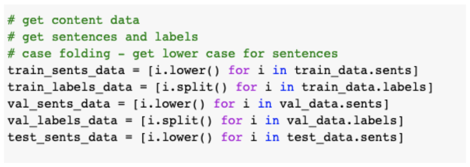
COMP5046-A2 Group41

Bingtao Zhao[1] and Yu Dian[2]

[1] University of Sydney, NSW 2006, Australia
bzha0935@sydney.edu.au
[2] University of Sydney, NSW 2006, Australia
diyu2097@uni.sydney.edu.au

1 Data preprocessing

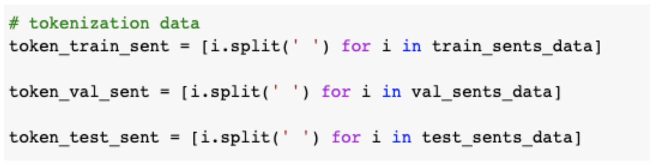
First, we have to download file through google drive and read three csv files train, val, and test through pandas. Second, we use code to lower word case. Case folding is performed on the sentence part of the file, and all uppercase letters are converted to lowercase, so that the sequence model can complete the task more accurately(1). On the other hand, case folding will also help us to unified data format. Code show in Fig.1.



```
# get content data
# get sentences and labels
# case folding - get lower case for sentences
train_sents_data = [i.lower() for i in train_data.sents]
train_labels_data = [i.split() for i in train_data.labels]
val_sents_data = [i.lower() for i in val_data.sents]
val_labels_data = [i.split() for i in val_data.labels]
test_sents_data = [i.lower() for i in test_data.sents]
```

Fig. 1. Case folding

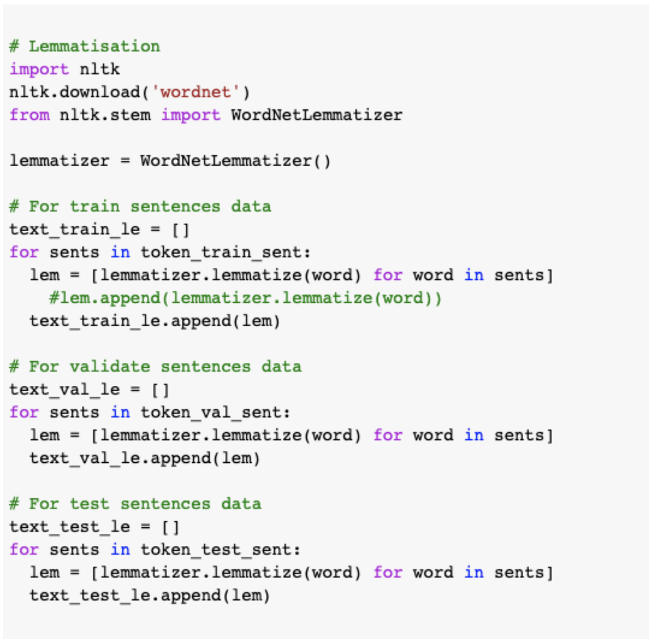
Then split the sentences and labels into lists through split(" "), perform tokennization processing, and make the individual tokens in the list correspond one-to-one. We can not use the whole sentence as input because the labels of each sentence are refer to each word. Code show in Fig.2.



```
# tokenization data
token_train_sent = [i.split(' ') for i in train_sents_data]

token_val_sent = [i.split(' ') for i in val_sents_data]

token_test_sent = [i.split(' ') for i in test_sents_data]
```

Fig. 2.

After tokennization, we use lemmatisation to process data further. The reason that we chosen lemmatisation not stemming is stemming will turn the word to the root word and it may cause some word can not be identify in the real word. However, lemmatisation is use dictionary to process word, like ”running” to ”run”(2). This way may better for understanding the meaning of word. Code show in Fig.3.



```
# Lemmatisation
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# For train sentences data
text_train_le = []
for sents in token_train_sent:
  lem = [lemmatizer.lemmatize(word) for word in sents]
    #lem.append(lemmatizer.lemmatize(word))
  text_train_le.append(lem)

# For validate sentences data
text_val_le = []
for sents in token_val_sent:
  lem = [lemmatizer.lemmatize(word) for word in sents]
  text_val_le.append(lem)

# For test sentences data
text_test_le = []
for sents in token_test_sent:
  lem = [lemmatizer.lemmatize(word) for word in sents]
  text_test_le.append(lem)
```

