numpy(pretrained_embed))
        self.embedding.weight.requires_grad = update_w2v
        # 卷积层
        self.conv = nn.Conv2d(1,num_filters,(kernel_size,embedding_dim))
        # Dropout
        self.dropout = nn.Dropout(drop_prob)
        # 全连接层
        self.fc = nn.Linear(num_filters, n_class)

    def forward(self, x):
        x = x.to(torch.int64)
        x = self.embedding(x)
        x = x.unsqueeze(1)
        x = F.relu(self.conv(x)).squeeze(3)
        x = F.max_pool1d(x, x.size(2)).squeeze(2)
        x = self.dropout(x)
        x = self.fc(x)
        return x


config = CONFIG()
model = TextCNN(config)
model.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = LR)
criterion = nn.CrossEntropyLoss()
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.5)
```

• 4.3 训练部分

```
# train
from tqdm import tqdm

train_losses = []
train_counter = []
test_accuracy = [0.0]
test_counter = [i*len(train_dataloader.dataset) for i in range(EPOCH_NUM +
1)]

for epoch in range(0, EPOCH_NUM):
    model.train()
    sum_loss = 0.0
    correct = 0.0
    total = 0.0
```

```
14
15      loop = tqdm(train_dataloader, total = len(train_dataloader))
16      for i, data in enumerate(loop):
17          # prepare dataset
18          length = len(train_dataloader)
19          inputs, labels = data
20          inputs, labels = inputs.to(device), labels.to(device)
21          optimizer.zero_grad()
22
23          # forward & backward
24          outputs = model(inputs)
25          loss = criterion(outputs, labels)
26          loss.backward()
27          optimizer.step()
28
29          # print ac & loss in each batch
30          sum_loss += loss.item()
31          _, predicted = torch.max(outputs.data, 1)
32          total += labels.size(0)
33          correct += predicted.eq(labels.data).cpu().sum()
34
35          train_losses.append(sum_loss / (i + 1))
36          if len(train_counter) == 0:
37              train_counter.append(len(data))
38          else:
39              train_counter.append(train_counter[-1] + len(data))
40          loop.set_description(f'Epoch [{epoch + 1}/{EPOCH_NUM}]')
41          loop.set_postfix(loss = loss.item())
42
43      scheduler.step()
44      print('[epoch:%d] <Train> Loss: %.03f, Acc: %.3f%% ' % (epoch + 1,
    train_losses[-1], 100. * correct / total), end='\t')
45
46      # validation
47      with torch.no_grad():
48          correct = 0
49          total = 0
50          for data in valid_dataloader:
51              model.eval()
52              vec, labels = data
53              vec, labels = vec.to(device), labels.to(device)
54              outputs = model(vec)
55              _, predicted = torch.max(outputs.data, 1)
56              total += labels.size(0)
57              correct += (predicted == labels).sum()
58          print('<Valid> Acc: %.3f%%' % (100 * correct / total))
59          test_accuracy.append(correct.item() / total)
60
61
62  torch.save(model.state_dict(), './' + str(EPOCH_NUM) +'epoch-model.pth')
```

