实验 6：自然语言处理

2024 年 5 月 10 日

1 实验简介

本实验以豆瓣（douban.com）中文影评情感分析问题为例，介绍如何利用深度学习技术搭建自然语言处理系统。

在本实验中，我们将搭建一个 LSTM 模型，用于完成对电影评价文本的情感分类（积极评价或消极评价），并评估模型效果。通过本实验，你将掌握以下技能：

- 了解中文自然语言处理任务中常用的工具；
- 了解自然语言处理中数据预处理、数据加载和模型训练的一般流程；
- 了解情感分析模型的构建方法。

2 实验数据下载

通过以下链接下载实验所需数据及预训练权重文件：[Link]（提取码：n41n）

数据下载完成后，解压到当前 Notebook 文件所在目录下（不需要建立子目录）。

数据下载过程中，可同步进行实验环境准备。

本实验中使用的数据是通过网络爬虫获取的豆瓣电影影评数据，并已预先根据网页上的 “好评”、“差评” 选项进行了标注。

3 实验环境准备

安装本实验所需的必要组件。

3.1 安装 Miniconda

从以下链接下载 Miniconda 安装包并完成安装[Link]，安装过程中需要选中 `Register Anaconda as my default Python 3.10`。若计算机上已经安装好 Miniconda 或 Anaconda，请跳过此步骤。

3.2 安装必要的 Package

在 `Anaconda Prompt` 中，通过 `pip` 包管理器安装 `Jupyter Notebook`、`PyTorch`、`Gensim`、`jieba`、`tqdm` 及 `zhconv` 包。

```
pip install notebook -i https://pypi.tuna.tsinghua.edu.cn/simple/
pip install torch -i https://pypi.tuna.tsinghua.edu.cn/simple/
pip install gensim -i https://pypi.tuna.tsinghua.edu.cn/simple/
pip install jieba -i https://pypi.tuna.tsinghua.edu.cn/simple/
pip install tqdm -i https://pypi.tuna.tsinghua.edu.cn/simple/
pip install zhconv -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

当然，可以直接通过以下命令进行全局修改，而不必每次都指定-i 参数：

```
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

你可以通过执行以下 Python 代码，检查上述包是否已经安装完成。

[1]:
```
import pkgutil

# 检查 PyTorch 是否已安装
if pkgutil.find_loader('torch'):
    print('PyTorch is available.')

# 检查 Gensim 是否已安装
if pkgutil.find_loader('gensim'):
    print('Gensim is available.')

# 检查 jieba 是否已安装
if pkgutil.find_loader('jieba'):
    print('jieba is available.')

# 检查 tqdm 是否已安装
if pkgutil.find_loader('tqdm'):
    print('tqdm is available.')

# 检查 zhconv 是否已安装
if pkgutil.find_loader('zhconv'):
    print('zhconv is available.')
```

```
PyTorch is available.
Gensim is available.
jieba is available.
tqdm is available.
zhconv is available.
```

4 基础参数配置

首先导入必要的 Package。

[2]:
```
import torch
import os
import random
import re
import gensim
import jieba
import numpy as np

from tqdm.notebook import tqdm
from zhconv import convert

from torch import nn
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
from torch.utils.tensorboard import SummaryWriter
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence,␣
 ↪pad_packed_sequence
```

[3]:
```
# 忽略 Warning 输出
import warnings
warnings.filterwarnings("ignore")
```

创建一个 DictObj 类，用于存储必要的配置信息，方便后续使用。

定义 Config 对象，指定默认的配置信息（包括数据存放的位置、模型超参数等）。

[4]:
```
class DictObj(object):
    def __init__(self, mp):
        self.map = mp

    def __setattr__(self, name, value):
        if name == 'map':
            object.__setattr__(self, name, value)
            return
        self.map[name] = value

    def __getattr__(self, name):
        return self.map[name]
```

```
Config = DictObj({
    'train_path' : './train.txt',
    'test_path' : './test.txt',
    'validation_path' : './validation.txt',
    'pred_word2vec_path': './wiki_word2vec_50.bin',
    'tensorboard_path':'./tensorboard',
    'model_save_path':'./model/model.pth',
    'embedding_dim':50,
    'hidden_dim':100,
    'lr':0.001,
    'LSTM_layers':3,
    'drop_prob': 0.5,
    'seed':0
})
```