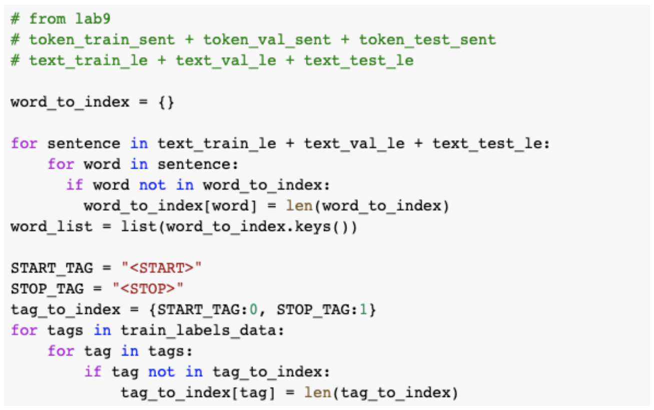
Fig. 3.

Then, turning each token into index and put start and stop tag in the labels index. Code show in Fig.4.

2 Input Embedding

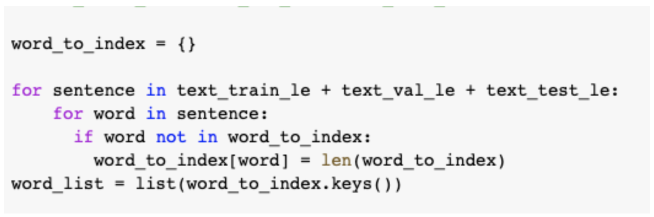
Refer to the code of lab9, combine the sent parts of the train, val, and test data sets and add them to the word_to_index dictionary. To get all the words in sents by calling list(word_to_index.keys()). Code show in Fig.5.

Control the start and end of CRF by assigning a value of 0 to ¡START¿tag and a value of 1 to ¡STOP¿tag. Code show in Fig.6.



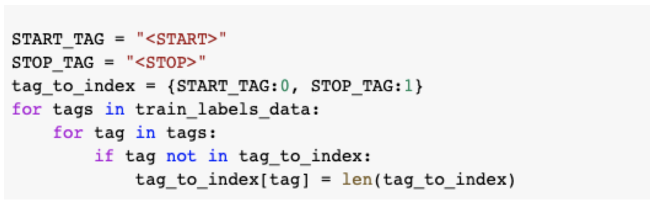
```
# from lab9
# token_train_sent + token_val_sent + token_test_sent
# text_train_le + text_val_le + text_test_le

word_to_index = {}

for sentence in text_train_le + text_val_le + text_test_le:
    for word in sentence:
      if word not in word_to_index:
        word_to_index[word] = len(word_to_index)
word_list = list(word_to_index.keys())

START_TAG = "<START>"
STOP_TAG = "<STOP>"
tag_to_index = {START_TAG:0, STOP_TAG:1}
for tags in train_labels_data:
    for tag in tags:
        if tag not in tag_to_index:
            tag_to_index[tag] = len(tag_to_index)
```

Fig. 4.



```
word_to_index = {}

for sentence in text_train_le + text_val_le + text_test_le:
    for word in sentence:
      if word not in word_to_index:
        word_to_index[word] = len(word_to_index)
word_list = list(word_to_index.keys())
```

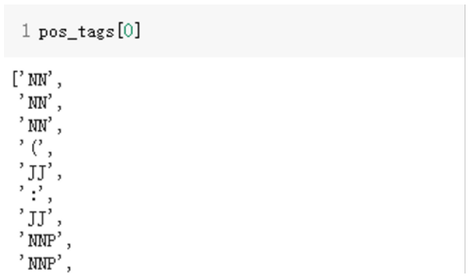
Fig. 5.



```
START_TAG = "<START>"
STOP_TAG = "<STOP>"
tag_to_index = {START_TAG:0, STOP_TAG:1}
for tags in train_labels_data:
    for tag in tags:
        if tag not in tag_to_index:
            tag_to_index[tag] = len(tag_to_index)
```

Fig. 6.

