

## task-2

March 28, 2023

```
[3]: from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

```
[4]: import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence,
    pad_packed_sequence
from torch.nn.modules import padding
from torch.optim.lr_scheduler import StepLR
```

### 0.1 Finding the distribution of TAGS

```
[5]: with open('train', "r") as f:
    d={}
    for line in f:
        line = line.strip()
        if len(line) != 0:
            parts = line.split(" ")
            label = parts[2]
            d[label]=1+d.get(label,0)
d
```

```
[5]: {'B-ORG': 6321,
      'O': 170524,
      'B-MISC': 3438,
      'B-PER': 6600,
      'I-PER': 4528,
      'B-LOC': 7140,
      'I-ORG': 3704,
      'I-MISC': 1155,
      'I-LOC': 1157}
```

## 0.2 Loading pre-trained Glove Embeddings

```
[6]: #https://www.kaggle.com/code/fyyccssx/first-try-lstm-with-glove-by-pytorch
embeddings_dictionary = dict()
glove_file = open('/content/drive/MyDrive/nlp/glove.6B.100d.txt',
                  encoding="utf8")
word2idx={}
for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = np.asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions
    if word not in word2idx:
        word2idx[word] = len(word2idx)
glove_file.close()
```

```
[7]: #adding special words to the vocabulary
word2idx['<unk>']=len(word2idx)
word2idx['<unkcap>']=len(word2idx)
word2idx['<pad>']=len(word2idx)
```

## 0.3 Creating Custom Dataset loader for Train and Dev

```
[8]: class NERDataset(Dataset):
    def __init__(self, filename, word2idx):
        self.word2idx = word2idx
        self.label2idx = {}
        self.max_sent_len = 0
        self.data = []

        with open(filename, "r") as f:
            sentence, labels, boolean = [], [], []
            for line in f:
                line = line.strip()
                if len(line) == 0:
                    if len(sentence) > self.max_sent_len:
                        self.max_sent_len = len(sentence)
                    self.data.append((sentence, labels, boolean))
                    sentence, labels, boolean = [], [], []
                else:
                    parts = line.split(" ")
                    word = parts[1]
                    label = parts[2]
                    if word[0].isupper():
                        boolean.append(1)
                    else:
                        boolean.append(0)
```

```

        #word=word.lower()
        if word.lower() not in self.word2idx:
            word = '<unkcap>' if word[0].isupper() else '<unk>'
        if label not in self.label2idx:
            self.label2idx[label] = len(self.label2idx)
        sentence.append(self.word2idx[word.lower()])
        labels.append(self.label2idx[label])

    if len(sentence) > 0:
        if len(sentence) > self.max_sent_len:
            self.max_sent_len = len(sentence)
        self.data.append((sentence, labels, boolean))

    self.word2idx['<pad>'] = len(self.word2idx)
    self.pad_idx = self.word2idx['<pad>']

    self.x, self.y, self.mask, self.lengths = [], [], [], []
    for sentence, labels, boolean in self.data:
        self.lengths.append(len(sentence))
        self.x.append(torch.tensor(sentence))
        self.y.append(torch.tensor(labels))
        self.mask.append(torch.tensor(boolean))

    self.x = pad_sequence(self.x, batch_first=True, padding_value=self.
↪pad_idx)
    self.y = pad_sequence(self.y, batch_first=True, padding_value=self.
↪pad_idx)
    self.mask = pad_sequence(self.mask, batch_first=True,
↪padding_value=self.pad_idx)

    print('done')

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.x[index], self.lengths[index], self.y[index], self.
↪mask[index]

```

```

[9]: class ValidateNERDataset(Dataset):
    def __init__(self, filename, word2idx, label2idx):
        self.data = []
        self.word2idx = word2idx
        self.label2idx = label2idx
        self.max_sent_len = 0

```

```

with open(filename, "r") as f:
    sentence, labels, boolean = [], [], []
    for line in f:
        line = line.strip()
        if len(line) == 0:
            if len(sentence) > self.max_sent_len:
                self.max_sent_len = len(sentence)
            self.data.append((sentence, labels, boolean))
            sentence, labels, boolean = [], [], []
        else:
            parts = line.split(" ")
            word = parts[1]
            label = parts[2]
            if word[0].isupper():
                boolean.append(1)
            else:
                boolean.append(0)
            #word=word.lower()
            if word.lower() not in self.word2idx:
                word = '<unkcap>' if word[0].isupper() else '<unk>'
            sentence.append(self.word2idx[word.lower()])
            labels.append(self.label2idx[label])