5 数据预处理

5.1 构建词汇表

利用训练数据构建词汇表，实现词与索引之间的转换。

构建词汇表的逻辑是：首先读取训练集数据，利用 zhconv 包将少部分繁体中文影评数据统一转换为简体中文。训练数据本身已经进行了分词，因此只需读入数据，再进行去重，即可构建词与索引的转换字典。

[5]:
```
def build_word_dict(train_path):
    words = []
    max_len = 0
    total_len = 0
    with open(train_path,'r',encoding='UTF-8') as f:
        lines = f.readlines()
        for line in  lines:
            line = convert(line, 'zh-cn')
            line_words = re.split(r'[\s]', line)[1:-1]
            max_len = max(max_len, len(line_words))
```

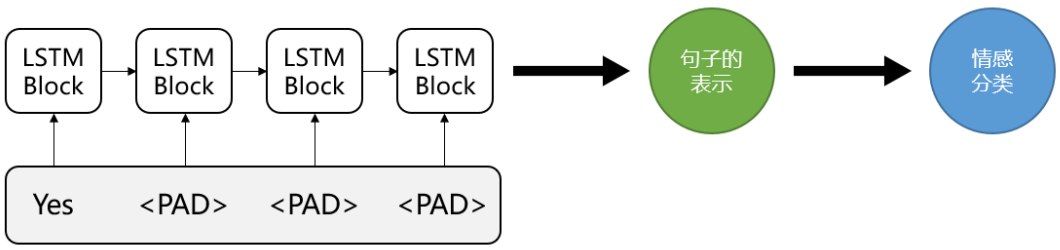
```
            total_len += len(line_words)
            for w in line_words:
                words.append(w)

    words = list(set(words))
    words = sorted(words)

    word2ix = {w:i+1 for i,w in enumerate(words)}
    ix2word = {i+1:w for i,w in enumerate(words)}
    word2ix['<unk>'] = 0
    ix2word[0] = '<unk>'
    avg_len = total_len / len(lines)
    return word2ix, ix2word, max_len, avg_len
```

[6]:
```
word2ix, ix2word, max_len, avg_len = build_word_dict(Config.train_path)
print(f'影评的最大长度为：{max_len}，平均长度为：{avg_len}。')
```

```
影评的最大长度为：679，平均长度为：44.67896789678968。
```

5.2 变长数据的处理

一般而言，LSTM 等 RNN 类的神经网络模型能够接受变长输入，也就是说，对于任意长度的影评数据，都不需要对模型结构进行改变。然而，深度学习框架（如 PyTorch）通常是以 batch 为单位进行训练的，这就涉及到变长序列的统一处理问题。目前常用的处理方法是，对文本数据按长度进行排序，之后按批次分别进行填充（Padding）处理，也就是 LSTM 单元数量总是取同一个批次最大的文本长度，长度不足的数据则使用零填充方法补齐长度。



[7]:
```
def mycollate_fn(data):
    # 这里的 data 是 getittem 返回的（input, label）的二元组，总共有 batch_size 个
    data.sort(key=lambda x: len(x[0]), reverse=True)  # 根据 input 来排序
```

```
    data_length = [len(sq[0]) for sq in data]
    input_data = []
    label_data = []
    for i in data:
        input_data.append(i[0])
        label_data.append(i[1])
    input_data = pad_sequence(input_data, batch_first=True, padding_value=0)
    label_data = torch.tensor(label_data)
    return input_data, label_data, data_length
```

