LING 530F Assignment 2

WASSA 2018 Implicit Emotion Shared Task

Yihao Zhang (82626169)

Meng Li (83978149)

1. Preprocess dataset

In [16]:

```
import csv
import re

from collections import Counter
from gensim.models import Word2Vec
from random import random
from nltk import word_tokenize
from nltk.translate.bleu_score import sentence_bleu
from torch import nn
from torch.autograd import Variable

import numpy as np
import torch
import torch.nn.functional as F
```

In [17]:

```
train_emotion = []
train_tweets = []
with open('dataset/train.csv', encoding='utf-8') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='"', skipinitialsp
ace=True)
    line_count = 0
    for row in spamreader:
        line_count += 1
        if line_count == 1: continue # skip header
        if not row: continue
        emotion = row[0]
        tweet = row[1]
        tweet = tweet.replace('@USERNAME', '')
        tweet = tweet.replace('[#TRIGGERWORD#]', '')
        tweet = result = re.sub(r"http\S+", "", tweet)
        train_tweets.append(tweet)
        train_emotion.append(emotion)
```

In [18]:

```
sentences = train_tweets

# Lower-case the sentence, tokenize them and add <SOS> and <EOS> tokens
sentences = [["<SOS>"] + word_tokenize(sentence.lower()) + ["<EOS>"] for sentenc
e in sentences]

# Create the vocabulary. Note that we add an <UNK> token to represent words not
in our vocabulary.
word_counts = Counter([word for sentence in sentences for word in sentence])
vocabulary = ["<UNK>"] + [e[0] for e in list(word_counts.items()) if e[1] > 2]
vocabularySize = len(vocabulary)
word2index = {word:index for index,word in enumerate(vocabulary)}
one_hot_embeddings = np.eye(vocabularySize)
```

In [19]:

```
# Create emotion array
emotions = sorted(list(set(train_emotion)))
emotions
```

Out[19]:

```
['anger', 'disgust', 'fear', 'joy', 'sad', 'surprise']
```

In [134]:

```
# Build the word2vec embeddings
wordEncodingSize = 300
filtered_sentences = [[word for word in sentence if word in word2index] for sent
ence in sentences]
w2v = Word2Vec(filtered_sentences, min_count=0, size=wordEncodingSize)
w2v_embeddings = np.concatenate((np.zeros((1, wordEncodingSize)), w2v.wv.vectors
))
```

In [21]:

```
def preprocess_numberize(sentence):
    """
    Given a sentence, in the form of a string, this function will preprocess it
    into list of numbers (denoting the index into the vocabulary).
    """
    tokenized = word_tokenize(sentence.lower())

    # Add the <SOS>/<EOS> tokens and numberize (all unknown words are represente
d as <UNK>).
    tokenized = ["<SOS>"] + tokenized + ["<EOS>"]
    numberized = [word2index.get(word, 0) for word in tokenized]

    return numberized

def preprocess_one_hot(sentence):
    """
    Given a sentence, in the form of a string, this function will preprocess it
    into a numpy array of one-hot vectors.
    """
    numberized = preprocess_numberize(sentence)

    # Represent each word as it's one-hot embedding
    one_hot_embedded = one_hot_embeddings[numberized]

    return one_hot_embedded

def preprocess_word2vec(sentence):
    """
    Given a sentence, in the form of a string, this function will preprocess it
    into a numpy array of word2vec embeddings.
    """
    numberized = preprocess_numberize(sentence)

    # Represent each word as it's one-hot embedding
    w2v_embedded = w2v_embeddings[numberized]

    return w2v_embedded

def compute_bleu(reference_sentence, predicted_sentence):
    """
    Given a reference sentence, and a predicted sentence, compute the BLEU simil
ary between them.
    """
    reference_tokenized = word_tokenize(reference_sentence.lower())
    predicted_tokenized = word_tokenize(predicted_sentence.lower())
    return sentence_bleu([reference_tokenized], predicted_tokenized)
```

1. Build a Emotion Decoder

In [129]:

```
use_cuda = False
class DecoderLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(DecoderLSTM, self).__init__()
        self.hidden_size = hidden_size

        self.lstm = nn.LSTM(input_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, input, hidden):
        output = F.relu(input)
        output, hidden = self.lstm(output, hidden)
        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
'''
# decoder for one hot embedding
decoder=DecoderLSTM(input_size=len(vocabulary),
                    hidden_size=300,
                    output_size=len(emotions))
'''
# decoder for word2vec embedding
decoder=DecoderLSTM(input_size=wordEncodingSize,
                    hidden_size=300,
                    output_size=len(emotions))
decoder
```