    if len(sentence) > 0:
        if len(sentence) > self.max_sent_len:
            self.max_sent_len = len(sentence)
        self.data.append((sentence, labels, boolean))

    self.word2idx['<pad>'] = len(self.word2idx)
    self.pad_idx = self.word2idx['<pad>']

    self.x, self.y, self.mask, self.lengths = [], [], [], []
    for sentence, labels, boolean in self.data:
        self.lengths.append(len(sentence))
        self.x.append(torch.tensor(sentence))
        self.y.append(torch.tensor(labels))
        self.mask.append(torch.tensor(boolean))

    self.x = pad_sequence(self.x, batch_first=True, padding_value=self.
↪pad_idx)
    self.y = pad_sequence(self.y, batch_first=True, padding_value=self.
↪pad_idx)
    self.mask = pad_sequence(self.mask, batch_first=True,
↪padding_value=self.pad_idx)

    print('done')

```

```

def __len__(self):
    return len(self.data)

def __getitem__(self, index):
    return self.x[index], self.lengths[index], self.y[index], self.
↪mask[index]

```

[9]:

## 0.4 BLSTM Modelwith Glove Embedding

```

[45]: class BLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim,
↪dropout, label_dim, embedding_mat):
        super().__init__()
        self.embedding = nn.Embedding(embedding_mat.shape[0], embedding_mat.
↪shape[1])
        self.embedding.weight.data.copy_(embedding_mat)
        self.embedding.weight.requires_grad = False
        self.lstm = nn.LSTM(embedding_dim+1, hidden_dim, bidirectional=True)
        self.fc = nn.Linear(hidden_dim * 2, output_dim)
        self.dropout = nn.Dropout(dropout)
        self.act = nn.ELU()
        self.classifier = nn.Linear(output_dim, label_dim)

    def forward(self, x, x_lengths, boolean):

        embedded = self.embedding(x)

        #Adding Boolean mask
        stacked_tensor = torch.cat((embedded, boolean.unsqueeze(-1)), dim=-1)

        packed_embedded = nn.utils.rnn.pack_padded_sequence(stacked_tensor,
↪x_lengths, batch_first=True, enforce_sorted=False)
        packed_output, (hidden, cell) = self.lstm(packed_embedded)
        output, output_lengths = nn.utils.rnn.
↪pad_packed_sequence(packed_output, batch_first=True)
        output = self.dropout(output)
        output=self.fc(output)
        output = self.act(output)
        output=self.classifier(output)
        output = output.permute(0, 2, 1)
        return output

```

## 0.5 Setting the hyperparameter

```
[46]: batch_size = 32
train_dataset = NERDataset('train',word2idx)

# Hyperparameters
EMBEDDING_DIM = 100
HIDDEN_DIM = 256
OUTPUT_DIM = 128
OUTPUT_LABEL_DIM = len(train_dataset.label2idx)
DROPOUT = 0.33
LEARNING_RATE = .2
EPOCHS = 30
STEP_SIZE = 20
GAMMA = 1
```

done

## 0.6 Building the Embedding Matrix from Glove Embedding Dictionary we formed

```
[47]: vectors = list(embeddings_dictionary.values())
unk_vector = .5*np.mean(vectors, axis=0)
unk_cap_vector = np.mean(vectors, axis=0)
#not needed cos anyway i have mebedding matrix all 0 initially
pad=np.zeros(100)
```

```
[48]: embedding_matrix = torch.zeros((len(train_dataset.word2idx)+1, EMBEDDING_DIM))
```

```
[49]: #unk tags tak mean of all vecotrs
#unkcap have .5 *unk
#pad all 0

for word,index in word2idx.items():
    if word=='<unk>':
        embedding_matrix[index]=torch.from_numpy(unk_vector)
    elif word=='<unkcap>':
        embedding_matrix[index]=torch.from_numpy(unk_cap_vector)
    elif word=='<pad>':
        embedding_matrix[index]=torch.from_numpy(pad)
    else:
        embedding_vector = embeddings_dictionary.get(word)
        embedding_matrix[index] = torch.from_numpy(embedding_vector)
```

```
[50]: device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = BLSTM(len(train_dataset.word2idx), EMBEDDING_DIM, HIDDEN_DIM,
↳OUTPUT_DIM, DROPOUT, OUTPUT_LABEL_DIM,embedding_matrix).to(device)
```