Import nltk package, through the ”averaged_perceptron_tagger” corpus, based on the definition and context of the word in sents, the word is marked as the corresponding specific Part of speech. Code show in Fig.7.



```
pos_tags[0]
```

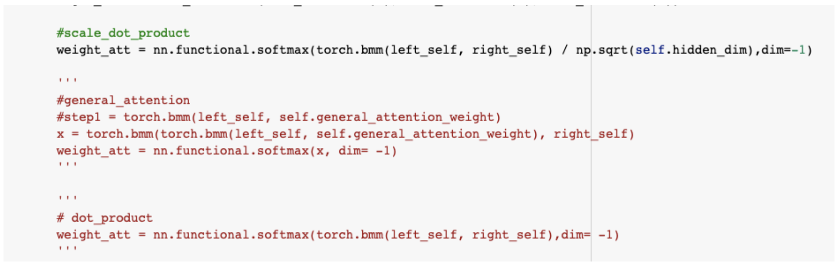
```
['NN',
 'NN',
 'NN',
 '(',
 'JJ',
 ':',
 'JJ',
 'NNP',
 'NNP',
```

Fig. 7.

After obtaining pos_tags, it is trained by calling the word2vec meth-od, transformed into the corresponding vector, and spliced to the word vector. Refer to the code of lab7 and download the ”en_core_web_sm” package through spacy to perform Dependency Parsing on data. Download word_emb_model (”glove-twitter-200”) through api to generate Embed-ding Matrix. Set the embedding dim to 261, and the embedding matrix corresponds to the dictionary, so that all strings can be converted into corresponding index forms. After a sentence is wordized, the word is converted into the corresponding index, and the index is used for training, so that the embedding layer in torch can be called. Because we try to use part of speech and dependency parsing at same time, so we have embedding two times.

3 NER model

For the NER model, we mainly use the BiLSTM_CRF model which from lab9, and we have modify some part to meet the requirement of Assignment 2. We have try three attention method, like scale dot product, general product and dot product. In the end, we try use scale dot product in this task.



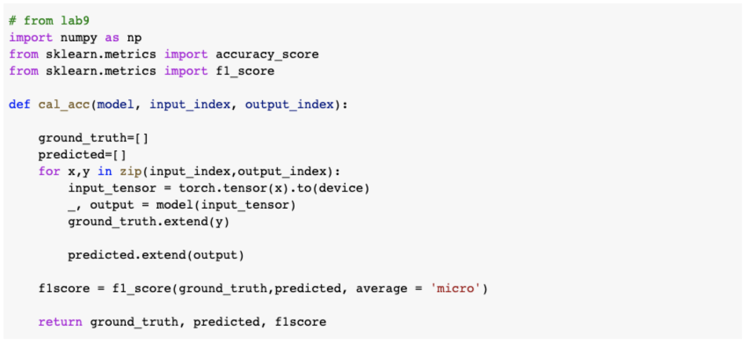
```
#scale_dot_product
weight_att = nn.functional.softmax(torch.bmm(left_self, right_self) / np.sqrt(self.hidden_dim),dim=-1)

'''
#general_attention
#step1 = torch.bmm(left_self, self.general_attention_weight)
x = torch.bmm(torch.bmm(left_self, self.general_attention_weight), right_self)
weight_att = nn.functional.softmax(x, dim= -1)
'''

'''
# dot_product
weight_att = nn.functional.softmax(torch.bmm(left_self, right_self),dim= -1)
'''
```

Fig. 8.