Out[129]:

```
DecoderLSTM(
  (lstm): LSTM(300, 300)
  (out): Linear(in_features=300, out_features=6, bias=True)
)
```

2. Train the Emotion Decoder

In [122]:

```
# build some helper function
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np

def showPlot(points):
    plt.figure()
    fig, ax = plt.subplots()
    loc = ticker.MultipleLocator(base=0.2)
    ax.yaxis.set_major_locator(loc)
    plt.plot(points)

import time
import math

def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)


def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (- %s)' % (asMinutes(s), asMinutes(rs))
```

In [124]:

```
def train(target_variable,
          emotion,
          decoder,
          decoder_optimizer,
          criterion,
          embeddings=w2v_embeddings,
          teacher_force=True):
    """
    Given a single training sample, go through a single step of training.
    """
    loss = 0
    decoder_optimizer.zero_grad()

    decoder_input = Variable(torch.FloatTensor([[embeddings[target_variable[0].d
ata[0]]]]))
    decoder_input = decoder_input.cuda() if use_cuda else decoder_input
    decoder_hidden = (decoder.initHidden(), decoder.initHidden())

    for di in range(0,target_variable.size(0)):
        decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
```

```
        topv, topi = decoder_output.data.topk(1)

        if teacher_force:
            ni = target_variable[di].data[0]
        else:
            ni = topi[0][0]

        decoder_input = Variable(torch.FloatTensor([[embeddings[ni]]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input
        if di == target_variable.size(0) - 2:
            loss += criterion(decoder_output, emotion)
        if vocabulary[ni] == "<EOS>":
            break

    loss.backward()

    torch.nn.utils.clip_grad_norm(decoder.parameters(), 10.0)

    decoder_optimizer.step()

    return loss.data[0] / target_variable.size(0)

decoder_optimizer = torch.optim.Adam(decoder.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

num_epochs = 1
numberized_emotion = [emotions.index(emotion) for emotion in train_emotion]
target_emotion = Variable(torch.LongTensor(numberized_emotion))
start = time.time()
total_loss = 0
avg_loss = []
for _ in range(num_epochs):
    for i,sentence in enumerate(train_tweets):

        numberized = preprocess_numberize(sentence)
        if len(numberized) == 2:
            continue
        target_variable = Variable(torch.LongTensor(numberized[1:]))

        loss = train(target_variable, target_emotion[i], decoder, decoder_optimi
zer, criterion)
        total_loss += loss
        avg_loss.append(total_loss/(i+1))
        if i % 1000 == 0:
            print('%s (%d %d%%) %.6f' %
                  (timeSince(start, (i+1)/len(train_tweets)), i, (i+1)/len(train
_tweets)*100, total_loss/(i+1)))
    name = 'decoder_biLSTM_ep' + str(_ + 1) + '.pt'
    torch.save(decoder.state_dict(), name)
```

```
0m 0s (- 319m 16s) (0 0%) 0.090516
2m 38s (- 403m 2s) (1000 0%) 0.101205
5m 18s (- 401m 19s) (2000 1%) 0.101878
7m 56s (- 397m 52s) (3000 1%) 0.101043
```

```
10m 36s (- 395m 49s) (4000 2%) 0.100553
13m 12s (- 391m 32s) (5000 3%) 0.100640
15m 49s (- 388m 35s) (6000 3%) 0.100328
18m 27s (- 385m 43s) (7000 4%) 0.100192
21m 4s (- 382m 36s) (8000 5%) 0.100152
23m 40s (- 379m 25s) (9000 5%) 0.100499
26m 15s (- 376m 7s) (10000 6%) 0.100300
28m 53s (- 373m 38s) (11000 7%) 0.099975
31m 30s (- 370m 52s) (12000 7%) 0.099711
34m 10s (- 368m 50s) (13000 8%) 0.099319
36m 45s (- 365m 42s) (14000 9%) 0.098944
39m 23s (- 363m 6s) (15000 9%) 0.098674
42m 0s (- 360m 21s) (16000 10%) 0.098387
44m 37s (- 357m 41s) (17000 11%) 0.098172
47m 16s (- 355m 16s) (18000 11%) 0.098007
49m 59s (- 353m 15s) (19000 12%) 0.097678
52m 42s (- 351m 12s) (20000 13%) 0.097320
55m 23s (- 348m 54s) (21000 13%) 0.097033
58m 6s (- 346m 45s) (22000 14%) 0.096798
60m 55s (- 345m 3s) (23000 15%) 0.096501
63m 42s (- 343m 7s) (24000 15%) 0.096229
66m 29s (- 341m 8s) (25000 16%) 0.096064
69m 22s (- 339m 36s) (26000 16%) 0.095832
72m 20s (- 338m 20s) (27000 17%) 0.095581
75m 15s (- 336m 44s) (28000 18%) 0.095440
78m 18s (- 335m 35s) (29000 18%) 0.095282
81m 22s (- 334m 23s) (30000 19%) 0.095149
84m 28s (- 333m 14s) (31000 20%) 0.094994
87m 41s (- 332m 22s) (32000 20%) 0.094745
90m 53s (- 331m 15s) (33000 21%) 0.094611
94m 10s (- 330m 23s) (34000 22%) 0.094410
97m 29s (- 329m 26s) (35000 22%) 0.094242
100m 54s (- 328m 44s) (36000 23%) 0.094111
104m 21s (- 327m 56s) (37000 24%) 0.094034
107m 50s (- 327m 11s) (38000 24%) 0.093885
111m 25s (- 326m 30s) (39000 25%) 0.093777
115m 2s (- 325m 48s) (40000 26%) 0.093616
118m 42s (- 325m 6s) (41000 26%) 0.093493
122m 21s (- 324m 11s) (42000 27%) 0.093341
126m 8s (- 323m 29s) (43000 28%) 0.093179
129m 51s (- 322m 30s) (44000 28%) 0.093029
133m 34s (- 321m 24s) (45000 29%) 0.092925
137m 21s (- 320m 20s) (46000 30%) 0.092840
141m 3s (- 318m 59s) (47000 30%) 0.092661
144m 48s (- 317m 37s) (48000 31%) 0.092538
148m 34s (- 316m 11s) (49000 31%) 0.092409
152m 18s (- 314m 37s) (50000 32%) 0.092263
156m 7s (- 313m 7s) (51000 33%) 0.092162
159m 59s (- 311m 36s) (52000 33%) 0.092020
163m 48s (- 309m 56s) (53000 34%) 0.091914
167m 41s (- 308m 18s) (54000 35%) 0.091810
171m 31s (- 306m 30s) (55000 35%) 0.091700
175m 32s (- 304m 56s) (56000 36%) 0.091551
```