```

train_loader = DataLoader(train_dataset, batch_size=batch_size, pin_memory=True)
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE, momentum=0.9)
scheduler = StepLR(optimizer, step_size=STEP_SIZE, gamma=GAMMA)
freq=list(d.values())
sum_freq = sum(freq)
result = [(3.5 - f/sum_freq) for f in freq]
class_weights = torch.tensor(result).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=train_dataset.
    ↪pad_idx, weight=class_weights).to(device)

```

```

[51]: dev_dataset = ValidateNERDataset('dev', train_dataset.word2idx, train_dataset.
    ↪label2idx)
dev_loader = DataLoader(dev_dataset)

```

done

## 0.7 Training the Model

```

[52]: # Train the model
for epoch in range(EPOCHS):
    model.train()

    for batch_idx, (x, lengths, y, boolean) in enumerate(train_loader):
        optimizer.zero_grad()

        target_packed_embedded = nn.utils.rnn.pack_padded_sequence(y.
    ↪to(device), lengths, batch_first=True, enforce_sorted=False)
        target, target_lengths = nn.utils.rnn.
    ↪pad_packed_sequence(target_packed_embedded, batch_first=True)

        #output = model(x.to(device), lengths.to(device))
        output = model(x.to(device), lengths.cpu(), boolean.to(device))

        loss = criterion(output, target.to(device))
        loss.backward()
        optimizer.step()
        scheduler.step()

        if batch_idx % 100 == 0:
            print(f"Epoch {epoch}, Batch {batch_idx}, Loss {loss.item()}")

        # Compute and print the validation loss
        if batch_idx % 2000 == 0:
            model.eval() # Set the model to evaluation mode

```

```

        with torch.no_grad():
            val_loss = 0
            for val_x, val_lengths, val_y, maskval in dev_loader:
                val_target_packed_embedded = nn.utils.rnn.
↪pack_padded_sequence(val_y.to(device), val_lengths, batch_first=True,
↪enforce_sorted=False)
                val_target, val_target_lengths = nn.utils.rnn.
↪pad_packed_sequence(val_target_packed_embedded, batch_first=True)

                val_output = model(val_x.to(device), val_lengths.
↪cpu(),maskval.to(device))
                val_loss += criterion(val_output, val_target.to(device)).
↪item()

            print(f"Epoch {epoch}, Batch {batch_idx}, Validation Loss,
↪{val_loss/len(dev_loader)}")
            model.train() # Set the model back to training mode

```

