核心代码

> 肖易佳 计83

MLP

```
class TwoLayerNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        """
            Use nn.Sequential, to make the structure clear
        """
        super(TwoLayerNet, self).__init__()
        self.twolayernet = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, output_size),
        )

    def forward(self, x):
        # x = np.mat(x)
        # x = x.view(-1, 300)
        """
            The backward gradient calculation will be automatically defined when
we have defined the forward
        """
        x = x.view(-1, 768)
        y_pred = self.twolayernet(x)
        return y_pred
input_size, hidden_size, output_size = 768, 256, 8
model = TwoLayerNet(input_size, hidden_size, output_size)
```

TextCNN

```
class TextCNN(nn.Module):
    def __init__(self, channel_out=16, kernel_size=[2,3,4], dropout=0.5,
pretrained_embed=Embed().idx_emb):
        super(TextCNN, self).__init__()
```

```
        channel_in = 1
        self.classes = 8
        self.embed_dim = 300
        self.embed = nn.Embedding(VOCAB_SIZE, self.embed_dim)
        #
self.embed.weight.data.copy_(torch.from_numpy(np.array(pretrained_embed)))

        self.conv1 = nn.Conv2d(in_channels=channel_in, out_channels=channel_out,
kernel_size=(kernel_size[0], self.embed_dim))
        self.conv2 = nn.Conv2d(in_channels=channel_in, out_channels=channel_out,
kernel_size=(kernel_size[1], self.embed_dim))
        self.conv3 = nn.Conv2d(in_channels=channel_in, out_channels=channel_out,
kernel_size=(kernel_size[2], self.embed_dim))

        self.dropout = nn.Dropout(dropout)
        self.fc1 = nn.Linear(len(kernel_size) * channel_out, 64)
        self.fc2 = nn.Linear(64, self.classes)

    def forward(self, x):
        x = x.to(device)
        x = self.embed(x)
        x = x.unsqueeze(1)

        x1=self.conv1(x)
        x1=F.relu(x1.squeeze(3))
        x1=F.max_pool1d(x1,x1.size(2)).squeeze(2)

        x2=self.conv2(x)
        x2=F.relu(x2.squeeze(3))
        x2=F.max_pool1d(x2,x2.size(2)).squeeze(2)

        x3=self.conv3(x)
        x3=F.relu(x3.squeeze(3))
        x3=F.max_pool1d(x3,x3.size(2)).squeeze(2)

        x=torch.cat((x1,x2,x3),1)
        x=self.dropout(x)

        x=self.fc1(x)
        x=F.relu(x)
        logit=self.fc2(x)
        return logit
```

RNN

```
class SeqRNN(nn.Module):
    '''
    vocab_size:词向量维度
    hidden_size:隐藏单元数量决定输出长度
    output_size:输出类别为8，维数为1
    '''
```

```
    def __init__(self, vocab_size=300, hidden_size=10, output_size=8,
pretrained_embed=Embed().idx_emb):
        super(SeqRNN, self).__init__()
        self.embed_dim = vocab_size
        self.embed = nn.Embedding(VOCAB_SIZE, self.embed_dim)
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.rnn = nn.RNN(self.vocab_size, self.hidden_size,
                          batch_first=True, dropout=0.5)
        self.linear = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input):
        input = self.embed(input)
        # print(input)
        # print('embeded size:', input.shape)
        h0 = torch.zeros(1, 1, self.hidden_size)
        h0 = h0.to(device)
        # print('h0 size:', h0.shape)
        output, hidden = self.rnn(input, h0)
        output = output[:, -1, :]
        output = self.linear(output)
        output = torch.nn.functional.softmax(output, dim=1)
        return output
```

Bert编码数据

```
(base) yijia@supermicro:~/emotion$

import json
from bert_serving.client import BertClient
from tqdm import tqdm
import numpy as np
import tensorflow as tf


bc = BertClient(port=5700, port_out=5701)

with open('data/test.text', 'r') as f:
    test = json.load(f)

with open('data/train.text', 'r') as f:
    train = json.load(f)

train = [l.split() for l in train]
test = [l.split() for l in test]

train_res = []
```

```
test_res = []

for l in tqdm(train):
train_res.append(bc.encode(l))
for l in tqdm(test):
    test_res.append(bc.encode(l))


np.save('train_res.npy', train_res)
np.save('test_res.npy', test_res)



IPython:
In [33]: for l in train[:2]:
    ...:     print(tf.reduce_mean(bc.encode(l), 0))
```

