01.112 Machine learning, Fall 2017

Design project report

Group Member:

Wu Zheyu 1001780

Zhou Xuexuan 1001603

Zuo Yunfei 1001441

Introduction

This report will briefly explain how we implement the code part by part as well as the result for each part. We have attached the code with the zip file. And we have also put the code up to Github. The url is:

https://github.com/Joe627487136/ML_Project

We are going to start with part 2 of the project since the part 1 wasn't technical but just generation of the annotated data.

Part 2

1. For the first question: to estimate the emission parameters from the training set using MLE (maximum likelihood estimation):

$$e(x|y) = \frac{\text{Count}(y \to x)}{\text{Count}(y)}$$

   We wrote a function called gen_*emission(file_path)* to create an emission dictionary called emi_dict: the keys are all the words from the training data (x in the above formula), and the values (an array) are all e(x|y) values with respect of every different y (the labels). Thus, each key-value pair is an emission parameter for a word.

   The format is:
   word: [0, B_Negative, I_Negative, B_Possitive, I_Possitive, B_Neutral, I_Neutral]

   For example: emi_dict= {Singapore: [0, 3/99, 1/78,2/76,5/67,9/45,2/95], …….} means that for the word "Singapore", the emission parameter from *0, B_Negative, I_Negative, B_Possitive, I_Possitive, B_Neutral, I_Neutral* to the word "Singapore" are 0, 3/99, 1/78,2/76,5/67,9/45,2/95.

2. To deal with the words which show up less than three times, the function is called modify_input_set (k, file_path, dict) in part2.py. It will replace these rarely-appearing words with #UNK# and generate another dev file for us to do the training (get the emission parameter).

3. To perform the simple sentiment analysis and generate the label. This is simply to label the word with the label corresponding to the highest emission parameter to the word.

Part 2 results:

1. French:
   #Entity in gold data: 223
   #Entity in prediction: 1149

   #Correct Entity: 182
   Entity precision: 0.1584
   Entity recall: 0.8161
   Entity F: 0.2653

   #Correct Sentiment: 68
   Sentiment precision: 0.0592
   Sentiment recall: 0.3049
   Sentiment F: 0.0991
2. English:
   #Entity in gold data: 226
   #Entity in prediction: 1201

#Correct Entity: 165
Entity precision: 0.1374
Entity recall: 0.7301
Entity F: 0.2313

#Correct Sentiment: 71
Sentiment precision: 0.0591
Sentiment recall: 0.3142
Sentiment F: 0.0995

3. Chinese:
   #Entity in gold data: 362
   #Entity in prediction: 3318

   #Correct Entity: 183
   Entity precision: 0.0552
   Entity recall: 0.5055
   Entity F: 0.0995

   #Correct Sentiment: 57
   Sentiment precision: 0.0172
   Sentiment recall: 0.1575
   Sentiment F: 0.0310
4. Singlish:
   #Entity in gold data: 1382
   #Entity in prediction: 6599

   #Correct Entity: 794
   Entity precision: 0.1203
   Entity recall: 0.5745
   Entity F: 0.1990

   #Correct Sentiment: 315
   Sentiment precision: 0.0477
   Sentiment recall: 0.2279
   Sentiment F: 0.0789

We can see that the overall F score is pretty low mainly because of the low precision, for both entity and sentiment, the recall is about 10 times larger that the precision.

Part 3

1. To estimate the transition parameters, we just calculated every possibility of going from one state to another one.
   We chose to use dictionary to represent this parameter. The dictionary's key is a tuple (previous-state, current-state), and the value would be the times of such transition happened in the files divided by the times of all the transitions started with this previous-state.
   The function in part3 is called gen_transition(file_path).

   In the later time of our work, we have refined the dictionary into a two-layer dictionary for it is more user-friendly, but the idea is the same, just changed the way of referring keys. The respective functions are refined_emi_dict(o_em_d,labels) and refined_trans_dict(tr_d,klabels).