```

Epoch 0, Batch 0, Loss 2.1771366596221924
Epoch 0, Batch 0, Validation Loss 2.0378872593633712
Epoch 0, Batch 100, Loss 0.18912072479724884
Epoch 0, Batch 200, Loss 0.09165876358747482
Epoch 0, Batch 300, Loss 0.08963826298713684
Epoch 0, Batch 400, Loss 0.06601230055093765
Epoch 1, Batch 0, Loss 0.09416676312685013
Epoch 1, Batch 0, Validation Loss 0.20208752412927436
Epoch 1, Batch 100, Loss 0.05514644458889961
Epoch 1, Batch 200, Loss 0.053480081260204315
Epoch 1, Batch 300, Loss 0.05889945477247238
Epoch 1, Batch 400, Loss 0.048128169029951096
Epoch 2, Batch 0, Loss 0.07594601064920425
Epoch 2, Batch 0, Validation Loss 0.13323234040283588
Epoch 2, Batch 100, Loss 0.04499710723757744
Epoch 2, Batch 200, Loss 0.034022632986307144
Epoch 2, Batch 300, Loss 0.053188879042863846
Epoch 2, Batch 400, Loss 0.04046478867530823
Epoch 3, Batch 0, Loss 0.06296923756599426
Epoch 3, Batch 0, Validation Loss 0.1135679487895133
Epoch 3, Batch 100, Loss 0.04381031543016434
Epoch 3, Batch 200, Loss 0.030425813049077988
Epoch 3, Batch 300, Loss 0.04784045368432999
Epoch 3, Batch 400, Loss 0.035188764333724976
Epoch 4, Batch 0, Loss 0.05732797086238861
Epoch 4, Batch 0, Validation Loss 0.1028174135871335
Epoch 4, Batch 100, Loss 0.04201902449131012
Epoch 4, Batch 200, Loss 0.03189961612224579
Epoch 4, Batch 300, Loss 0.0414433516561985

```



Epoch 4, Batch 400, Loss 0.030121084302663803  
Epoch 5, Batch 0, Loss 0.04852161556482315  
Epoch 5, Batch 0, Validation Loss 0.09460323478941486  
Epoch 5, Batch 100, Loss 0.039695169776678085  
Epoch 5, Batch 200, Loss 0.02245660126209259  
Epoch 5, Batch 300, Loss 0.0415397547185421  
Epoch 5, Batch 400, Loss 0.028133371844887733  
Epoch 6, Batch 0, Loss 0.04665520414710045  
Epoch 6, Batch 0, Validation Loss 0.08799094289545714  
Epoch 6, Batch 100, Loss 0.03556210920214653  
Epoch 6, Batch 200, Loss 0.01714201644062996  
Epoch 6, Batch 300, Loss 0.03838273137807846  
Epoch 6, Batch 400, Loss 0.028016014024615288  
Epoch 7, Batch 0, Loss 0.04711184650659561  
Epoch 7, Batch 0, Validation Loss 0.0832627987082479  
Epoch 7, Batch 100, Loss 0.031504325568675995  
Epoch 7, Batch 200, Loss 0.017077775672078133  
Epoch 7, Batch 300, Loss 0.03371096029877663  
Epoch 7, Batch 400, Loss 0.027153320610523224  
Epoch 8, Batch 0, Loss 0.046937618404626846  
Epoch 8, Batch 0, Validation Loss 0.07954265358373377  
Epoch 8, Batch 100, Loss 0.030757179483771324  
Epoch 8, Batch 200, Loss 0.014135724864900112  
Epoch 8, Batch 300, Loss 0.03408756107091904  
Epoch 8, Batch 400, Loss 0.022798683494329453  
Epoch 9, Batch 0, Loss 0.03554520756006241  
Epoch 9, Batch 0, Validation Loss 0.07547263912556734  
Epoch 9, Batch 100, Loss 0.029879285022616386  
Epoch 9, Batch 200, Loss 0.014797782525420189  
Epoch 9, Batch 300, Loss 0.032002441585063934  
Epoch 9, Batch 400, Loss 0.023935144767165184  
Epoch 10, Batch 0, Loss 0.03807394206523895  
Epoch 10, Batch 0, Validation Loss 0.07204590031477563  
Epoch 10, Batch 100, Loss 0.024188298732042313  
Epoch 10, Batch 200, Loss 0.014040793292224407  
Epoch 10, Batch 300, Loss 0.02901799976825714  
Epoch 10, Batch 400, Loss 0.020314117893576622  
Epoch 11, Batch 0, Loss 0.03190286085009575  
Epoch 11, Batch 0, Validation Loss 0.07000171070203848  
Epoch 11, Batch 100, Loss 0.021754654124379158  
Epoch 11, Batch 200, Loss 0.014514250680804253  
Epoch 11, Batch 300, Loss 0.025099582970142365  
Epoch 11, Batch 400, Loss 0.018941810354590416  
Epoch 12, Batch 0, Loss 0.03389151766896248  
Epoch 12, Batch 0, Validation Loss 0.06841743832796092  
Epoch 12, Batch 100, Loss 0.022400442510843277  
Epoch 12, Batch 200, Loss 0.012180590070784092  
Epoch 12, Batch 300, Loss 0.03280370682477951

Epoch 12, Batch 400, Loss 0.018208565190434456  
Epoch 13, Batch 0, Loss 0.029940025880932808  
Epoch 13, Batch 0, Validation Loss 0.0665087758757449  
Epoch 13, Batch 100, Loss 0.019352976232767105  
Epoch 13, Batch 200, Loss 0.012225599959492683  
Epoch 13, Batch 300, Loss 0.024557698518037796  
Epoch 13, Batch 400, Loss 0.019890638068318367  
Epoch 14, Batch 0, Loss 0.024747727438807487  
Epoch 14, Batch 0, Validation Loss 0.06503406008315656  
Epoch 14, Batch 100, Loss 0.019865866750478745  
Epoch 14, Batch 200, Loss 0.0103210574015975  
