# 对我们提取特征之后的数据重新利用神经网络进行学习

### 介绍

我们在word\_data提取了一系列的特征，比如senti(情感标签)、cixing(词性)、diversity(词的多义性)、freq(在字典中的词频)、vowel\_percentage(包含元音百分比)、if\_weekdays(是否工作日)、correation(单词字母相关度),在这些特征之外，我们还加入了月份和日期作为另外的两个维度，通过一共输入维度为9、输出维度为7、两个50维隐含层的神经网络输出最终的分布值，我们将数据划分70%为训练集、剩下30%为测试集，最终在测试集上所得MSE为0.0042,且由图片可以看出预测的分布较为精准。

# 读取word\_data.xlsx中的数据，将其转换为特征向量，然后使用简单的神经网络进行分类  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
from torchinfo import summary  
from simpleNet import SimpleNet

### 加载数据

# 读取数据  
data = pd.read\_excel('word\_data\_new.xlsx')  
data.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

Date
Date.1
Contest number
word
senti
senti\_score
cixing
diversity
freq\_1
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
1 try
2 tries
3 tries
4 tries
5 tries
6 tries
7 or more tries (X)
0
2022-12-31
44926
560
manly
1
0.0000
RB
0.620000
0.264214
0.265298
20.0
0
0.343806
0
2
17
37
29
12
2
1
2022-12-30
44925
559
molar
1
0.0000
JJ
0.777778
0.329989
0.315016
40.0
1
0.491583
0
4
21
38
26
9
1
2
2022-12-29
44924
558
havoc
0
-0.5994
NN
0.001000
0.254738
0.237445
40.0
1
0.097901
0
2
16
38
30
12
2
3
2022-12-28
44923
557
impel
1
0.0000
NN
0.500000
0.287625
0.314575
40.0
1
0.187709
0
3
21
40
25
9
1
4
2022-12-27
44922
556
condo
1
0.0000
NN
0.001000
0.267001
0.290084
40.0
1
0.308737
0
2
17
35
29
14
3

# 列出cixing的所有取值  
cixing = data['cixing'].unique()  
print(cixing)

['RB' 'JJ' 'NN' 'IN' 'VBD' 'VB' 'VBN' 'NNS' 'VBP' 'EX' 'NNP' 'VBZ' 'CC'  
 'VBG' 'FW' 'RBR' 'JJR' 'MD' 'JJS' 'PRP$' 'DT']

# 将cixing的取值转换为数字  
# 先制作一个map ['RB' 'JJ' 'NN' 'IN' 'VBD' 'VB' 'VBN' 'NNS' 'VBP' 'EX' 'NNP' 'VBZ' 'CC' 'VBG' 'FW' 'RBR' 'JJR' 'MD' 'JJS' 'PRP$' 'DT']  
cixing\_map = {'RB': 0, 'JJ': 1, 'NN': 2, 'IN': 3, 'VBD': 4, 'VB': 5, 'VBN': 6, 'NNS': 7, 'VBP': 8, 'EX': 9, 'NNP': 10, 'VBZ': 11, 'CC': 12, 'VBG': 13, 'FW': 14, 'RBR': 15, 'JJR': 16, 'MD': 17, 'JJS': 18, 'PRP$': 19, 'DT': 20}  
data['cixing'] = data['cixing'].map(cixing\_map)  
print(data['cixing'].unique())

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

data.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

Date
Date.1
Contest number
word
senti
senti\_score
cixing
diversity
freq\_1
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
1 try
2 tries
3 tries
4 tries
5 tries
6 tries
7 or more tries (X)
0
2022-12-31
44926
560
manly
1
0.0000
0
0.620000
0.264214
0.265298
20.0
0
0.343806
0
2
17
37
29
12
2
1
2022-12-30
44925
559
molar
1
0.0000
1
0.777778
0.329989
0.315016
40.0
1
0.491583
0
4
21
38
26
9
1
2
2022-12-29
44924
558
havoc
0
-0.5994
2
0.001000
0.254738
0.237445
40.0
1
0.097901
0
2
16
38
30
12
2
3
2022-12-28
44923
557
impel
1
0.0000
2
0.500000
0.287625
0.314575
40.0
1
0.187709
0
3
21
40
25
9
1
4
2022-12-27
44922
556
condo
1
0.0000
2
0.001000
0.267001
0.290084
40.0
1
0.308737
0
2
17
35
29
14
3