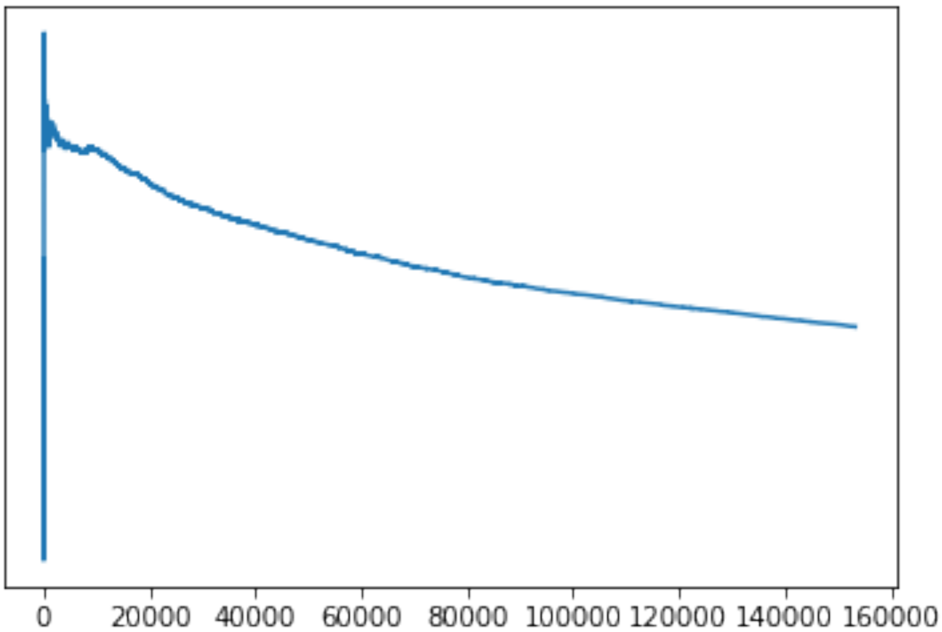
```
179m 32s (- 303m 17s) (57000 37%) 0.091390
183m 32s (- 301m 31s) (58000 37%) 0.091264
187m 37s (- 299m 49s) (59000 38%) 0.091090
191m 35s (- 297m 52s) (60000 39%) 0.091059
195m 39s (- 295m 59s) (61000 39%) 0.090903
199m 37s (- 293m 55s) (62000 40%) 0.090768
203m 37s (- 291m 47s) (63000 41%) 0.090716
207m 40s (- 289m 43s) (64000 41%) 0.090614
211m 41s (- 287m 31s) (65000 42%) 0.090481
215m 47s (- 285m 22s) (66000 43%) 0.090334
219m 49s (- 283m 5s) (67000 43%) 0.090227
223m 48s (- 280m 41s) (68000 44%) 0.090118
227m 53s (- 278m 21s) (69000 45%) 0.089984
231m 51s (- 275m 51s) (70000 45%) 0.089852
235m 47s (- 273m 15s) (71000 46%) 0.089778
239m 43s (- 270m 37s) (72000 46%) 0.089704
243m 40s (- 267m 59s) (73000 47%) 0.089622
247m 37s (- 265m 17s) (74000 48%) 0.089558
251m 36s (- 262m 36s) (75000 48%) 0.089447
255m 40s (- 259m 59s) (76000 49%) 0.089347
259m 42s (- 257m 17s) (77000 50%) 0.089229
263m 41s (- 254m 30s) (78000 50%) 0.089106
267m 45s (- 251m 45s) (79000 51%) 0.088964
271m 46s (- 248m 56s) (80000 52%) 0.088873
275m 45s (- 246m 5s) (81000 52%) 0.088791
279m 43s (- 243m 9s) (82000 53%) 0.088732
283m 43s (- 240m 15s) (83000 54%) 0.088654
287m 46s (- 237m 21s) (84000 54%) 0.088559
291m 49s (- 234m 26s) (85000 55%) 0.088460
295m 50s (- 231m 27s) (86000 56%) 0.088415
299m 44s (- 228m 21s) (87000 56%) 0.088359
303m 51s (- 225m 24s) (88000 57%) 0.088287
308m 1s (- 222m 28s) (89000 58%) 0.088187
312m 1s (- 219m 23s) (90000 58%) 0.088148
316m 3s (- 216m 19s) (91000 59%) 0.088086
320m 1s (- 213m 10s) (92000 60%) 0.088002
324m 6s (- 210m 5s) (93000 60%) 0.087898
328m 10s (- 206m 58s) (94000 61%) 0.087835
332m 5s (- 203m 44s) (95000 61%) 0.087764
336m 5s (- 200m 33s) (96000 62%) 0.087679
340m 3s (- 197m 19s) (97000 63%) 0.087632
344m 4s (- 194m 6s) (98000 63%) 0.087564
348m 3s (- 190m 50s) (99000 64%) 0.087528
352m 0s (- 187m 33s) (100000 65%) 0.087439
355m 56s (- 184m 15s) (101000 65%) 0.087387
359m 52s (- 180m 56s) (102000 66%) 0.087354
363m 50s (- 177m 37s) (103000 67%) 0.087292
367m 47s (- 174m 17s) (104000 67%) 0.087228