5.3 数据集类的定义

定义一个影评数据集类，提供空白字符处理等基本功能。

[8]:
```
class CommentDataSet(Dataset):
    def __init__(self, data_path, word2ix, ix2word):
        self.data_path = data_path
        self.word2ix = word2ix
        self.ix2word = ix2word
        self.data, self.label = self.get_data_label()

    def __getitem__(self, idx: int):
        return self.data[idx], self.label[idx]

    def __len__(self):
        return len(self.data)

    def get_data_label(self):
        data = []
        label = []
        with open(self.data_path, 'r', encoding='UTF-8') as f:
            lines = f.readlines()
            for line in lines:
                try:
                    label.append(torch.tensor(int(line[0]), dtype=torch.int64))
```

```
                except BaseException:  # 遇到首个字符不是标签的就跳过比如空行，并
打印
                    print('not expected line:' + line)
                    continue
                line = convert(line, 'zh-cn')  # 转换成大陆简体
                line_words = re.split(r'[\s]', line)[1:-1]  # 按照空字符\t\n 空
格来切分
                words_to_idx = []
                for w in line_words:
                    try:
                        index = self.word2ix[w]
                    except BaseException:
                        index = 0  # 测试集，验证集中可能出现没有收录的词语，置为 0
                    words_to_idx.append(index)
                data.append(torch.tensor(words_to_idx, dtype=torch.int64))
        return data, label
```

5.4 数据集划分

根据配置信息构建训练集、验证集、测试集，并创建数据加载器（DataLoader）。

[9]:
```
train_data = CommentDataSet(Config.train_path, word2ix, ix2word)
train_loader = DataLoader(train_data, batch_size=16, shuffle=True,
                          num_workers=0, collate_fn=mycollate_fn,)

validation_data = CommentDataSet(Config.validation_path, word2ix, ix2word)
validation_loader = DataLoader(validation_data, batch_size=16, shuffle=True,
                               num_workers=0, collate_fn=mycollate_fn,)

test_data = CommentDataSet(Config.test_path, word2ix, ix2word)
test_loader = DataLoader(test_data, batch_size=16, shuffle=False,
                         num_workers=0, collate_fn=mycollate_fn,)
```

```
not expected line:

not expected line:
```

6 模型构建

6.1 Word2vec 预训练权重加载

使用 Gensim 加载中文 Word2vec 预训练权重，结合之前创建的词汇表，初始化 Embedding 层。

[10]:
```
word2vec_model = gensim.models.KeyedVectors.load_word2vec_format(Config.
 ↪pred_word2vec_path, binary=True)


def pre_weight(vocab_size):
    weight = torch.zeros(vocab_size,Config.embedding_dim)
    # 初始权重
    for i in range(len(word2vec_model.index_to_key)):  # 预训练中没有 word2ix，所
以只能用索引来遍历
        try:
            index = word2ix[word2vec_model.index_to_key[i]]  # 得到预训练中的词汇
的新索引
        except:
            continue
        weight[index, :] = torch.from_numpy(word2vec_model.get_vector(
            ix2word[word2ix[word2vec_model.index_to_key[i]]]))  # 得到对应的词向
量
    return weight
```

6.2 模型结构定义

使用 PyTorch 自带的 LSTM 模块创建模型，并在其后添加 3 个全连接层用于完成分类任务。

[11]:
```
class SentimentModel(nn.Module):
    def __init__(self, embedding_dim, hidden_dim,pre_weight):
        super(SentimentModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding.from_pretrained(pre_weight)
        self.embeddings.weight.requires_grad = True
        self.lstm = nn.LSTM(embedding_dim, self.hidden_dim, num_layers=Config.
 ↪LSTM_layers,
```

```
                            batch_first=True, dropout=Config.drop_prob,␣
↪bidirectional=False)
        self.dropout = nn.Dropout(Config.drop_prob)
        self.fc1 = nn.Linear(self.hidden_dim,256)
        self.fc2 = nn.Linear(256,32)
        self.fc3 = nn.Linear(32,2)

    def forward(self, input, batch_seq_len, hidden=None):
        embeds = self.embeddings(input)
        embeds = pack_padded_sequence(embeds,batch_seq_len, batch_first=True)
        batch_size, seq_len = input.size()
        if hidden is None:
            h_0 = input.data.new(Config.LSTM_layers*1, batch_size, self.
↪hidden_dim).fill_(0).float()
            c_0 = input.data.new(Config.LSTM_layers*1, batch_size, self.
↪hidden_dim).fill_(0).float()
        else:
            h_0, c_0 = hidden
        output, hidden = self.lstm(embeds, (h_0, c_0))
        output,_ = pad_packed_sequence(output,batch_first=True)

        output = self.dropout(torch.tanh(self.fc1(output)))
        output = torch.tanh(self.fc2(output))
        output = self.fc3(output)
        last_outputs = self.get_last_output(output, batch_seq_len)
        return last_outputs,hidden

    def get_last_output(self,output,batch_seq_len):
        last_outputs = torch.zeros((output.shape[0],output.shape[2]))
        for i in range(len(batch_seq_len)):
            last_outputs[i] =  output[i][batch_seq_len[i]-1]
        last_outputs = last_outputs.to(output.device)
        return last_outputs
```