# 删除不需要的列 senti\_score, freq\_1  
data = data.drop(['senti\_score', 'freq\_1','Date.1'], axis=1)

data.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

Date
Contest number
word
senti
cixing
diversity
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
1 try
2 tries
3 tries
4 tries
5 tries
6 tries
7 or more tries (X)
0
2022-12-31
560
manly
1
0
0.620000
0.265298
20.0
0
0.343806
0
2
17
37
29
12
2
1
2022-12-30
559
molar
1
1
0.777778
0.315016
40.0
1
0.491583
0
4
21
38
26
9
1
2
2022-12-29
558
havoc
0
2
0.001000
0.237445
40.0
1
0.097901
0
2
16
38
30
12
2
3
2022-12-28
557
impel
1
2
0.500000
0.314575
40.0
1
0.187709
0
3
21
40
25
9
1
4
2022-12-27
556
condo
1
2
0.001000
0.290084
40.0
1
0.308737
0
2
17
35
29
14
3

# 构造输入输出数据  
data\_features = data.iloc[:,[0,3,4,5,6,7,8,9]]  
data\_distribution = data.iloc[:,10:]

data\_features.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

Date
senti
cixing
diversity
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
0
2022-12-31
1
0
0.620000
0.265298
20.0
0
0.343806
1
2022-12-30
1
1
0.777778
0.315016
40.0
1
0.491583
2
2022-12-29
0
2
0.001000
0.237445
40.0
1
0.097901
3
2022-12-28
1
2
0.500000
0.314575
40.0
1
0.187709
4
2022-12-27
1
2
0.001000
0.290084
40.0
1
0.308737

data\_distribution.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

1 try
2 tries
3 tries
4 tries
5 tries
6 tries
7 or more tries (X)
0
0
2
17
37
29
12
2
1
0
4
21
38
26
9
1
2
0
2
16
38
30
12
2
3
0
3
21
40
25
9
1
4
0
2
17
35
29
14
3

### 删除一些feature

# 删除不需要的列  
# data\_features = data\_features.drop(['cixing','senti'], axis=1)  
data\_features.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

Date
senti
cixing
diversity
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
0
2022-12-31
1
0
0.620000
0.265298
20.0
0
0.343806
1
2022-12-30
1
1
0.777778
0.315016
40.0
1
0.491583
2
2022-12-29
0
2
0.001000
0.237445
40.0
1
0.097901
3
2022-12-28
1
2
0.500000
0.314575
40.0
1
0.187709
4
2022-12-27
1
2
0.001000
0.290084
40.0
1
0.308737

### 转换数据

# 将data\_features第一列数据分成两列 即将Date列分成月和日两列  
data\_features['Month'] = data\_features['Date'].apply(lambda x: x.month)  
data\_features['Day'] = data\_features['Date'].apply(lambda x: x.day)  
data\_features = data\_features.drop(['Date'], axis=1)  
data\_features.head()

C:\Users\dongl\AppData\Local\Temp/ipykernel\_24168/1307404672.py:2: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 data\_features['Month'] = data\_features['Date'].apply(lambda x: x.month)  
C:\Users\dongl\AppData\Local\Temp/ipykernel\_24168/1307404672.py:3: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 data\_features['Day'] = data\_features['Date'].apply(lambda x: x.day)