371m 51s (- 170m 59s) (105000 68%) 0.087164
375m 50s (- 167m 38s) (106000 69%) 0.087090
379m 53s (- 164m 19s) (107000 69%) 0.087013
383m 55s (- 160m 58s) (108000 70%) 0.086933
387m 53s (- 157m 35s) (109000 71%) 0.086851
```

```
391m 47s (- 154m 10s) (110000 71%) 0.086810
395m 48s (- 150m 46s) (111000 72%) 0.086733
399m 43s (- 147m 20s) (112000 73%) 0.086704
403m 43s (- 143m 55s) (113000 73%) 0.086646
407m 43s (- 140m 29s) (114000 74%) 0.086592
411m 44s (- 137m 4s) (115000 75%) 0.086551
415m 53s (- 133m 40s) (116000 75%) 0.086488
420m 0s (- 130m 15s) (117000 76%) 0.086389
424m 0s (- 126m 46s) (118000 76%) 0.086349
428m 0s (- 123m 18s) (119000 77%) 0.086302
432m 3s (- 119m 50s) (120000 78%) 0.086233
436m 17s (- 116m 24s) (121000 78%) 0.086199
440m 20s (- 112m 54s) (122000 79%) 0.086143
444m 28s (- 109m 26s) (123000 80%) 0.086091
448m 32s (- 105m 55s) (124000 80%) 0.086045
452m 34s (- 102m 24s) (125000 81%) 0.085999
456m 42s (- 98m 53s) (126000 82%) 0.085939
460m 43s (- 95m 21s) (127000 82%) 0.085897
464m 46s (- 91m 48s) (128000 83%) 0.085828
468m 53s (- 88m 16s) (129000 84%) 0.085794
473m 1s (- 84m 43s) (130000 84%) 0.085733
477m 4s (- 81m 9s) (131000 85%) 0.085674
481m 12s (- 77m 35s) (132000 86%) 0.085617
485m 19s (- 74m 1s) (133000 86%) 0.085561
489m 25s (- 70m 25s) (134000 87%) 0.085511
493m 34s (- 66m 50s) (135000 88%) 0.085430
497m 40s (- 63m 14s) (136000 88%) 0.085382
501m 49s (- 59m 38s) (137000 89%) 0.085341
505m 53s (- 56m 1s) (138000 90%) 0.085267
509m 55s (- 52m 24s) (139000 90%) 0.085215
514m 3s (- 48m 46s) (140000 91%) 0.085172
518m 17s (- 45m 9s) (141000 91%) 0.085124
522m 27s (- 41m 31s) (142000 92%) 0.085056
526m 37s (- 37m 52s) (143000 93%) 0.085012
530m 51s (- 34m 13s) (144000 93%) 0.084958
535m 6s (- 30m 34s) (145000 94%) 0.084902
539m 17s (- 26m 54s) (146000 95%) 0.084836
543m 25s (- 23m 13s) (147000 95%) 0.084798
547m 33s (- 19m 32s) (148000 96%) 0.084759
551m 44s (- 15m 51s) (149000 97%) 0.084714
555m 56s (- 12m 10s) (150000 97%) 0.084662
560m 7s (- 8m 28s) (151000 98%) 0.084609
564m 21s (- 4m 46s) (152000 99%) 0.084542
568m 34s (- 1m 3s) (153000 99%) 0.084494
```