4 Evaluation

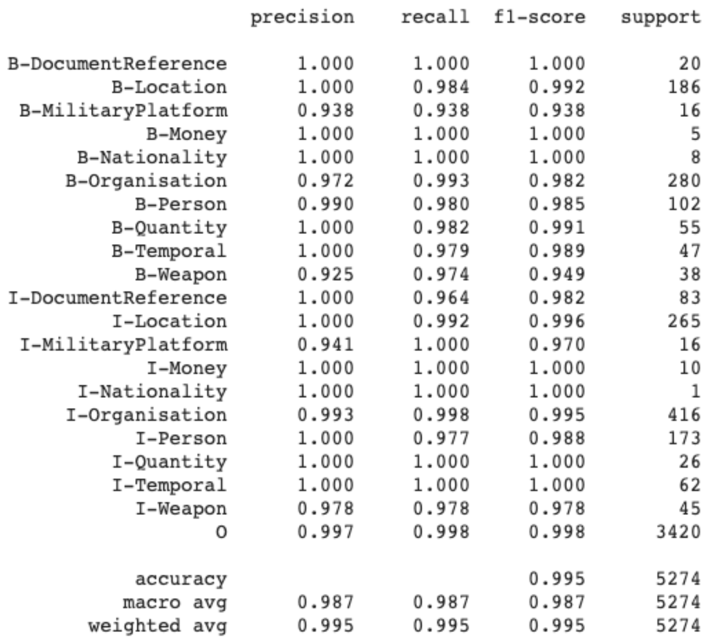
4.1 Evaluation setup

For the evaluation, we use valid data to evaluate our model performance. In the data processing, we have convert valid sentence and valid labels into index. Then, we use the cal_acc model which from Lab9 to calculate the F1 score and ground truth. After training model, using "classification_report" to get more details information.



```
# from lab9
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

def cal_acc(model, input_index, output_index):

    ground_truth=[]
    predicted=[]
    for x,y in zip(input_index,output_index):
        input_tensor = torch.tensor(x).to(device)
        _, output = model(input_tensor)
        ground_truth.extend(y)

        predicted.extend(output)

    f1score = f1_score(ground_truth,predicted, average = 'micro')

    return ground_truth, predicted, f1score
```

Fig. 9. The Calculate Accuracy Code Part



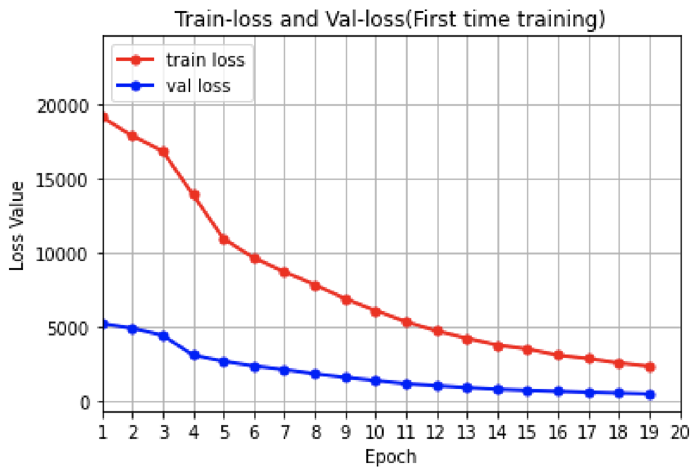
```
                     precision    recall  f1-score   support

B-DocumentReference      1.000     1.000     1.000        20
         B-Location      1.000     0.984     0.992       186
 B-MilitaryPlatform      0.938     0.938     0.938        16
            B-Money      1.000     1.000     1.000         5
      B-Nationality      1.000     1.000     1.000         8
     B-Organisation      0.972     0.993     0.982       280
           B-Person      0.990     0.980     0.985       102
         B-Quantity      1.000     0.982     0.991        55
         B-Temporal      1.000     0.979     0.989        47
           B-Weapon      0.925     0.974     0.949        38
I-DocumentReference      1.000     0.964     0.982        83
         I-Location      1.000     0.992     0.996       265
 I-MilitaryPlatform      0.941     1.000     0.970        16
            I-Money      1.000     1.000     1.000        10
      I-Nationality      1.000     1.000     1.000         1
     I-Organisation      0.993     0.998     0.995       416
           I-Person      1.000     0.977     0.988       173
         I-Quantity      1.000     1.000     1.000        26
         I-Temporal      1.000     1.000     1.000        62
           I-Weapon      0.978     0.978     0.978        45
                  O      0.997     0.998     0.998      3420

           accuracy                          0.995      5274
          macro avg      0.987     0.987     0.987      5274
       weighted avg      0.995     0.995     0.995      5274
```

Fig. 10. Classification Report

4.2 Evaluation result

For the predict result of test data, we have convert index to real tag after our training model. Because we have to turn labels which in train data set and val data set to vector and as input to train model with sentences. Second, we use our best performance model to predict test data. In this processing, test data will transfer to tensor vector and put into model to get result, then output will be convert to real labels by the list we processing in the first step. In the end, we use code to create output csv file.

To compare the performance between First Training and After Training, we have plot the line graph below.