Epoch 14, Batch 300, Loss 0.023174572736024857  
Epoch 14, Batch 400, Loss 0.017381194978952408  
Epoch 15, Batch 0, Loss 0.02457638643682003  
Epoch 15, Batch 0, Validation Loss 0.06460177380930986  
Epoch 15, Batch 100, Loss 0.014516841620206833  
Epoch 15, Batch 200, Loss 0.011039137840270996  
Epoch 15, Batch 300, Loss 0.02368372119963169  
Epoch 15, Batch 400, Loss 0.015873687341809273  
Epoch 16, Batch 0, Loss 0.027697453275322914  
Epoch 16, Batch 0, Validation Loss 0.0633585461663978  
Epoch 16, Batch 100, Loss 0.01792646199464798  
Epoch 16, Batch 200, Loss 0.00958426296710968  
Epoch 16, Batch 300, Loss 0.020488113164901733  
Epoch 16, Batch 400, Loss 0.015759935602545738  
Epoch 17, Batch 0, Loss 0.022112244740128517  
Epoch 17, Batch 0, Validation Loss 0.06261751205773035  
Epoch 17, Batch 100, Loss 0.015503007918596268  
Epoch 17, Batch 200, Loss 0.009411104954779148  
Epoch 17, Batch 300, Loss 0.014281835407018661  
Epoch 17, Batch 400, Loss 0.011961434036493301  
Epoch 18, Batch 0, Loss 0.01971888355910778  
Epoch 18, Batch 0, Validation Loss 0.06320681494168248  
Epoch 18, Batch 100, Loss 0.011514031328260899  
Epoch 18, Batch 200, Loss 0.009041204117238522  
Epoch 18, Batch 300, Loss 0.013813057914376259  
Epoch 18, Batch 400, Loss 0.014438800513744354  
Epoch 19, Batch 0, Loss 0.022432832047343254  
Epoch 19, Batch 0, Validation Loss 0.06288394058057642  
Epoch 19, Batch 100, Loss 0.012833742424845695  
Epoch 19, Batch 200, Loss 0.007975730113685131  
Epoch 19, Batch 300, Loss 0.019686635583639145  
Epoch 19, Batch 400, Loss 0.012448363937437534  
Epoch 20, Batch 0, Loss 0.02176596038043499  
Epoch 20, Batch 0, Validation Loss 0.06253270520192016  
Epoch 20, Batch 100, Loss 0.014254134148359299  
Epoch 20, Batch 200, Loss 0.00903613492846489  
Epoch 20, Batch 300, Loss 0.014572606422007084

Epoch 20, Batch 400, Loss 0.011624954640865326  
Epoch 21, Batch 0, Loss 0.014061275869607925  
Epoch 21, Batch 0, Validation Loss 0.06261052503220378  
Epoch 21, Batch 100, Loss 0.011915216222405434  
Epoch 21, Batch 200, Loss 0.009701368398964405  
Epoch 21, Batch 300, Loss 0.01602853462100029  
Epoch 21, Batch 400, Loss 0.012490930035710335  
Epoch 22, Batch 0, Loss 0.020999066531658173  
Epoch 22, Batch 0, Validation Loss 0.0623825811202699  
Epoch 22, Batch 100, Loss 0.01053980179131031  
Epoch 22, Batch 200, Loss 0.0071085188537836075  
Epoch 22, Batch 300, Loss 0.009974798187613487  
Epoch 22, Batch 400, Loss 0.008262149058282375  
Epoch 23, Batch 0, Loss 0.013811985962092876  
Epoch 23, Batch 0, Validation Loss 0.06215185439985998  
Epoch 23, Batch 100, Loss 0.009438501670956612  
Epoch 23, Batch 200, Loss 0.007254648022353649  
Epoch 23, Batch 300, Loss 0.011144937016069889  
Epoch 23, Batch 400, Loss 0.010917844250798225  
Epoch 24, Batch 0, Loss 0.010026111267507076  
Epoch 24, Batch 0, Validation Loss 0.06229647865335271  
Epoch 24, Batch 100, Loss 0.009389537386596203  
Epoch 24, Batch 200, Loss 0.006547275464981794  
Epoch 24, Batch 300, Loss 0.010171162895858288  
Epoch 24, Batch 400, Loss 0.009658551774919033  
Epoch 25, Batch 0, Loss 0.011995739303529263  
Epoch 25, Batch 0, Validation Loss 0.0626744486698716  
Epoch 25, Batch 100, Loss 0.008927025832235813  
Epoch 25, Batch 200, Loss 0.006820125039666891  
Epoch 25, Batch 300, Loss 0.008275952190160751  
Epoch 25, Batch 400, Loss 0.010458706878125668  
Epoch 26, Batch 0, Loss 0.012470036745071411  
Epoch 26, Batch 0, Validation Loss 0.06541723003507659  
Epoch 26, Batch 100, Loss 0.007058395072817802  
Epoch 26, Batch 200, Loss 0.005971649195998907  
Epoch 26, Batch 300, Loss 0.010014999657869339  
Epoch 26, Batch 400, Loss 0.008409903384745121  
Epoch 27, Batch 0, Loss 0.009136774577200413  
Epoch 27, Batch 0, Validation Loss 0.0640439717528075  
Epoch 27, Batch 100, Loss 0.008130069822072983  
Epoch 27, Batch 200, Loss 0.0067215412855148315  
Epoch 27, Batch 300, Loss 0.006466693244874477  
Epoch 27, Batch 400, Loss 0.009291666559875011  
Epoch 28, Batch 0, Loss 0.008443073369562626  
Epoch 28, Batch 0, Validation Loss 0.06583114181395722  
Epoch 28, Batch 100, Loss 0.007471282035112381  
Epoch 28, Batch 200, Loss 0.008127037435770035  
Epoch 28, Batch 300, Loss 0.008621660061180592