In [130]:

```
len(train_tweets)
showPlot(avg_loss)
```

<Figure size 432x288 with 0 Axes>



In [56]:

```
# after training, save model
name = 'decoder4ep' + 'test' + '.pt'
torch.save(decoder.state_dict(), name)
```

In [131]:

```
# load previously training model:
decoder.load_state_dict(torch.load('decoder_nonstop_ep0.pt'))
```

3. Evaluate the Emotion decoder

In [125]:

```
dev_tweets = []
with open('dataset/dev.csv', encoding='utf-8') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='"', skipinitialsp
ace=True, quoting=csv.QUOTE_NONE)
    line_count = 0
    for row in spamreader:
        line_count += 1
        if line_count == 1: continue # skip header
        if not row: continue
        tweet = row[1]
        tweet = tweet.replace('@USERNAME', '')
        tweet = tweet.replace('[#TRIGGERWORD#]', '')
        tweet = result = re.sub(r"http\S+", "", tweet)
        dev_tweets.append(tweet)
```

In [126]:

```
dev_emotions = []
with open('dataset/trial-v3.csv') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='"', skipinitialsp
ace=True)
    line_count = 0
    for row in spamreader:
        line_count += 1
        if line_count == 1: continue # skip header
        if not row: continue
        dev_emotions.append(row[0])
```

In [132]:

```
actual_result = []
def evaluate(decoder,
             target_variable,
             embeddings=w2v_embeddings,
             teacher_force=True):

    decoder_input = Variable(torch.FloatTensor([[embeddings[target_variable[0].d
ata[0]]]]))
    decoder_input = decoder_input.cuda() if use_cuda else decoder_input
    decoder_hidden = (decoder.initHidden(),decoder.initHidden())

    softmax = nn.Softmax()
    for di in range(0,target_variable.size(0)):
        decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
        topv, topi = decoder_output.data.topk(1)

        if teacher_force:
            ni = target_variable[di].data[0]
        else:
            ni = topi[0][0]

        decoder_input = Variable(torch.FloatTensor([[embeddings[ni]]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        if di == target_variable.size(0) - 2: # last output
            actual_result.append(emotions[topi[0][0]])

            if dev_emotions[i] == emotions[topi[0][0]]:
                return True
            #print (dev_emotions[i], emotions[topi[0][0]])

        if vocabulary[ni] == "<EOS>":
            break
    return False

# evaluate the model
print ("ground truth, model prediction")
correct_prediction_counts = 0
for i,tweet in enumerate(dev_tweets):
    numberized = preprocess_numberize(tweet)
    if len(numberized) == 2: continue
    target_variable = Variable(torch.LongTensor(numberized[1:]))

    if evaluate(decoder, target_variable):
        correct_prediction_counts += 1

    if i % 100 == 0:
        print (correct_prediction_counts, " correct predictions in ", i+1)
        print ("acurray: ", correct_prediction_counts/(i+1))
```