6.3 模型性能评价指标

使用 TopK 的 AverageMeter 及混淆矩阵（ConfuseMeter）用于模型评价。

[12]:
```
class AvgrageMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.avg = 0
        self.sum = 0
        self.cnt = 0

    def update(self, val, n=1):
        self.sum += val * n
        self.cnt += n
        self.avg = self.sum / self.cnt

class ConfuseMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.confuse_mat = torch.zeros(2,2)
        self.tp = self.confuse_mat[0,0]
        self.fp = self.confuse_mat[0,1]
        self.tn = self.confuse_mat[1,1]
        self.fn = self.confuse_mat[1,0]
        self.acc = 0
        self.pre = 0
        self.rec = 0
        self.F1 = 0

    def update(self, output, label):
        pred = output.argmax(dim = 1)
        for l, p in zip(label.view(-1),pred.view(-1)):
            self.confuse_mat[p.long(), l.long()] += 1
```

```
        self.tp = self.confuse_mat[0,0]
        self.fp = self.confuse_mat[0,1]
        self.tn = self.confuse_mat[1,1]
        self.fn = self.confuse_mat[1,0]
        self.acc = (self.tp+self.tn) / self.confuse_mat.sum()
        self.pre = self.tp / (self.tp + self.fp)
        self.rec = self.tp / (self.tp + self.fn)
        self.F1 = 2 * self.pre*self.rec / (self.pre + self.rec)

def accuracy(output, label, topk=(1,)):
    maxk = max(topk)
    batch_size = label.size(0)

    _, pred = output.topk(maxk, 1, True, True) # 使用 topk 来获得前 k 个的索引
    pred = pred.t()
    correct = pred.eq(label.view(1, -1).expand_as(pred)) # 与正确标签序列形成的矩
阵相比，生成 True/False 矩阵

    rtn = []
    for k in topk:
        correct_k = correct[:k].reshape(-1).float().sum(0)
        rtn.append(correct_k.mul_(100.0 / batch_size))
    return rtn
```

7 模型训练

7.1 定义训练、验证及测试函数

定义训练函数，该函数包含了单个 epoch 的训练逻辑。

[13]:
```
def train(epoch,epochs, train_loader, device, model, criterion,␣
↪optimizer,scheduler,tensorboard_path):
    model.train()
    top1 = AvgrageMeter()
    model = model.to(device)
```

```
    train_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels, batch_seq_len = data[0].to(device), data[1].to(device),␣
↪data[2]
        # 初始为 0，清除上个 batch 的梯度信息
        optimizer.zero_grad()
        outputs,hidden = model(inputs,batch_seq_len)

        loss = criterion(outputs,labels)
        loss.backward()
        optimizer.step()
        _,pred = outputs.topk(1)
        prec1, prec2= accuracy(outputs, labels, topk=(1,2))
        n = inputs.size(0)
        top1.update(prec1.item(), n)
        train_loss += loss.item()
        postfix = {'train_loss': '%.6f' % (train_loss / (i + 1)), 'train_acc':␣
↪'%.6f' % top1.avg}
        train_loader.set_postfix(log=postfix)

        # ternsorboard 曲线绘制
        if os.path.exists(tensorboard_path) == False:
            os.mkdir(tensorboard_path)
        writer = SummaryWriter(tensorboard_path)
        writer.add_scalar('Train/Loss', loss.item(), epoch)
        writer.add_scalar('Train/Accuracy', top1.avg, epoch)
        writer.flush()

    scheduler.step()
```