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

senti
cixing
diversity
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
Month
Day
0
1
0
0.620000
0.265298
20.0
0
0.343806
12
31
1
1
1
0.777778
0.315016
40.0
1
0.491583
12
30
2
0
2
0.001000
0.237445
40.0
1
0.097901
12
29
3
1
2
0.500000
0.314575
40.0
1
0.187709
12
28
4
1
2
0.001000
0.290084
40.0
1
0.308737
12
27

# 转换数据格式  
data\_features\_np = data\_features.values  
data\_distribution\_np = data\_distribution.values  
print('data\_features\_np.shape = ', data\_features\_np.shape)  
print('data\_features\_np[:5,:] = \n', data\_features\_np[:5,:])  
print('data\_distribution\_np.shape = ', data\_distribution\_np.shape)  
print('data\_distribution\_np[:5,:] = \n', data\_distribution\_np[:5,:])

data\_features\_np.shape = (359, 9)  
data\_features\_np[:5,:] =   
 [[1.00000000e+00 0.00000000e+00 6.20000000e-01 2.65297965e-01  
 2.00000000e+01 0.00000000e+00 3.43805514e-01 1.20000000e+01  
 3.10000000e+01]  
 [1.00000000e+00 1.00000000e+00 7.77777778e-01 3.15016189e-01  
 4.00000000e+01 1.00000000e+00 4.91582513e-01 1.20000000e+01  
 3.00000000e+01]  
 [0.00000000e+00 2.00000000e+00 1.00000000e-03 2.37445032e-01  
 4.00000000e+01 1.00000000e+00 9.79006807e-02 1.20000000e+01  
 2.90000000e+01]  
 [1.00000000e+00 2.00000000e+00 5.00000000e-01 3.14574926e-01  
 4.00000000e+01 1.00000000e+00 1.87708873e-01 1.20000000e+01  
 2.80000000e+01]  
 [1.00000000e+00 2.00000000e+00 1.00000000e-03 2.90084013e-01  
 4.00000000e+01 1.00000000e+00 3.08737246e-01 1.20000000e+01  
 2.70000000e+01]]  
data\_distribution\_np.shape = (359, 7)  
data\_distribution\_np[:5,:] =   
 [[ 0 2 17 37 29 12 2]  
 [ 0 4 21 38 26 9 1]  
 [ 0 2 16 38 30 12 2]  
 [ 0 3 21 40 25 9 1]  
 [ 0 2 17 35 29 14 3]]

# 做数据处理  
# 对data\_features\_np进行normalization  
features\_mean = np.mean(data\_features\_np, axis=0)  
features\_std = np.std(data\_features\_np, axis=0)  
data\_features\_np\_norm = (data\_features\_np - features\_mean) / features\_std  
# 将data\_distribution\_np中的数据转换为概率分布  
data\_distribution\_np = data\_distribution\_np / 100  
print('data\_features\_np\_norm[:5,:] = \n', data\_features\_np\_norm[:5,:])  
print('data\_distribution\_np[:5,:] = \n', data\_distribution\_np[:5,:])

data\_features\_np\_norm[:5,:] =   
 [[ 0.06666118 -0.86473559 0.30348369 -0.51696379 -1.27981959 -1.57652685  
 -0.04498888 1.58231754 1.72832776]  
 [ 0.06666118 -0.55827944 0.8165452 0.36174245 0.34561166 0.63430572  
 0.41264337 1.58231754 1.61368074]  
 [-2.59237924 -0.25182329 -1.70937944 -1.0092289 0.34561166 0.63430572  
 -0.80650105 1.58231754 1.49903372]  
 [ 0.06666118 -0.25182329 -0.0867321 0.35394368 0.34561166 0.63430572  
 -0.52838521 1.58231754 1.3843867 ]  
 [ 0.06666118 -0.25182329 -1.70937944 -0.07890199 0.34561166 0.63430572  
 -0.15358745 1.58231754 1.26973968]]  
data\_distribution\_np[:5,:] =   
 [[0. 0.02 0.17 0.37 0.29 0.12 0.02]  
 [0. 0.04 0.21 0.38 0.26 0.09 0.01]  
 [0. 0.02 0.16 0.38 0.3 0.12 0.02]  
 [0. 0.03 0.21 0.4 0.25 0.09 0.01]  
 [0. 0.02 0.17 0.35 0.29 0.14 0.03]]