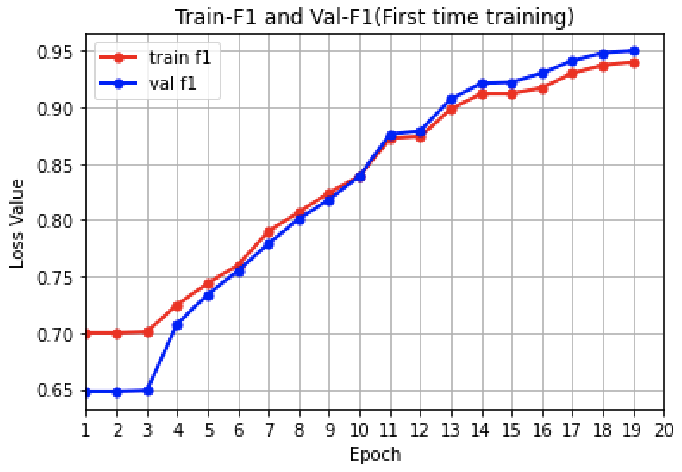


Fig. 11.

The train loss value and Val loss value have drop significant initially, then tends to stabilise. For the F1 score, both of them increase fast in the beginning stage and then grows slow.

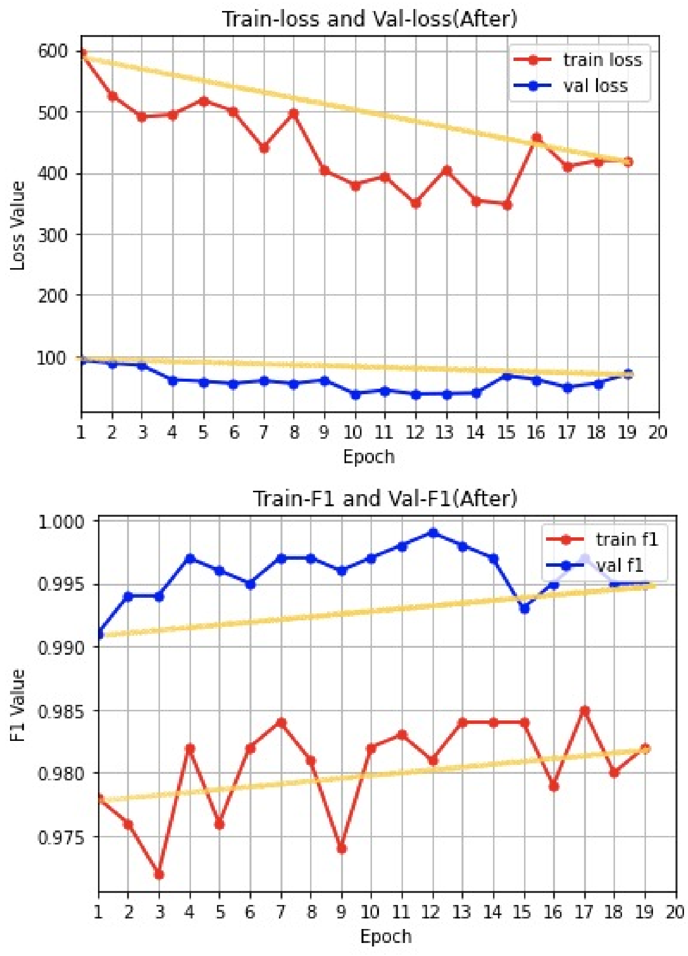


Fig. 12.

After we train the model several times, the loss value and F1 score are becoming floating in a range. Their trends are starting to fluctuate, but overall they are still trending up (F1) and down (Loss). This means that the model can continue to learn and train, but if the training epoch is increased, it may produce overfitting.

Bibliography

[1] C. Manning, P. Raghavan and H. Schütze, "Capitalization/case-folding.", Nlp.stanford.edu, 2008. [Online]. Available: `https://nlp.stanford.edu/IR-book/html/htmledition/capitalizationcase-folding-1.html`.[Accessed: 12- Jun- 2021].

[2] C. Manning, P. Raghavan and H. Schütze, "Stemming and lemmatization", Nlp.stanford.edu, 2008. [Online]. Available: `https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html`. [Accessed: 13- Jun- 2021].