ground truth, model prediction

```
1  correct predictions in  1
acurray:  1.0
51  correct predictions in  101
acurray:  0.504950495049505
101  correct predictions in  201
acurray:  0.5024875621890548
154  correct predictions in  301
acurray:  0.5116279069767442
205  correct predictions in  401
acurray:  0.5112219451371571
257  correct predictions in  501
acurray:  0.5129740518962076
305  correct predictions in  601
acurray:  0.5074875207986689
362  correct predictions in  701
acurray:  0.5164051355206848
405  correct predictions in  801
acurray:  0.5056179775280899
446  correct predictions in  901
acurray:  0.49500554938956715
490  correct predictions in  1001
acurray:  0.48951048951048953
540  correct predictions in  1101
acurray:  0.4904632152588556
587  correct predictions in  1201
acurray:  0.488759367194005
635  correct predictions in  1301
acurray:  0.4880860876249039
678  correct predictions in  1401
acurray:  0.48394004282655245
723  correct predictions in  1501
acurray:  0.4816788807461692
780  correct predictions in  1601
acurray:  0.4871955028107433
828  correct predictions in  1701
acurray:  0.48677248677248675
869  correct predictions in  1801
acurray:  0.4825097168239867
916  correct predictions in  1901
acurray:  0.4818516570226197
967  correct predictions in  2001
acurray:  0.4832583708145927
1010  correct predictions in  2101
acurray:  0.48072346501665875
1061  correct predictions in  2201
acurray:  0.48205361199454794
1101  correct predictions in  2301
acurray:  0.4784876140808344
1148  correct predictions in  2401
acurray:  0.478134110787172
1201  correct predictions in  2501
acurray:  0.4802079168332667
1255  correct predictions in  2601
```

```
acurray:  0.48250672818146867
1300  correct predictions in  2701
acurray:  0.4813032102924844
1344  correct predictions in  2801
acurray:  0.4798286326312031
1382  correct predictions in  2901
acurray:  0.47638745260255083
1427  correct predictions in  3001
acurray:  0.47550816394535156
1479  correct predictions in  3101
acurray:  0.47694292163818125
1521  correct predictions in  3201
acurray:  0.4751640112464855
1568  correct predictions in  3301
acurray:  0.4750075734625871
1611  correct predictions in  3401
acurray:  0.47368421052631576
1668  correct predictions in  3501
acurray:  0.47643530419880037
1709  correct predictions in  3601
acurray:  0.47459039155790056
1762  correct predictions in  3701
acurray:  0.47608754390705216
1815  correct predictions in  3801
acurray:  0.47750591949486976
1869  correct predictions in  3901
acurray:  0.4791079210458857
1915  correct predictions in  4001
acurray:  0.4786303424143964
1970  correct predictions in  4101
acurray:  0.4803706413069983
2014  correct predictions in  4201
acurray:  0.4794096643656272
2067  correct predictions in  4301
acurray:  0.48058591025342945
2112  correct predictions in  4401
acurray:  0.47989093387866394
2160  correct predictions in  4501
acurray:  0.4798933570317707
2211  correct predictions in  4601
acurray:  0.480547707020213
2258  correct predictions in  4701
acurray:  0.4803233354605403
2310  correct predictions in  4801
acurray:  0.4811497604665695
2351  correct predictions in  4901
acurray:  0.4796980208120792
2401  correct predictions in  5001
acurray:  0.48010397920415915
2452  correct predictions in  5101
acurray:  0.48069006077239756
2502  correct predictions in  5201
acurray:  0.4810613343587772
```

```
2546  correct predictions in  5301
acurray:  0.48028673835125446
2594  correct predictions in  5401
acurray:  0.4802814293649324
2641  correct predictions in  5501
acurray:  0.4800945282675877
2691  correct predictions in  5601
acurray:  0.48044991965720407
2734  correct predictions in  5701
acurray:  0.4795649885984915
2782  correct predictions in  5801
acurray:  0.4795724875021548
2825  correct predictions in  5901
acurray:  0.4787324182341976
2872  correct predictions in  6001
acurray:  0.4785869021829695
2921  correct predictions in  6101
acurray:  0.4787739714800852
2967  correct predictions in  6201
acurray:  0.4784712143202709
3022  correct predictions in  6301
acurray:  0.47960641168068563
3066  correct predictions in  6401
acurray:  0.4789876581784096
3113  correct predictions in  6501
acurray:  0.47884940778341795
3168  correct predictions in  6601
acurray:  0.47992728374488713
3216  correct predictions in  6701
acurray:  0.47992836890016416
3263  correct predictions in  6801
acurray:  0.4797823849433907
3314  correct predictions in  6901
acurray:  0.4802202579336328
3355  correct predictions in  7001
acurray:  0.47921725467790316
3397  correct predictions in  7101
acurray:  0.47838332629207153
3439  correct predictions in  7201
acurray:  0.47757255936675463
3491  correct predictions in  7301
acurray:  0.4781536775784139
3536  correct predictions in  7401
acurray:  0.4777732738819078
3579  correct predictions in  7501
acurray:  0.4771363818157579
3631  correct predictions in  7601
acurray:  0.4777003025917642
3675  correct predictions in  7701
acurray:  0.47721075185040907
3730  correct predictions in  7801
acurray:  0.47814382771439556
3777  correct predictions in  7901
```

```
acurray:  0.4780407543348943
3823  correct predictions in  8001
acurray:  0.47781527309086363
3874  correct predictions in  8101
acurray:  0.47821256634983333
3921  correct predictions in  8201
acurray:  0.4781124253139861
3967  correct predictions in  8301
acurray:  0.47789422961089023
4010  correct predictions in  8401
acurray:  0.4773241280799905
4057  correct predictions in  8501
acurray:  0.4772379720032937
4100  correct predictions in  8601
acurray:  0.47668875712126496
4140  correct predictions in  8701
acurray:  0.47580737846224574
4190  correct predictions in  8801
acurray:  0.4760822633791615
4232  correct predictions in  8901
acurray:  0.4754521963824289
4280  correct predictions in  9001
acurray:  0.4755027219197867
4333  correct predictions in  9101
acurray:  0.4761015273046918
4374  correct predictions in  9201
acurray:  0.475383110531464
4417  correct predictions in  9301
acurray:  0.4748951725620901
4479  correct predictions in  9401
acurray:  0.4764386767365174
4520  correct predictions in  9501
acurray:  0.4757393958530681
```