2. To use the Viterbi algorithm on the dev set. We first parse the data into complete sentences (instead of lines of words), then feed the sentences to a function we wrote called viterbi(trans_dict,emi_dict,obs,labels).
   Assuming N is the length of this sentence and T is the number of labels. This function would firstly compute two N by T matrices filled with zero called V and prev, V is for storing the current max possibility for each step, and the prev is to store the previous state which lead to the maximum of the probability. Then it will fill these two matrices with the max and argmax for each step accordingly.

```
for i in range(N):
    if i == 0:
        for j in range(T):
            if obs_data[i] in emi_dict[labels[j]]:
                V[j][i] = trans_dict['START'][labels[j]]*emi_dict[labels[j]][obs_data[i]]
            elif obs_data[i] not in emi_dict[labels[j]]:
                V[j][i] = trans_dict['START'][labels[j]]*emi_dict[labels[j]]['#UNK#']
    else:
        for j in range(T):
            score = []
            for k in range(T):
                score.append(np.multiply(trans_dict[labels[k]][labels[j]], V[k][i-1]))
            if obs_data[i] in emi_dict[labels[j]]:
                score = np.multiply(score, emi_dict[labels[j]][obs_data[i]])
            elif obs_data[i] not in emi_dict[labels[j]]:
                score = np.multiply(score, emi_dict[labels[j]]['#UNK#'])
            V[j][i] = max(score)
            prev[j][i]=np.argmax(score)
```

   The "i == 0" part is the base state while the "else" part is the recursive step(s) except for the last step. The last step is as follow:

```
judge_case=[]
reverse_out_index_list=[]
for j in range(T):
    judge_case.append(np.multiply(V[j][-1],trans_dict[labels[j]]['STOP']))
fi_last = int(np.argmax(judge_case))
reverse_out_index_list.append(fi_last)
```

The fi_last is the index for the last label, we add it to the reverse_out_index_list and we will use prev matrix to trace back from the last word to start to get the label for each word, and the function will return index of the label in this sentence as an output.

Part 3 results:

1. French:
   #Entity in gold data: 223
   #Entity in prediction: 166

   #Correct Entity: 112
   Entity precision: 0.6747
   Entity recall: 0.5022
   Entity F: 0.5758

   #Correct Sentiment: 72
   Sentiment precision: 0.4337
   Sentiment recall: 0.3229
   Sentiment F: 0.3702
2. English:
   #Entity in gold data: 226
   #Entity in prediction: 162

   #Correct Entity: 104
   Entity precision: 0.6420
   Entity recall: 0.4602
   Entity F: 0.5361

   #Correct Sentiment: 64
   Sentiment precision: 0.3951
   Sentiment recall: 0.2832
   Sentiment F: 0.3299
3. Chinese:
   #Entity in gold data: 362
   #Entity in prediction: 158

   #Correct Entity: 64
   Entity precision: 0.4051
   Entity recall: 0.1768
   Entity F: 0.2462

#Correct Sentiment: 47
Sentiment precision: 0.2975
Sentiment recall: 0.1298
Sentiment F: 0.1808

4. Singlish:
   #Entity in gold data: 1382
   #Entity in prediction: 723

   #Correct Entity: 386
   Entity precision: 0.5339
   Entity recall: 0.2793
   Entity F: 0.3667

   #Correct Sentiment: 244
   Sentiment precision: 0.3375
   Sentiment recall: 0.1766
   Sentiment F: 0.2318

It is expected but also excited to see that the overall F score is much higher than only doing sentiment analysis. The function tends to predict much less, or more “cautious”.