### 划分数据集

# 将数据转换为tensor，并用device指定运行的设备  
device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
print('device = ', device)  
data\_features\_tensor = torch.tensor(data\_features\_np\_norm, dtype=torch.float32, device=device)  
data\_distribution\_tensor = torch.tensor(data\_distribution\_np, dtype=torch.float32, device=device)

device = cpu

print('data\_features\_tensor.shape = ', data\_features\_tensor.shape)  
print('data\_features\_tensor[:5,:] = \n', data\_features\_tensor[:5,:])  
print('data\_distribution\_tensor.shape = ', data\_distribution\_tensor.shape)  
print('data\_distribution\_tensor[:5,:] = \n', data\_distribution\_tensor[:5,:])

data\_features\_tensor.shape = torch.Size([359, 9])  
data\_features\_tensor[:5,:] =   
 tensor([[ 0.0667, -0.8647, 0.3035, -0.5170, -1.2798, -1.5765, -0.0450, 1.5823,  
 1.7283],  
 [ 0.0667, -0.5583, 0.8165, 0.3617, 0.3456, 0.6343, 0.4126, 1.5823,  
 1.6137],  
 [-2.5924, -0.2518, -1.7094, -1.0092, 0.3456, 0.6343, -0.8065, 1.5823,  
 1.4990],  
 [ 0.0667, -0.2518, -0.0867, 0.3539, 0.3456, 0.6343, -0.5284, 1.5823,  
 1.3844],  
 [ 0.0667, -0.2518, -1.7094, -0.0789, 0.3456, 0.6343, -0.1536, 1.5823,  
 1.2697]])  
data\_distribution\_tensor.shape = torch.Size([359, 7])  
data\_distribution\_tensor[:5,:] =   
 tensor([[0.0000, 0.0200, 0.1700, 0.3700, 0.2900, 0.1200, 0.0200],  
 [0.0000, 0.0400, 0.2100, 0.3800, 0.2600, 0.0900, 0.0100],  
 [0.0000, 0.0200, 0.1600, 0.3800, 0.3000, 0.1200, 0.0200],  
 [0.0000, 0.0300, 0.2100, 0.4000, 0.2500, 0.0900, 0.0100],  
 [0.0000, 0.0200, 0.1700, 0.3500, 0.2900, 0.1400, 0.0300]])

# 划分训练集和测试集  
train\_size = int(0.7 \* data\_features\_tensor.shape[0])  
print('train\_size = ', train\_size)

train\_size = 251

# 先将数据打乱  
indices = torch.randperm(data\_features\_tensor.shape[0])  
data\_features\_tensor = data\_features\_tensor[indices]  
data\_distribution\_tensor = data\_distribution\_tensor[indices]

test\_indices = indices[train\_size:]  
test\_indices = test\_indices.cpu().numpy()

# 划分训练集和测试集  
train\_features = data\_features\_tensor[:train\_size,:]  
train\_distribution = data\_distribution\_tensor[:train\_size,:]  
test\_features = data\_features\_tensor[train\_size:,:]  
test\_distribution = data\_distribution\_tensor[train\_size:,:]  
  
print('train\_features.shape = ', train\_features.shape)  
print('train\_distribution.shape = ', train\_distribution.shape)  
print('test\_features.shape = ', test\_features.shape)  
print('test\_distribution.shape = ', test\_distribution.shape)

train\_features.shape = torch.Size([251, 9])  
train\_distribution.shape = torch.Size([251, 7])  
test\_features.shape = torch.Size([108, 9])  
test\_distribution.shape = torch.Size([108, 7])