```
Epoch 28, Batch 400, Loss 0.007481699343770742
Epoch 29, Batch 0, Loss 0.008564052172005177
Epoch 29, Batch 0, Validation Loss 0.06520530858995174
Epoch 29, Batch 100, Loss 0.00701866066083312
Epoch 29, Batch 200, Loss 0.00598725862801075
Epoch 29, Batch 300, Loss 0.009616649709641933
Epoch 29, Batch 400, Loss 0.010477243922650814
```

[52]:

## 1 Model Saving and Loading

[53]: `torch.save(model.state_dict(), 'blstm2.pt')`

[54]: `model2 = BLSTM(len(word2idx), EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, DROPOUT,   
↳ OUTPUT_LABEL_DIM, embedding_matrix).to(device)`  
  
`model2.load_state_dict(torch.load('blstm2.pt'))`

[54]: <All keys matched successfully>

### 1.1 Dev Prediction

[55]: `from sklearn.metrics import precision_recall_fscore_support`  
  
`model2.eval()`  
`predicted_labels = []`  
`true_labels = []`  
  
`with torch.no_grad():`  
    `for x, lengths, y, mask in dev_loader:`  
        `x = x.to(device)`  
  
        `y = y.to(device)`  
  
        `target_packed_embedded = nn.utils.rnn.pack_padded_sequence(y, lengths,   
↳ batch_first=True, enforce_sorted=False)`  
        `target, target_lengths = nn.utils.rnn.`  
        `↳ pad_packed_sequence(target_packed_embedded, batch_first=True)`  
        `#print('y', y)`  
        `#print('target', target)`  
        `output = model2(x, lengths, mask.to(device))`  
  
        `predicted = torch.argmax(output, dim=1)`  
        `predicted_labels.extend(predicted.cpu().numpy().tolist())`  
        `true_labels.extend(target.cpu().numpy().tolist())`