Part 4:

1. To get the forward probability for each state, we have:

```
#ini alpha
alpha = np.zeros((lb_len, obs_len))

for j in range (obs_len):
    if j==0:
        for u in range(lb_len):
            label_u = labels[u]
            alpha[u][j]=trans_dict['START'][label_u]
    else:
        for u in range(lb_len):
            label_u=labels[u]
            for v in range(lb_len):
                label_v=labels[v]
                if obs_data[j-1] in emi_dict[label_v].keys():
                    alpha[u][j] += alpha[v][j-1]*trans_dict[label_v][label_u]*emi_dict[label_v][obs_data[j-1]]
                else:
                    alpha[u][j] += alpha[v][j-1]*trans_dict[label_v][label_u]*emi_dict[label_v]['#UNK#']
```

J == 0 is the base case and the else case is the recursive case. The matrix alpha is to store the probability for each state.

2. Similarly, we can calculate the backwards probability for each state:

```
#ini beta
beta = np.zeros((lb_len, obs_len))

for j in range (obs_len-1,-1,-1):
    if j==obs_len-1:
        for u in range(lb_len):
            label_u = labels[u]
            if obs_data[j] in emi_dict[label_u].keys():
                beta[u][j]=trans_dict[label_u]['STOP']*emi_dict[label_u][obs_data[j]]
            else:
                beta[u][j]=trans_dict[label_u]['STOP']*emi_dict[label_u]['#UNK#']
    else:
        for u in range(lb_len):
            label_u=labels[u]
            for v in range(lb_len):
                label_v=labels[v]
                if obs_data[j] in emi_dict[label_u].keys():
                    beta[u][j] += beta[v][j+1]*trans_dict[label_u][label_v]*emi_dict[label_u][obs_data[j]]
                else:
                    beta[u][j] += beta[v][j+1]*trans_dict[label_u][label_v]*emi_dict[label_u]['#UNK#']
```

The “j==obs_len-1” is the base case and the else condition is the recursive case.

3. To multiply them up and get the probability of the paths from start to stop going through that specific label for each word and we will take the label with highest probability for each word:

```
rs_path = []

for j in range(obs_len):
    score = []
    for u in range(len(labels)):
        score.append(alpha[u][j]*beta[u][j])
    slcted_index = np.argmax(score)
    rs_path.append(labels[slcted_index])
```

And the rs_path is output which contains the label for every word for the input sentence.

Part 4 results:

1. For English the results are:
   #Entity in gold data: 226
   #Entity in prediction: 175

   #Correct Entity: 108
   Entity precision: 0.6171
   Entity recall: 0.4779
   Entity F: 0.5387

   #Correct Sentiment: 69
   Sentiment precision: 0.3943
   Sentiment recall: 0.3053
   Sentiment F: 0.3441

2. For French the results are:
   #Entity in gold data: 223
   #Entity in prediction: 173

   #Correct Entity: 113
   Entity precision: 0.6532
   Entity recall: 0.5067
   Entity F: 0.5707

   #Correct Sentiment: 73
   Sentiment precision: 0.4220
   Sentiment recall: 0.3274
   Sentiment F: 0.3687

Part 5:

1. To come out with a better way for labeling we consider a method that combine both Viterbi and Forward-Backward algorithm.
   a. After analyzing the labeling result, we found that Viterbi sometimes are too over-fitting on some labels due to its strict rule over transition.

      Eg: from left to right over word beer (Gold set, Forward-Backward, Viterbi)

We O	We O	We O
have O	have O	have O
since O	since O	since O
returned O	returned O	returned O
and O	and O	and O
also O	also O	also O
had O	had O	had O
a O	a O	a O
great O	great O	great O
experience O	experience O	experience O
, O	, O	, O
sampling O	sampling O	sampling O
more O	more O	more O
small O	small O	small O
plates O	plates O	plates O
and O	and O	and O
a O	a O	a O
variety O	variety O	variety O
of O	of O	of O
the O	the O	the O
beer B-positive	beer I-positive	beer O
(O	(O	(O
cold O	cold O	cold O
and O	and O	and O
good O	good O	good O
) O	) O	) O
. O	. O	. O

   b. Due to this reason, we would like to determine if at some point, Viterbi cannot create an entity but actually forward-backward do have a sentiment over the word. Then we decide to create this label depends on the forward-backward sentiment result but modify the label with the entity format.