Bayes

```
word_stat = dict()
emotion_cnt = list()
for i in range(8):
    word_stat[i] = dict()
    emotion_cnt.append(0)
for text, label in zip(train_text_label[0], train_text_label[1]):
    for seg in text.split():
        if seg not in word_stat[np.argmax(label[1:])]:
            word_stat[np.argmax(label[1:])][seg] = 0
        word_stat[np.argmax(label[1:])][seg] += 1
    emotion_cnt[np.argmax(label[1:])] += 1

for i in range(8):
    toti = 0
    for freq in word_stat[i].values():
        toti += freq
    for k in word_stat[i]:
        word_stat[i][k] /= toti
    for k in word_stat[i]:
        word_stat[i][k] = math.log2(word_stat[i][k])

emotion_cnt = [math.log2(i) for i in emotion_cnt]

def bayes(factor):
    cnter = 0
    miss = 0
    for text, label in zip(test_text_label[0], test_text_label[1]):
        scores = list()
        segments = text.split()
        for i in range(8):
            # 类别不均衡，加上trade off 因子
```

```
            # score_i = emotion_cnt[i] * 2000
            # 1057
            score_i = emotion_cnt[i] * factor
            for seg in segments:
                if seg in word_stat[i]:
                    score_i += word_stat[i][seg]
            scores.append(score_i)
        if np.argmax(scores) == np.argmax(label[1:]):
            cnter += 1
    print('factor:', factor, 'cnt:', cnter, 'ratio:', cnter /
len(test_text_label[1]))
    return [factor, cnter, cnter / len(test_text_label[1])]
```

自定义数据集

```
class MyDataset(Dataset):
    """params: name, embed_file_path, name in ['train', 'test']"""
    def __init__(self, name, embed_path='../data/wordem.json', mode='int'):
        super(Dataset, self).__init__()
        self.embed = Embedding(embed_path)
        # with open('../data/' + name + '.text', 'r') as f:
            # text = json.load(f)
        # with open('../data/' + name + '.label', 'r') as f:
            # label = json.load(f)
        with open('../data/{}_text_label.json'.format(name), 'r') as f:
            text_label = json.load(f)
        text, label = text_label[0], text_label[1]
        self.feat = torch.Tensor([self.embed.embed(l) for l in text])
        if mode == 'int':
            self.label = [np.array(l[1:]) for l in label]
        elif mode == 'prob' or mode == 'float':
            self.label = list()
            for l in label:
                tot_vote = l[0]
                prob = np.array([vote / tot_vote for vote in l[1:]])
                self.label.append(prob)
            self.label = torch.Tensor(self.label)

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

    def __getitem__(self, item):
        return self.feat[item], self.label[item]
```

评估函数封装

```
# score_cor_ma_mi_wei(model)
# correlation_macro_micro_weighted
def score_cor_ma_mi_wei(net2):
```

```
    net2.eval()
    cnt = 0
    # for feat, label in testset:
    pred_emotions = list()
    true_emotions = list()
    for feat, label in testset:
        if torch.argmax(net2(torch.Tensor(feat))) ==
torch.argmax(torch.Tensor(label)):
            cnt += 1
        pred_emotions.append(int(torch.argmax(net2(torch.Tensor(feat))).item()))
        true_emotions.append(int(torch.argmax(torch.Tensor(label)).item()))
    # print('correct:', cnt)
    # print('ratio:', cnt / test_size)

    pears = list()
    for feat, label in testset:
        pred = torch.Tensor(net2(feat)[0]).detach().numpy()
        pears.append(pearsonr(label, pred))

    # print('corr:', np.mean(corr))
    # print('f_score_macro:', f1_score(true_emotions, pred_emotions,
average='macro'))
    # print('f_score_micro:', f1_score(true_emotions, pred_emotions,
average='micro'))
    # print('f_score_weighted:', f1_score(true_emotions, pred_emotions,
average='weighted'))
    return [np.mean(pears), f1_score(true_emotions, pred_emotions,
average='macro'),
           f1_score(true_emotions, pred_emotions, average='micro'),
f1_score(true_emotions, pred_emotions, average='weighted')]
```

绘图

```
import matplotlib.pyplot as plt
%matplotlib inline

def paint(name, color='red', leng=0):
    vals = eval(name + '_list')
    if leng == 0:
        leng = len(vals)
    vals = vals[:leng]
    x_axis = list(range(len(vals)))
    plt.figure(figsize=(12.8, 9.6))
    plt.xlim(0, np.max(x_axis) * 1.1)
    plt.ylim(0, np.max(vals) * 1.1)
    plt.xlabel('epoch')
    plt.ylabel('value')
    plt.text(x=np.max(x_axis) * 0.3, y=np.max(vals) * 0.9, s=name + '_curve',
fontsize=20)
    plt.plot(x_axis, vals, label=name, color=color)
```

```
        plt.legend()
        plt.savefig('mlp_{}{}.png'.format(name, leng))
```