Artificial Intelligence In Medical Applications and Services
Homework3 Report

1. Score Result

f1score
0.4258144301165879

2. Process

As the TA mentioned, we can seperate the whole project into 2 parts which are **Preprocessing** and **Model_Training.**

1. Preprocessing

First we use **loadInputFile** function to load the data given by TA (SampleData_deid.txt).

```python
def loadInputFile(path):
    trainingset = list()  # store trainingset [content,content,...]
    position = list()  # store position [article_id, start_pos, end
    mentions = dict()  # store mentions[mention] = Type
    with open(file_path, 'r', encoding='utf8') as f:
        file_text=f.read().encode('utf-8').decode('utf-8-sig')
    datas=file_text.split('\n\n--------------------\n\n')[:-1]
    for data in datas:
        data=data.split('\n')
        content=data[0]
        trainingset.append(content)
        annotations=data[1:]
        for annot in annotations[1:]:
            annot=annot.split('\t') #annot= article_id, start_pos,
            position.extend(annot)
            mentions[annot[3]]=annot[4]

    return trainingset, position, mentions
```

Then use CRFFormatData to change the input data into a corresponding format (clear the space, adding 'B' 'O' 'I' label)

```python
def CRFFormatData(trainingset, position, path):
    if (os.path.isfile(path)):
        os.remove(path)
    outputfile = open(path, 'a', encoding= 'utf-8')

    # output file lines
    count = 0 # annotation counts in each content
    tagged = list()
    for article_id in range(len(trainingset)):
        trainingset_split = list(trainingset[article_id])
        while '' or ' ' in trainingset_split:
            if '' in trainingset_split:
                trainingset_split.remove('')
            else:
                trainingset_split.remove(' ')
        start_tmp = 0
        for position_idx in range(0,len(position),5):
            if int(position[position_idx]) == article_id:
                count += 1
                if count == 1:
                    start_pos = int(position[position_idx+1])
                    end_pos = int(position[position_idx+2])
                    entity_type=position[position_idx+4]
                    if start_pos == 0:
                        token = list(trainingset[article_id][start_pos:end_pos])
                        whole_token = trainingset[article_id][start_pos:end_pos]
                        for token_idx in range(len(token)):
                            if len(token[token_idx].replace(' ','')) == 0:
                                continue
                            # BIO states
                            if token_idx == 0:
                                label = 'B-'+entity_type
                            else:
                                label = 'I-'+entity_type
```

2. Model_Training

Create a model to fit and predict, and don’t forget to flatten to get final f1score and print the prediction result

```python
def CRF(x_train, y_train, x_test, y_test):
    crf = sklearn_crfsuite.CRF(
        algorithm='lbfgs',
        c1=0.1,
        c2=0.1,
        max_iterations=100,
        all_possible_transitions=True
    )
    crf.fit(x_train, y_train)
    # print(crf)
    y_pred = crf.predict(x_test)
    y_pred_mar = crf.predict_marginals(x_test)

    # print(y_pred_mar)

    labels = list(crf.classes_)
    labels.remove('O')
    f1score = metrics.flat_f1_score(y_test, y_pred, average='weighted', labels=labels)
    sorted_labels = sorted(labels,key=lambda name: (name[1:], name[0])) # group B and I results
    print(flat_classification_report(y_test, y_pred, labels=sorted_labels, digits=3))
    return y_pred, y_pred_mar, f1score
```

This part is very important. We need to prepare a pretrained word vector file to let it find corresponding key value of the token that get from input data.

The first two tokens are the parameters which are vocabulary_size and it's dimension. And keep the rest tokens into **vec** array in float type, and return word_vecs dictionary.

```python
# load pretrained word vectors
# get a dict of tokens (key) and their pretrained word vectors (value)
# pretrained word2vec CBOW word vector: https://fgc.stpi.narl.org.tw/activity/videoDe
dim = 0
word_vecs= {}
# open pretrained word vector file
with open('/content/drive/MyDrive/Colab Notebooks/NER/glove.42B.300d.txt') as f:
    for line in f:
        tokens = line.strip().split()

        # there 2 integers in the first line: vocabulary_size, word_vector_dim
        if len(tokens) == 2:
            dim = int(tokens[1])
            continue

        word = tokens[0]
        vec = np.array([ float(t) for t in tokens[1:] ])
        word_vecs[word] = vec

```

Prepare dataset and split them into training_set and testing_set with test_size = 0.33

```python
def Dataset(data_path):
    with open(data_path, 'r', encoding='utf-8') as f:
        data=f.readlines()#.encode('utf-8').decode('utf-8-sig')
    data_list, data_list_tmp = list(), list()
    article_id_list=list()
    idx=0
    for row in data:
        data_tuple = tuple()
        if row == '\n':
            article_id_list.append(idx)
            idx+=1
            data_list.append(data_list_tmp)
            data_list_tmp = []
        else:
            row = row.strip('\n').split(' ')
            data_tuple = (row[0], row[1])
            data_list_tmp.append(data_tuple)
    if len(data_list_tmp) != 0:
        data_list.append(data_list_tmp)

    # here we random split data into training dataset and testing dataset
    # but you should take `development data` or `test data` as testing data
    # At that time, you could just delete this line,
    # and generate data_list of `train data` and data_list of `development/test data` by this function
    traindata_list, testdata_list, traindata_article_id_list, testdata_article_id_list=train_test_split(data_list,
                                                                                                          article_id_list,
                                                                                                          test_size=0.33,
                                                                                                          random_state=42)

    return data_list, traindata_list, testdata_list, traindata_article_id_list, testdata_article_id_list
```