In [133]:

```
#!/usr/bin/env python
# Author: roman.klinger@ims.uni-stuttgart.de
# Evaluation script for IEST at WASSA 2018
from __future__ import print_function
import sys
import itertools
import pandas as pd
from IPython.display import display, HTML
import seaborn as sns
import matplotlib.pyplot as plt
import random

test_result = []
def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)

def welcome():
```

```
    eprint("=======================================")

    eprint("Evaluation script v0.2 for the Implicit Emotions Shared Task 2018.")
    eprint("Please call it via")
    eprint("./evaluate-iest.py <gold> <prediction>")
    eprint("where each csv file has labels in its first column.")
    eprint("The rows correspond to each other (1st row in <gold>")
    eprint("is the gold label for the 1st column in <prediction>).")
    eprint("")
    eprint("If you have questions, please contact klinger@wassa2018.com")
    eprint("====================================\n\n")

def checkParameters():
    if ((len(sys.argv) < 3 or len(sys.argv) > 3)):
        eprint("Please call the script with two files as parameters.")
        sys.exit(1)

def readFileToList(filename):
    eprint("Reading data from",filename)
    f=open(filename,"r")
    lines=f.readlines()
    result=[]
    for x in lines:
        result.append(x.split('\t')[0].rstrip())
    f.close()
    eprint("Read",len(result),"labels.")
    return result

def calculatePRF(gold,prediction):
    # initialize counters
    labels = set(gold+prediction)
    print("Labels: "+';'.join(labels))
    tp = dict.fromkeys(labels, 0.0)
    fp = dict.fromkeys(labels, 0.0)
    fn = dict.fromkeys(labels, 0.0)
    precision = dict.fromkeys(labels, 0.0)
    recall = dict.fromkeys(labels, 0.0)
    f = dict.fromkeys(labels, 0.0)
    # check every element
    for g,p in zip(gold,prediction):
        #    print(g,p)
        # TP
        if (g == p):
            tp[g] += 1
        else:
            fp[p] += 1
            fn[g] += 1
    # print stats
    print("Label\tTP\tFP\tFN\tP\tR\tF")
    for label in labels:
        recall[label] = 0.0 if (tp[label]+fn[label]) == 0.0 else (tp[label])/(tp
[label]+fn[label])
        precision[label] = 1.0 if (tp[label]+fp[label]) == 0.0 else (tp[label])/
(tp[label]+fp[label])
```

```
        f[label] = 0.0 if (precision[label]+recall[label])==0 else (2*precision[

label]*recall[label])/(precision[label]+recall[label])
        print(label[:7]+
            "\t"+str(int(tp[label]))+
            "\t"+str(int(fp[label]))+
            "\t"+str(int(fn[label]))+
            "\t"+str(round(precision[label],3))+
            "\t"+str(round(recall[label],3))+
            "\t"+str(round(f[label],3))
            )
        # micro average
        microrecall = (sum(tp.values()))/(sum(tp.values())+sum(fn.values()))
        microprecision = (sum(tp.values()))/(sum(tp.values())+sum(fp.values()))
        microf = 0.0 if (microprecision+microrecall)==0 else (2*microprecision*m
icrorecall)/(microprecision+microrecall)
    # Micro average
    print("MicAvg"+
        "\t"+str(int(sum(tp.values())))+
        "\t"+str(int(sum(fp.values())))+
        "\t"+str(int(sum(fn.values())))+