      As showing above example, the best way to modify is to pick Part 4 Forward-Backward "beer" with its positive sentiment and correct its entity format to "B-positive" which will can result an exact match with gold output set.

2. Implementation:

```
final_labels = viterbi_label # Use viterbi as mother board
for index in range (N):
    c_vb_lb = viterbi_label[index] # Current viterbi label
    c_fb_lb = fb_label[index] # Current forward-backward label
    if c_vb_lb == 'O' and c_fb_lb != 'O': # Check if current viterbi is O and fb is some other
        # If so
        fb_likely = False  # ini flag
        # Loop all previous index for viterbi
        for p_index in range(index):
            # Check if all previous viterbi output are all 'O'
            if viterbi_label[p_index] != 'O':
                # If got some non-'O' entity which indicate viterbi would probably working, set flag to false, break
                fb_likely = 'False'
                break
            # If all viterbi out are 'O' then set flag to true
            fb_likely = True
        if fb_likely == True:
            # Start to adapt fb output if fb flag set to true
            if c_fb_lb[0] == 'I':
                # Correct fb entity since FB are more sentiment driven (since all 'O' before then should be 'B' here)
                c_fb_lb = 'B-'+c_fb_lb[2:]
            # Write to final output
            final_labels[index] = c_fb_lb
```

We use Viterbi labels for a sentence as mother board and insert modified Forward-Backward sentiment as if the Viterbi giving all 'O' output (which is most likely over-fitting)

3. Similar to previous output method we can output our result.

Part 5 results and comparisons:

1. For English the results are:
#Entity in gold data: 226
#Entity in prediction: 172

#Correct Entity: 110
Entity precision: 0.6395
Entity recall: 0.4867
Entity F: 0.5528

#Correct Sentiment: 69
Sentiment precision: 0.4012
Sentiment recall: 0.3053
Sentiment F: 0.3467

1st ----Mix-method: (Entity F: 0.5528, Sentiment F: 0.3467)
2nd ---Forward-Backward: (Entity F: 0.5387, Sentiment F: 0.3441)
3rd ----Viterbi: (Entity F: 0.5361, Sentiment F: 0.3299)

2. For FR the results are:
   #Entity in gold data: 223
   #Entity in prediction: 169

   #Correct Entity: 112
   Entity precision: 0.6627
   Entity recall: 0.5022
   Entity F: 0.5714

   #Correct Sentiment: 72
   Sentiment precision: 0.4260
   Sentiment recall: 0.3229
   Sentiment F: 0.3673

   1st ---- Viterbi: (Entity F: 0.5758, Sentiment F: 0.3702)
   2nd --- Mix-method: (Entity F: 0.5714, Sentiment F: 0.3673)
   3rd ----Forward-Backward: (Entity F: 0.5705, Sentiment F: 0.3687)

3. For CN set:
   1st ----Mix-method: (Entity F: 0.2500, Sentiment F: 0.1818)
   2nd ---Viterbi: (Entity F: 0.2462, Sentiment F: 0.1808)
   3rd ----Forward-Backward: (Entity F: 0.2432, Sentiment F: 0.1706)

4. For SG set:
   1st ----Mix-method: (Entity F: 0.3673, Sentiment F: 0.2330)
   2nd ---Viterbi: (Entity F: 0.3667, Sentiment F: 0.2318)
   3rd ---Forward-Backward: (Entity F: 0.3639, Sentiment F: 0.2383)

Part 5 conclusions:

Due to FR set has a relative bad response from Forward-Backward method so the mix-method doesn't work well. But generally, a language set like EN set which has a relatively precise Forward-backward will has a better result if we rationally use the mix-method.