# 制作dataloader  
batch\_size = 64  
train\_dataset = torch.utils.data.TensorDataset(train\_features, train\_distribution)  
test\_dataset = torch.utils.data.TensorDataset(test\_features, test\_distribution)  
train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  
test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

### 网络参数

inp\_dim = 9  
hidden\_dim = 50  
out\_dim = 7

net = SimpleNet(inp\_dim, hidden\_dim, out\_dim).to(device)

summary(net, (batch\_size,inp\_dim))

==========================================================================================  
Layer (type:depth-idx) Output Shape Param #  
==========================================================================================  
SimpleNet [64, 7] --  
├─Linear: 1-1 [64, 50] 500  
├─Linear: 1-2 [64, 50] 2,550  
├─Linear: 1-3 [64, 7] 357  
├─Softmax: 1-4 [64, 7] --  
==========================================================================================  
Total params: 3,407  
Trainable params: 3,407  
Non-trainable params: 0  
Total mult-adds (M): 0.22  
==========================================================================================  
Input size (MB): 0.00  
Forward/backward pass size (MB): 0.05  
Params size (MB): 0.01  
Estimated Total Size (MB): 0.07  
==========================================================================================

criterion = nn.MSELoss()  
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)

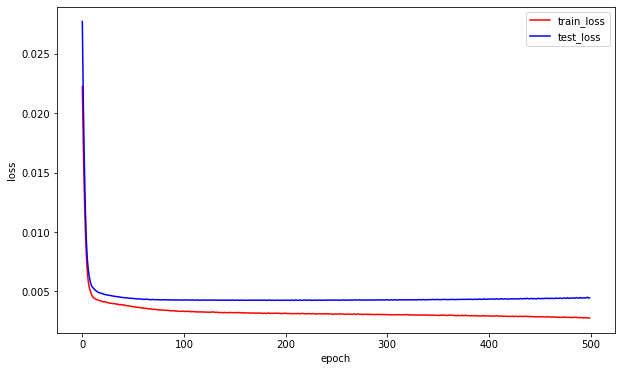
### 开始训练

# 训练  
# optimizer = torch.optim.SGD(net.parameters(), lr=1e-7)  
# optimizer = torch.optim.Adam(net.parameters(), lr=1e-4)  
num\_epochs = 500  
train\_loss\_list = []  
test\_loss\_list = []  
for epoch in range(num\_epochs):  
 train\_loss = 0  
 test\_loss = 0  
 net.train()  
 for i, (features, distribution) in enumerate(train\_loader):  
 optimizer.zero\_grad()  
 output = net(features)  
 loss = criterion(output, distribution)  
 loss.backward()  
 optimizer.step()  
 train\_loss += loss.item()  
 train\_loss = train\_loss / i  
 train\_loss\_list.append(train\_loss)  
 net.eval()  
 with torch.no\_grad():  
 for i, (features, distribution) in enumerate(test\_loader):  
 output = net(features)  
 loss = criterion(output, distribution)  
 test\_loss += loss.item()  
 test\_loss = test\_loss / i  
 test\_loss\_list.append(test\_loss)  
 if epoch % 10 == 0:  
 print('epoch = %d, train\_loss = %.4f, test\_loss = %.4f' % (epoch, train\_loss, test\_loss))