        "\t"+str(round(microprecision,3))+
        "\t"+str(round(microrecall,3))+
        "\t"+str(round(microf,3))
        )
    # Macro average
    macrorecall = sum(recall.values())/len(recall)
    macroprecision = sum(precision.values())/len(precision)
    macroF = sum(f.values())/len(f)
    print("MacAvg"+
        "\t"+str( )+
        "\t"+str( )+
        "\t"+str( )+
        "\t"+str(round(macroprecision,3))+
        "\t"+str(round(macrorecall,3))+
        "\t"+str(round(macroF,3))
        )
    print("Official result:",macroF)

    test_result.append(macroF)

if (len(actual_result) != len(dev_emotions)):
    eprint("Number of labels is not aligned!")
    sys.exit(1)
calculatePRF(dev_emotions,actual_result)

percent_matrix = [[0 for col in range(6)] for row in range(6)]
for i in range(len(dev_emotions)):
    percent_matrix[emotions.index(dev_emotions[i])][emotions.index(actual_result
[i])] += 1

dim_matrix = len(percent_matrix)
col_sum = [0] * len(emotions)
for i in range(dim_matrix):
```

```
    for j in range(dim_matrix):

        col_sum[i] = col_sum[i] + percent_matrix[i][j]

for i in range(len(percent_matrix)):
    for j in range(len(percent_matrix)):
        percent_matrix[i][j] = percent_matrix[i][j] / col_sum[i]

percent_matrix = [[0 for col in range(6)] for row in range(6)]
for i in range(len(dev_emotions)):
    percent_matrix[emotions.index(dev_emotions[i])][emotions.index(actual_result
[i])] += 1

tmp_matrix=pd.DataFrame(percent_matrix, columns = emotions, index = emotions)
display(HTML(tmp_matrix.to_html()))
dim_matrix = len(percent_matrix)
col_sum = [0] * len(emotions)
for i in range(dim_matrix):
    for j in range(dim_matrix):
        col_sum[i] = col_sum[i] + percent_matrix[i][j]

for i in range(len(percent_matrix)):
    for j in range(len(percent_matrix)):
        percent_matrix[i][j] = percent_matrix[i][j] / col_sum[i]

percent_matrix=pd.DataFrame(percent_matrix, columns = emotions, index = emotions
)
# display(HTML(percent_matrix.to_html()))

# plot heatmap
ax = sns.heatmap(percent_matrix.T, annot=True, linewidths=.5, cmap="Blues")

# turn the axis label
for item in ax.get_xticklabels():
    item.set_rotation(0)

for item in ax.get_xticklabels():
    item.set_rotation(90)
plt.ylabel("Predicted label")
plt.xlabel("True label")
# save figure
plt.savefig('resultHeatmap.png', dpi=100)
plt.show()
```

```
Labels: sad;surprise;anger;disgust;joy;fear
Label    TP      FP      FN      P       R       F
sad      397     280     1063    0.586   0.272   0.372
surpris  697     799     903     0.466   0.436   0.45
anger    691     1077    909     0.391   0.432   0.41
disgust  908     1198    689     0.431   0.569   0.49
joy      983     622     753     0.612   0.566   0.588
fear     891     1048    707     0.46    0.558   0.504
MicAvg   4567    5024    5024    0.476   0.476   0.476
MacAvg                           0.491   0.472   0.469
Official result: 0.4691360641845818
```

	anger	disgust	fear	joy	sad	surprise
anger	691	254	231	155	59	210
disgust	205	908	147	78	76	183
fear	236	176	891	91	39	165
joy	174	147	258	983	54	120
sad	248	325	184	185	397	121
surprise	214	296	228	113	52	697