Use the pretrained word vector that we just imported as embedding_dict to find the input data_list's corresponding value with three for loops and keep them as a list.

```python
def Word2Vector(data_list, embedding_dict):
    embedding_list = list()

    # No Match Word (unknown word) Vector in Embedding
    unk_vector=np.random.rand(*(list(embedding_dict.values())[0].shape))

    for idx_list in range(len(data_list)):
        embedding_list_tmp = list()
        for idx_tuple in range(len(data_list[idx_list])):
            key = data_list[idx_list][idx_tuple][0] # token
            # print(str(idx_tuple) + key)
            if key in embedding_dict:
                value = embedding_dict[key]
            else:
                value = unk_vector
            embedding_list_tmp.append(value)
        embedding_list.append(embedding_list_tmp)
    return embedding_list
```

Extract all the features of embedding_list and keep also(three loops go through 3 dimension of embed_list)

```python
# input features: pretrained word vectors of each token
# return a list of feature dicts, each feature dict corresponding to each token
def Feature(embed_list):
    feature_list = list()
    for idx_list in range(len(embed_list)):
        feature_list_tmp = list()
        for idx_tuple in range(len(embed_list[idx_list])):
            feature_dict = dict()
            for idx_vec in range(len(embed_list[idx_list][idx_tuple])):
                feature_dict['dim_' + str(idx_vec+1)] = embed_list[idx_list][idx_tuple][idx_vec]
            feature_list_tmp.append(feature_dict)

        feature_list.append(feature_list_tmp)
    return feature_list
```

This is how embed_list[idx_list][idx_tuple][idx_vec] looks like:

```
0.6180830816584256
0.23310316349906202
0.7140401709909245
0.19546616359681368
0.8567940396548351
0.7400688220122569
```

Prepare a Label list by set the third dimension of data_list to **1**

```python
# get the labels of each tokens in train.data
# return a list of lists of labels
def Preprocess(data_list):
    label_list = list()
    for idx_list in range(len(data_list)):
        label_list_tmp = list()
        for idx_tuple in range(len(data_list[idx_list])):
            label_list_tmp.append(data_list[idx_list][idx_tuple][1])
        label_list.append(label_list_tmp)
    return label_list
```

Call the functions

```
# Load Word Embedding
trainembed_list = Word2Vector(traindata_list, word_vecs)
testembed_list = Word2Vector(testdata_list, word_vecs)

# CRF - Train Data (Augmentation Data)
x_train = Feature(trainembed_list)
y_train = Preprocess(traindata_list)

# CRF - Test Data (Golden Standard)
x_test = Feature(testembed_list)
y_test = Preprocess(testdata_list)
```

Fit and predict

```
y_pred, y_pred_mar, f1score = CRF(x_train, y_train, x_test, y_test)
```

Result

```
              precision    recall  f1-score   support

  B-location      0.000     0.000     0.000        15
  I-location      0.000     0.000     0.000        41
  B-med_exam      0.200     0.030     0.053        33
  I-med_exam      1.000     0.075     0.140        80
     B-money      0.364     0.333     0.348        12
     I-money      0.353     0.171     0.231        35
      B-name      0.500     0.143     0.222         7
      I-name      0.333     0.100     0.154        10
      B-time      0.667     0.450     0.538       111
      I-time      0.825     0.532     0.647       265

   micro avg      0.714     0.345     0.465       609
   macro avg      0.424     0.184     0.233       609
weighted avg      0.661     0.345     0.426       609
```

F1Score

```
f1score
```

```
0.4258144301165879
```

Change the output format and save into output.tsv file
Here we show the entity text’s id, start_position, ending_position, content and type.

```python
output="article_id\tstart_position\tend_position\tentity_text\tentity_type\n"
for test_id in range(len(y_pred)):
    pos=0
    start_pos=None
    end_pos=None
    entity_text=None
    entity_type=None
    for pred_id in range(len(y_pred[test_id])):
        if y_pred[test_id][pred_id][0]=='B':
            start_pos=pos
            entity_type=y_pred[test_id][pred_id][2:]
        elif start_pos is not None and y_pred[test_id][pred_id][0]=='I' and y_pred[test_id][pred_id+1][0]=='O':
            end_pos=pos
            entity_text=''.join([testdata_list[test_id][position][0] for position in range(start_pos,end_pos+1)])
            line=str(testdata_article_id_list[test_id])+'\t\t'+str(start_pos)+'\t\t'+str(end_pos+1)+'\t\t'+entity_text+'\t\t'+entity_type
            output+=line+'\n'
        pos+=1
```