```
63  torch.save(optimizer.state_dict(), './' + str(EPOCH_NUM) +'epoch-
    optimizer.pth')
64  print('\nTrain has finished, total epoch is %d, model saved.' % EPOCH_NUM)
```

4.4 测试部分

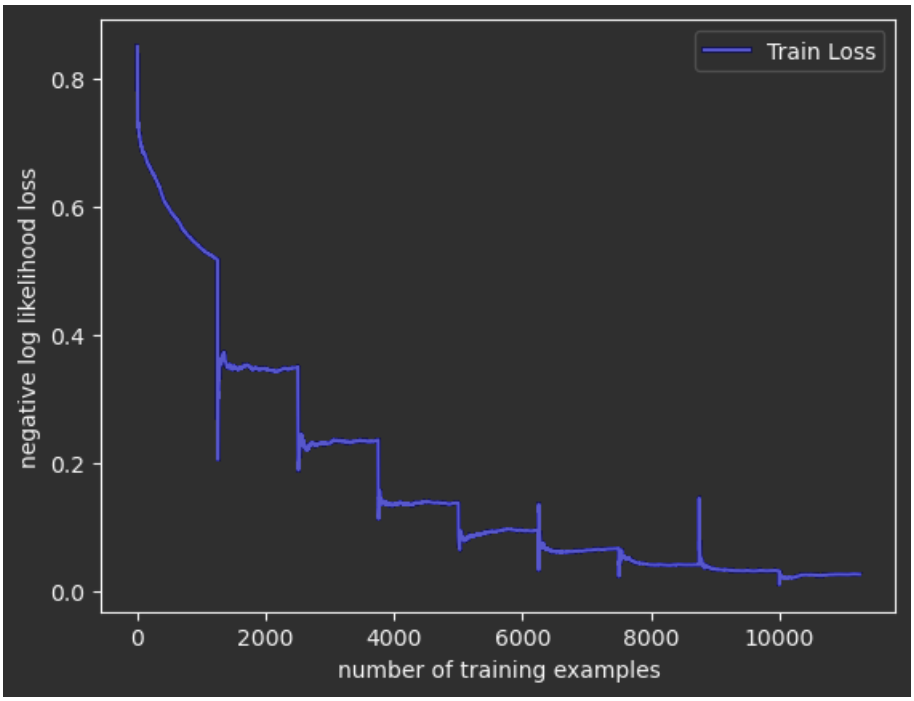
```
1   # 加载测试集
2   print('Loading Test Dataset...')
3   test_contents, test_labels = load_corpus('./data/test.txt', word2id,
    max_sen_len=50)
4
5   test_dataset =
    TensorDataset(torch.from_numpy(test_contents).type(torch.float),
6                                 torch.from_numpy(test_labels).type(torch.long))
7   test_dataloader = DataLoader(dataset = test_dataset, batch_size = BATCH_SIZE,
8                                 shuffle = True, num_workers = NUM_WORKERS)
9
10  config = CONFIG()
11  model_path = './9epoch-model.pth'
12  model = TextCNN(config)
13  model.load_state_dict(torch.load(model_path))
14  model.to(device)
15
16  with torch.no_grad():
17          correct = 0
18          total = 0
19          for data in tqdm(test_dataloader):
20              model.eval()
21              vec, labels = data
22              vec, labels = vec.to(device), labels.to(device)
23              outputs = model(vec)
24              _, predicted = torch.max(outputs.data, 1)
25              total += labels.size(0)
26              correct += (predicted == labels).sum()
27          print('<Test> Acc: %.3f%%' % (100 * correct / total))
28          test_accuracy.append(correct.item() / total)
```

5 实验结果

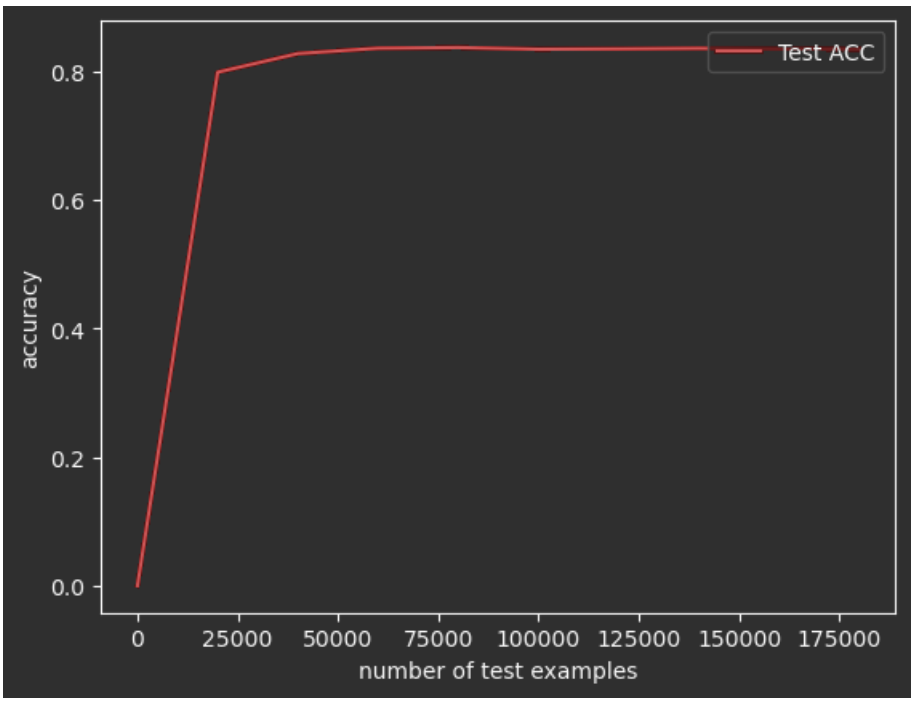
5.1 训练部分输出展示

```
Epoch [1/9]: 100%|██████████| 625/625 [00:10<00:00, 58.55it/s, loss=0.383]
[epoch:1] <Train> Loss: 0.517, Acc: 73.832%     <Valid> Acc: 79.819%
Epoch [2/9]: 100%|██████████| 625/625 [00:07<00:00, 84.95it/s, loss=0.486]
[epoch:2] <Train> Loss: 0.350, Acc: 85.059%     <Valid> Acc: 82.715%
 ...  ...
Epoch [9/9]: 100%|██████████| 625/625 [00:06<00:00, 92.07it/s, loss=0.00358]
[epoch:9] <Train> Loss: 0.026, Acc: 99.310%     <Valid> Acc: 83.390%
Train has finished, total epoch is 9, model saved.
```

- 5.2 Train Loss图像



- 5.3 Validation ACC图像



5.4 测试集ACC

```
100%|██████████| 12/12 [00:02<00:00,  5.24it/s]

<Test> Acc: 83.740%
```

6 总结

从训练结果来看，模型效果并不理想。该结果在训练过程中对验证集的效果就可以猜测到。模型在训练过程中，从欠拟合到过拟合，对验证集的正确率均在80%附近。

正常来讲不应该会出现这种情况，个人猜测原因为验证集+测试集的分布与训练集有较大差距，其中存在部分训练集中没有的样本，模型在训练时根本没见过类似的分布，从而造成当前对测试集和验证集正确率都只有83%的结果。

此外通过本实验，对文本类模型的使用了解了更多，对词向量的使用也学习了不少，对整个模型的训练与测试过程更加娴熟了。