类似地，定义验证、测试函数。

[14]:
```
def validate(epoch,validate_loader, device, model, criterion, tensorboard_path):
    val_acc = 0.0
    model = model.to(device)
    model.eval()
    with torch.no_grad():  # 进行评测的时候网络不更新梯度
```

```
        val_top1 = AvgrageMeter()
        validate_loader = tqdm(validate_loader)
        validate_loss = 0.0
        for i, data in enumerate(validate_loader, 0):
            inputs, labels, batch_seq_len = data[0].to(device), data[1].
 ↪to(device), data[2]
            outputs,_ = model(inputs, batch_seq_len)
            loss = criterion(outputs, labels)

            prec1, prec2 = accuracy(outputs, labels, topk=(1, 2))
            n = inputs.size(0)
            val_top1.update(prec1.item(), n)
            validate_loss += loss.item()
            postfix = {'validate_loss': '%.6f' % (validate_loss / (i + 1)),␣
 ↪'validate_acc': '%.6f' % val_top1.avg}
            validate_loader.set_postfix(log=postfix)

            # ternsorboard 曲线绘制
            if os.path.exists(tensorboard_path) == False:
                os.mkdir(tensorboard_path)
            writer = SummaryWriter(tensorboard_path)
            writer.add_scalar('Validate/Loss', loss.item(), epoch)
            writer.add_scalar('Validate/Accuracy', val_top1.avg, epoch)
            writer.flush()
        val_acc = val_top1.avg
    return val_acc

def test(validate_loader, device, model, criterion):
    val_acc = 0.0
    model = model.to(device)
    model.eval()
    confuse_meter = ConfuseMeter()
    with torch.no_grad():  # 进行评测的时候网络不更新梯度
        val_top1 = AvgrageMeter()
        validate_loader = tqdm(validate_loader)
        validate_loss = 0.0
```

```
        for i, data in enumerate(validate_loader, 0):
            inputs, labels, batch_seq_len = data[0].to(device), data[1].
↪to(device), data[2]
            outputs,_ = model(inputs, batch_seq_len)
            prec1, prec2 = accuracy(outputs, labels, topk=(1, 2))
            n = inputs.size(0)
            val_top1.update(prec1.item(), n)
            confuse_meter.update(outputs, labels)
            postfix = { 'test_acc': '%.6f' % val_top1.avg,
                      'confuse_acc': '%.6f' % confuse_meter.acc}
            validate_loader.set_postfix(log=postfix)
        val_acc = val_top1.avg
    return confuse_meter
```

7.2 设置随机种子

设置随机种子，保证模型在每次初始化时得到的是相同的权重，保证结果的可复现性。

[15]:
```
def set_seed(seed):
    np.random.seed(seed)
    random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True
```

[16]:
```
set_seed(Config.seed)
```

7.3 模型初始化

[17]:
```
model = SentimentModel(embedding_dim=Config.embedding_dim,
                       hidden_dim=Config.hidden_dim,
                       pre_weight=pre_weight(len(word2ix)))
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
optimizer = optim.Adam(model.parameters(), lr=Config.lr)
```

```
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size = 10, gamma=0.
↪1)
criterion = nn.CrossEntropyLoss()
print(model)
```

```
SentimentModel(
  (embeddings): Embedding(51404, 50)
  (lstm): LSTM(50, 100, num_layers=3, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=100, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=2, bias=True)
)
```

7.4 模型迭代训练

因为使用 tensorboard 画图会产生很多日志文件，这里进行清空操作。

[18]:
```
import shutil
if os.path.exists(Config.tensorboard_path):
    shutil.rmtree(Config.tensorboard_path)
    os.mkdir(Config.tensorboard_path)
```