epoch = 0, train\_loss = 0.0222, test\_loss = 0.0277  
epoch = 10, train\_loss = 0.0046, test\_loss = 0.0053  
epoch = 20, train\_loss = 0.0041, test\_loss = 0.0048  
epoch = 30, train\_loss = 0.0040, test\_loss = 0.0046  
epoch = 40, train\_loss = 0.0039, test\_loss = 0.0045  
epoch = 50, train\_loss = 0.0037, test\_loss = 0.0044  
epoch = 60, train\_loss = 0.0036, test\_loss = 0.0043  
epoch = 70, train\_loss = 0.0035, test\_loss = 0.0043  
epoch = 80, train\_loss = 0.0034, test\_loss = 0.0043  
epoch = 90, train\_loss = 0.0034, test\_loss = 0.0043  
epoch = 100, train\_loss = 0.0033, test\_loss = 0.0043  
epoch = 110, train\_loss = 0.0033, test\_loss = 0.0043  
epoch = 120, train\_loss = 0.0033, test\_loss = 0.0043  
epoch = 130, train\_loss = 0.0032, test\_loss = 0.0043  
epoch = 140, train\_loss = 0.0032, test\_loss = 0.0043  
epoch = 150, train\_loss = 0.0032, test\_loss = 0.0042  
epoch = 160, train\_loss = 0.0032, test\_loss = 0.0043  
epoch = 170, train\_loss = 0.0032, test\_loss = 0.0043  
epoch = 180, train\_loss = 0.0032, test\_loss = 0.0043  
epoch = 190, train\_loss = 0.0031, test\_loss = 0.0042  
epoch = 200, train\_loss = 0.0032, test\_loss = 0.0042  
epoch = 210, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 220, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 230, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 240, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 250, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 260, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 270, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 280, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 290, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 300, train\_loss = 0.0031, test\_loss = 0.0043  
epoch = 310, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 320, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 330, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 340, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 350, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 360, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 370, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 380, train\_loss = 0.0030, test\_loss = 0.0043  
epoch = 390, train\_loss = 0.0029, test\_loss = 0.0044  
epoch = 400, train\_loss = 0.0029, test\_loss = 0.0043  
epoch = 410, train\_loss = 0.0029, test\_loss = 0.0043  
epoch = 420, train\_loss = 0.0029, test\_loss = 0.0044  
epoch = 430, train\_loss = 0.0029, test\_loss = 0.0044  
epoch = 440, train\_loss = 0.0029, test\_loss = 0.0044  
epoch = 450, train\_loss = 0.0029, test\_loss = 0.0044  
epoch = 460, train\_loss = 0.0028, test\_loss = 0.0044  
epoch = 470, train\_loss = 0.0028, test\_loss = 0.0044  
epoch = 480, train\_loss = 0.0028, test\_loss = 0.0044  
epoch = 490, train\_loss = 0.0028, test\_loss = 0.0045

### 训练过程loss-epoch曲线

# 画图  
plt.figure(figsize=(10, 6))  
plt.plot(train\_loss\_list, label='train\_loss', color='r')  
plt.plot(test\_loss\_list, label='test\_loss', color='b')  
plt.xlabel('epoch')  
plt.ylabel('loss')  
plt.legend()  
plt.show()



### 保存模型

# 保存模型  
# torch.save(net.state\_dict(), 'model\_useFeatures.pth')

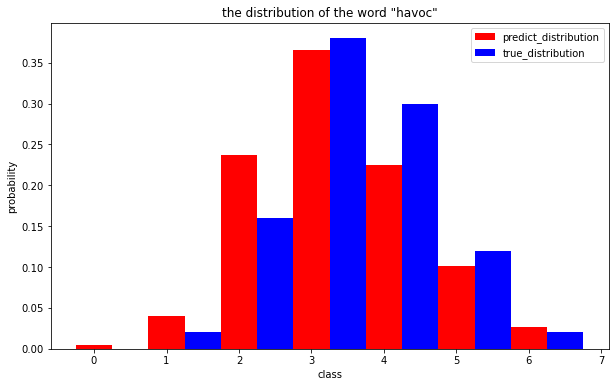
### 加载模型与评估

# # 加载模型  
# device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
# net = SimpleNet(inp\_dim, hidden\_dim, out\_dim).to(device)  
# net.load\_state\_dict(torch.load('model\_useFeatures.pth'))