```

[56]: y_pred = [element for sub_list in predicted_labels for element in sub_list]
      y_true=[element for sub_list in true_labels for element in sub_list]

      list1 = y_true.copy()
      list2 = y_pred.copy()

      value_to_remove = train_loader.dataset.pad_idx

      i = 0
      while i < len(list1):
          if list1[i] == value_to_remove:
              # remove the element from list1
              list1.pop(i)
              # remove the corresponding element from list2 by using its index
              list2.pop(i)
          else:
              # only increment the loop counter if an element wasn't removed
              i += 1

      print(list1)  # [1, 2, 4, 5, 6]
      print(list2)

```

```

[1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 2, 7, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 0,
1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 8, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4,
1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 5, 1, 3, 1, 1, 1, 1, 5, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 0, 1, 3, 4, 1, 1, 1, 1, 1, 5, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
5, 1, 3, 4, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
2, 3, 4, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 2, 1, 1, 3, 4, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 7, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 2,
1, 1, 1, 1, 1, 1, 5, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 3, 4, 1, 1,
1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 5, 1, 0, 1, 1, 1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 3,
4, 1, 1, 1, 0, 1, 1, 3, 4, 1, 1, 1, 1, 5, 8, 1, 0, 1, 1, 3, 4, 1, 1, 3, 4, 1,
1, 1, 0, 1, 1, 5, 1, 5, 8, 1, 1, 0, 1, 1, 0, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1,
3, 4, 1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 1, 5, 1, 0, 1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 3,
4, 1, 1, 1, 0, 1, 1, 3, 4, 1, 1, 5, 1, 0, 1, 1, 1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 3,
4, 1, 1, 1, 0, 1, 1, 1, 1, 5, 1, 0, 1, 1, 1, 1, 0, 1, 1, 3, 4, 1, 1, 3, 1, 1, 1,
1, 3, 4, 1, 1, 3, 4, 1, 1, 5, 1, 0, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 0, 1, 1, 3,

```

4	1	1	1	3	4	1	1	1	1	1	1	1	1	1	2	1	1	5	1	5	1	1	1	2	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	6	6	6	6	6	1	1	1	1	5	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	
1	0	6	1	1	1	1	1	1	1	1	1	0	6	1	5	1	1	1	1	1	1	1	1	1	1	1	5	1	1
1	1	5	8	1	1	1	0	6	6	6	6	6	1	1	5	1	1	1	1	0	1	1	1	0	1	1	1	1	
0	1	1	1	1	1	1	1	5	1	0	1	1	1	1	1	1	1	1	1	5	8	1	5	1	1	1	1	1	
1	1	1	1	5	8	1	5	1	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	5	1	5	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	5	8	1	1	1	1	0	6	1	1	5	1	1	
1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	5	8	1	5	1	1	1	1	1	
0	6	6	1	1	1	5	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	
1	1	5	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	5	8	1	5	1	1	1	1	1	1	
0	1	1	1	1	1	1	1	1	1	1	1	5	8	1	5	1	1	1	1	1	1	3	1	1	5	1	1	5	
1	1	1	1	1	1	1	3	4	1	1	1	1	1	5	1	1	1	1	1	1	1	1	1	1	0	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	3	4	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	
1	3	4	1	3	1	3	1	1	1	1	1	1	5	1	1	2	7	1	1	5	1	1	1	3	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	3	4	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	1	1	1	1	1	
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
2	7	1	1	3	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	1	0	1	1	1	1	
3	4	1	1	1	1	1	1	1	1	5	1	1	1	1	1	1	1	1	1	0	6	1	5	1	1				

[illegible]

[illegible]



[illegible]

[illegible]







[illegible]









[illegible]

[illegible]



[illegible]





[illegible]



[illegible]

6,	1,	1,	2,	1,	1,	0,	6,	1,	0,	6,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	2,	1,	1,	1,	1,
1,	1,	1,	1,	1,	5,	8,	1,	1,	1,	1,	1,	1,	1,	1,	1,	2,	0,	1,	1,	1,	0,	6,	1,	3,	4,	
1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	2,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	2,	1,	1,	3,	4,	1,	1,	1,	1,	5,	1,	1,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	5,	8,	8,	8,	1,	1,	1,	1,	5,	1,	1,	0,	6,	6,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	2,	1,	1,	1,	1,	1,	5,	8,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	0,	1,	1,	1,	1,	1,	1,	2,		
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	
8,	8,	1,	1,	1,	1,	0,	1,	1,	1,	1,	1,	1,	1,	1,	5,	8,	1,	5,	1,	1,	1,	1,	1,	1,	3,	
4,	1,	1,	1,	1,	0,	6,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	2,	1,	1,	1,	1,	1,	
2,	1,	1,	1,	1,	1,	1,	1,	0,	1,	1,	1,	1,	1,	1,	1,	1,	1,	0,	6,	1,	0,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	0,	1,	3,	4,	1,	1,	1,	1,	1,	1,	2,	2,	1,	1,	3,	1,	
1,	0,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	0,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	0,	1,	1,	1,	1,	1,	1,	1,	0,	1,	1,	
1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	0,	1,	1,	1,	1,	1,	1,	1,	1,	0,	1,	3,	4,	1,	1,	1,	1,	1,	
0,	1,	1,	2,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	1,	1,	
1,	1,	3,	4,	1,	1,	1,	1,	3,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	2,	7,	1,	0,	1,	0,	1,	1,	1,	1,	5,	1,	5,	1,	1,	2,	1,	1,	1,	1,	1,	1,	1,	1,	0,	
1,	0,	6,	1,	1,	1,	1,	5,	1,	5,	1,	2,	1,	1,	1,	5,	1,	5,	1,	1,	2,	1,	1,	1,	1,	1,	
1,	1,	1,	5,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	2,	1,	1,	1,	1,	1,	1,	5,												

[illegible]

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 3, 4, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 6, 1, 1,  