article_id	start_position	end_position	entity_text	entity_type
8	52	54	前天	time
8	68	70	昨天	time
8	189	193	二十分鐘	time
8	293	295	五年	time
8	540	544	兩個禮拜	time
8	726	728	前天	time
8	730	732	前天	time
8	858	860	前天	time
8	898	900	前天	time
8	1549	1551	五天	time
8	1622	1626	五天禮拜	time
8	2352	2354	去喬	time
8	2560	2563	兩個月	time
16	51	55	九、十點	time
16	60	64	九、十點	time
16	122	124	三年	time
16	130	132	三年	time
16	247	249	三年	time
0	1268	1271	8公分	med_exam
0	1358	1362	三多路上	time
0	2576	2578	五天	time

3. Features

這裡原本我是只使用助教給的抓取data_list的第一個([0])也就是token內容去比對embedding_dict的key value 作為features

```python
for idx_tuple in range(len(data_list[idx_list])):
    key = data_list[idx_list][idx_tuple][0] # token
    # print(str(idx_tuple) + key)
    if key in embedding_dict:
        value = embedding_dict[key]
    else:
        value = unk_vector
```

```python
for idx_list in range(len(embed_list)):
    feature_list_tmp = list()
    for idx_tuple in range(len(embed_list[idx_list])):
        feature_dict = dict()
        for idx_vec in range(len(embed_list[idx_list][idx_tuple])):
            feature_dict['dim_' + str(idx_vec+1)] = embed_list[idx_list][idx_tuple][idx_vec]
```

This is how embed_list[idx_list][idx_tuple][idx_vec] looks like:

```
0.6180830816584256
0.2331031634990620 2
0.7140401709909245
0.1954661635968136 8
0.8567940396548351
0.7400688220122569
```

1. 後來有試着先做了加入word_length的features，平均f1score有比較微微的上升(比原本的0.35~0.41)，固定維持在0.4以上
2. 再來是加入word_position(start_pos)，我這裡是選擇給start_pos，因為給end_pos的效果其實沒什麼影響，給了start_pos後只加了一點點
3. 再來是對於features的組織是要通過word2vector和features這兩個function後才能作成x_train, x_test的，我發現其實我的word2vector有成功把我想要的features加入到embedding_dict中並且return吐出，但是由於時間的關係我在features的dict中沒有成功整合，所以model其實沒有吃到我加入的features。不過就上述情況而言，我已經成功了解如果要訓練一個NER-CRF的模型的大概流程是怎麼樣的了。

```
else:
    # print('Row:' + row)
    row = row.strip('\n').split(' ')
    data_tuple = (row[0], row[1], row[2], row[3])
    # print(data_tuple)
```

```
for idx_list in range(len(data_list)):
    embedding_list_tmp = list()
    embed_dict = dict()
    for idx_tuple in range(len(data_list[idx_list])):
        key = data_list[idx_list][idx_tuple][0] # token
        # print(str(idx_tuple) + key)
        if key in embedding_dict:
            value = embedding_dict[key]
        else:
            value = unk_vector
        embed_dict[0] = value # token in embedding_dict's key value
        embed_dict[1] = data_list[idx_list][idx_tuple][2] # word_position(start_pos)
        embed_dict[2] = data_list[idx_list][idx_tuple][3] # word_length
        # embedding_list_tmp.append(value)
        embedding_list_tmp.append(embed_dict)
    embedding_list.append(embedding_list_tmp)
return embedding_list
```

```
def Feature(embed_list):
    feature_list = list()
    for idx_list in range(len(embed_list)):
        feature_list_tmp = list()
        for idx_tuple in range(len(embed_list[idx_list])):
            feature_dict = dict()
            # print(embed_list[idx_list][idx_tuple][0])
            for idx_vec in range(len(embed_list[idx_list][idx_tuple])):
                feature_dict['dim_' + str(idx_vec+1)] = embed_list[idx_list][idx_tuple][idx_vec]
                # feature_dict[str(0+1)] = embed_list[idx_list][0]
                # feature_dict[str(1+1)] = embed_list[idx_list][2]
                # feature_dict[str(2+1)] = embed_list[idx_list][3]
            # print(feature_dict)
            feature_list_tmp.append(feature_dict)

        feature_list.append(feature_list_tmp)
    return feature_list
```

4. Experience

- From this homework, I had learned what are **NER** and **CRF model**,and how computer learn a high-level human language.
- As we know, **f1score** show our model performance, but the point is how to get higher score! This is definitely different to the typical ML model, because it is not only the input_data but also how we processing the input_data into tokens and label is decisively important !
- There is no so much parameters to let you adjust to get better model performance, so a great processing_data and completely pretrained word vector file will help your model to become better!
- Interesting topic for us to explore how great ML is in this generation.
- 雖然由於時間的關係我在features的dict中沒有成功整合不過就上述情況而言，我已經成功了解如果要訓練一個NER-CRF的模型的大概流程是怎麼樣的了。
- Thanks for Prof and TA! :)