使用 CPU 进行训练时，每个 epoch 约耗时 3~5 分钟，可根据实际算力进行调整。

[19]:
```
epochs = 3  # 按需调整

for epoch in range(epochs):
    train_loader = tqdm(train_loader)
    train_loader.set_description('[%s%04d/%04d %s%f]' % ('Epoch:', epoch + 1,␣
↪epochs, 'lr:', scheduler.get_lr()[0]))
    train(epoch, epochs, train_loader, device, model, criterion,␣
↪optimizer,scheduler, Config.tensorboard_path)
    validate(epoch, validation_loader,device,model,criterion,Config.
↪tensorboard_path)
```

```
  0%|          | 0/1250 [00:00<?, ?it/s]
```

```
  0%|          | 0/352 [00:00<?, ?it/s]

  0%|          | 0/1250 [00:00<?, ?it/s]

  0%|          | 0/352 [00:00<?, ?it/s]

  0%|          | 0/1250 [00:00<?, ?it/s]

  0%|          | 0/352 [00:00<?, ?it/s]
```

[20]:
```
# 模型保存
if os.path.exists(Config.model_save_path) == False:
    os.mkdir('./model/')
torch.save(model.state_dict(), Config.model_save_path)
```

8 模型测试

在测试集上评估模型性能，包括准确率、召回率、F1 值及混淆矩阵。

[21]:
```
model_test = SentimentModel(embedding_dim=Config.embedding_dim,
                            hidden_dim=Config.hidden_dim,
                            pre_weight=pre_weight(len(word2ix)))
optimizer_test = optim.Adam(model_test.parameters(), lr=Config.lr)
scheduler_test = torch.optim.lr_scheduler.StepLR(optimizer, step_size = 10,␣
 ↪gamma=0.1)
criterion_test = nn.CrossEntropyLoss()
model_test.load_state_dict(torch.load(Config.model_save_path), strict=True)
```

[21]:
```
<All keys matched successfully>
```

[22]:
```
confuse_meter = ConfuseMeter()
confuse_meter = test(test_loader,device,model_test,criterion_test)
print('prec:%.6f  recall:%.6f  F1:%.6f'%(confuse_meter.pre,confuse_meter.rec,␣
 ↪confuse_meter.F1))
```

```
  0%|          | 0/24 [00:00<?, ?it/s]

prec:0.511236  recall:1.000000  F1:0.676580
```

[23]:
```
confuse_meter.confuse_mat
```

[23]:
```
tensor([[182., 174.],
        [  0.,  13.]])
```

9 使用模型进行推理

为方便使用，定义一个预测函数。

[24]:
```
def predict(comment_str, model=model, device=device):
    model = model.to(device)
    seg_list = jieba.lcut(comment_str, cut_all=False)
    words_to_idx = []
    for w in seg_list:
        try:
            index = word2ix[w]
        except:
            index = 0
        words_to_idx.append(index)
    inputs = torch.tensor(words_to_idx).to(device)
    inputs = inputs.reshape(1,len(inputs))
    outputs,_ = model(inputs, [len(inputs),])
    pred = outputs.argmax(1).item()
    if pred:
        print("Negative")
    else:
        print("Positive")
    return pred
```

使用一些影评字符串进行测试。

[25]:
```
comment_str = "这一部导演、监制、男一都和《怒晴湘西》都是原班人马，这次是黄土高原上《龙岭密窟》的探险故事，有蝙蝠群、巨型蜘蛛这些让人瑟瑟发抖的元素，紧张刺激的剧情挺期待的。潘老师演技一如既往地稳。本来对姜超的印象也还在李大嘴这个喜剧角色里，居然没让人失望，还挺贴合王胖子这个角色。"
predict(comment_str)
```

```
Building prefix dict from the default dictionary …
Loading model from cache C:\Users\Lenovo\AppData\Local\Temp\jieba.cache
```

```
Loading model cost 0.565 seconds.
Prefix dict has been built successfully.

Positive
```

[25]: 0

[26]:
```
comment_str = "年代感太差，剧情非常的拖沓，还是冗余情节的拖沓。特效五毛，实在是太烂。潘粤明对这剧也太不上心了，胖得都能演王胖子了，好歹也锻炼一下。烂剧！"
predict(comment_str,model,device)
```

```
Negative
```

[26]: 1

尝试自己编写一些影评文本，观察模型输出结果是否符合预期。