# 从test\_features中随机抽取一个样本

idx = np.random.randint(0, test\_features.shape[0])  
random\_feature = test\_features[idx,:]  
# 找出该样本对应的单词  
right\_idx = test\_indices[idx]  
word = data['word'][right\_idx]  
# 推理  
net.eval()  
with torch.no\_grad():  
 output = net(test\_features[idx,:].unsqueeze(0))  
 output\_np = output.cpu().numpy()  
 print('output\_np = \n', output\_np)  
 print('test\_distribution\_np[idx,:] = \n', test\_distribution[idx,:].cpu().numpy())  
 # 画图  
 plt.figure(figsize=(10, 6))  
 plt.bar(np.arange(7), output\_np[0], width=0.5, label='predict\_distribution', color='r')  
 plt.bar(np.arange(7)+0.5, test\_distribution[idx,:].cpu().numpy() , width=0.5, label='true\_distribution', color='b')  
 plt.xlabel('class')  
 plt.ylabel('probability')  
 plt.title('the distribution of the word "{}"'.format(word))  
 plt.savefig('distribution\_of\_{}.png'.format(word))  
 plt.legend()  
 plt.show()

output\_np =   
 [[0.00492735 0.03992923 0.23680885 0.36537868 0.22463727 0.10132587  
 0.02699264]]  
test\_distribution\_np[idx,:] =   
 [0. 0.02 0.16 0.38 0.3 0.12 0.02]



#### 单词eerie的分布

| word | date | if\_weekday | senti | cixing | diversity | freq | vowel percentage(%) | correlations |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| eerie | 0 | 0/1 | 1 | NN(2) | 0.444444 | 0.482179277 | 80 | 1.4018470349381116 |

data\_features.head()

.dataframe tbody tr th:only-of-type {
vertical-align: middle;
}</body>

.dataframe tbody tr th {  
 vertical-align: top;  
}  
  
.dataframe thead th {  
 text-align: right;  
}

senti
cixing
diversity
freq\_2
vowel\_percentage(%)
if\_weekdays
correlations
Month
Day
0
1
0
0.620000
0.265298
20.0
0
0.343806
12
31
1
1
1
0.777778
0.315016
40.0
1
0.491583
12
30
2
0
2
0.001000
0.237445
40.0
1
0.097901
12
29
3
1
2
0.500000
0.314575
40.0
1
0.187709
12
28
4
1
2
0.001000
0.290084
40.0
1
0.308737
12
27

# data\_eerie\_np = np.array([1,0.4444444,0.482179277,80,0,1.4018470349381116]) # 删掉了cixing  
# data\_eerie\_np = np.array([0.4444444,0.482179277,80,0,1.4018470349381116]) # 删掉了cixing 和 senti  
data\_eerie\_np = np.array([1, 2, 0.4444444,0.482179277,80,0,1.4018470349381116,3,1])

# 将数据转换为tensor，并用device指定运行的设备  
data\_eerie\_np = (data\_eerie\_np - features\_mean) / features\_std  
data\_eerie\_tensor = torch.tensor(data\_eerie\_np, dtype=torch.float32, device=device)

# 推理  
net.eval()  
with torch.no\_grad():  
 output = net(data\_eerie\_tensor.unsqueeze(0))  
 output\_np = output.cpu().numpy()  
 print('output\_np = \n', output\_np)  
 # 画图  
 plt.figure(figsize=(10, 6))  
 plt.bar(np.arange(7), output\_np[0], width=0.5, label='predict\_distribution', color='r')  
 plt.xlabel('class')  
 plt.ylabel('probability')  
 plt.title('the distribution of the word "{}"'.format('eerie'))  
 plt.savefig('distribution\_of\_{}.png'.format('eerie'))  
 plt.legend()  
 plt.show()

output\_np =   
 [[0.00751348 0.08591553 0.20796828 0.3003361 0.25300962 0.1165532  
 0.02870386]]