2, 7, 7, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 5, 1, 1, 1, 1, 1, 5, 1, 3, 4, 1, 1,  
1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 6, 1, 1, 0, 1, 1, 5, 8,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 3, 4, 1, 3, 4, 1, 3, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 3, 4, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 3, 4, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 1, 0, 6, 1, 1, 1, 1, 3, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
3, 4, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 6, 1, 1, 2, 7, 1,  
1, 1, 1, 0, 1, 1, 1, 1, 1, 5, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 6, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 3,  
1, 3, 4, 1, 3, 4, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 2, 1,  
1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 6, 6, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 3, 4, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 3, 4, 4, 1, 1, 1, 1,  
1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 6, 6, 1, 1, 3, 4, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,  
1, 1, 1, 1, 1, 5, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 3, 4, 1, 1, 3, 4, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 0, 1, 1, 1, 1,  
1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 3, 4, 1, 1,  
1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 5, 1, 0,  
1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,  
1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 3, 4,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 8, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,  
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,  
1, 0, 1, 3, 4, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 3, 4, 1, 1, 0, 1,  
2, 1, 3, 4, 1,  
1,  
1, 1, 1, 1, 1, 2, 7, 1, 5, 1, 1, 1, 1, 1, 1, 2, 7, 1, 1, 2, 7, 1, 1, 3, 4, 1, 1,  
1, 2, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 3, 4, 1, 5, 1, 1, 1, 3, 4, 1,  
1, 3, 4, 1, 5, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 3, 4, 1, 1, 3, 4, 1, 5, 1, 1, 1, 3,  
4, 1, 1, 3, 4, 1, 1, 3, 4, 1, 5, 1, 1, 1, 3, 4, 1, 5, 8, 1, 1, 1, 3, 4, 1, 5, 1,  
1, 1, 3, 4, 1, 5, 1, 1, 1, 3, 4, 1, 5, 8, 1, 1, 1, 3, 4, 4, 1, 5, 1, 1, 1, 3, 4,  
1, 1, 3, 4, 4, 1, 5, 1, 1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 0, 6, 1, 2, 1, 2, 1, 2, 1,  
1, 5, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 7, 7, 1, 0, 1, 0, 1, 0, 6, 1,  
0, 1, 0, 1, 0, 6, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 2, 1, 1, 0, 1, 0, 1, 0,  
1, 0, 6, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 2, 1, 1, 1, 0, 1, 0, 1, 0, 1,  
0, 6, 6, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 5, 1, 5, 8, 1, 2, 7, 1, 1, 5, 1,  
5, 1, 5, 8, 1, 1, 1, 1, 1, 1, 2, 7, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1,  
1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 5, 1, 5, 8, 1, 2, 7, 1, 1, 5, 1, 5, 1, 5, 8, 1, 1, 1, 1, 1, 1, 2,  
7, 1, 2, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 3, 1, 1,  
1, 0, 1, 0, 1, 5, 1, 1, 0, 1, 3, 4, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 5, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1,



[illegible]



[illegible]





[illegible]

[illegible]

5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	2,	3,	4,	1,	1,	
1,	0,	1,	1,	1,	1,	1,	1,	3,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	3,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	3,	4,	4,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	
1,	1,	1,	1,	1,	5,	1,	3,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	3,	1,	0,	1,	1,	1,	5,	1,	1,	1,	1,	
2,	3,	4,	1,	1,	1,	0,	1,	1,	1,	1,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	2,	7,	1,	1,	1,	5,	1,	1,	1,	1,	2,	7,	2,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	5,	8,	1,	1,	1,	1,	3,	4,	1,	1,	1,	1,	1,	3,	4,	1,	1,	3,	4,	1,	1,	3,	4,	1,	1,	3,	4,	1,	1,	
1,	3,	1,	1,	3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	1,	3,	4,	1,	1,	1,	1,	5,	1,	1,	1,	1,	
3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	3,	1,	1,	1,	2,	7,	7,	1,	5,	1,	2,	1,	1,	1,	1,	1,	1,	
3,	4,	1,	1,	1,	1,	1,	1,	1,	1,	2,	7,	1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	0,	6,	1,	1,	1,	3,	1,	1,	1,	
1,	1,	1,	5,	1,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	2,	1,	1,	5,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	
1,	1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	5,	1,	1,	1,	1,	1,	5,	1,	5,	1,	1,	1,	1,	1,	1,	1,	5,	1,	1,	1,	1,	1,
1,	1,	1,	1,	1,	1,	1,	1,	5,	1,	3,	4,	1,	1,	1,	1,	3,	4,	1,	1,	1,	5,	1,	3,	4,	1,	1,	1,</					

1	1	2	7	7	1	2	1	1	3	4	1	1	1	2	1	1	1	1	5	1	1	1	1	1	1	1
1	1	1	1	1	1	2	1	1	1	3	4	1	1	2	7	7	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	4	1	1	1	1	1	1	5	8	1	5
8	1	1	1	1	1	1	1	1	1	5	8	1	2	1	1	1	1	5	8	1	2	1	2	1	1	1
1	1	1	1	5	1	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	3
4	4	4	4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5	1	1	1
1	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	5	1	5	1	5
5	1	5	1	5	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	2	7	7	1	1	1
1	1	1	1	1	1	5	1	1	1	5	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	5	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	3	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	5	1	1	1	1	5	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5	1	3	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5	1	1	1	1	1	5	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	6	6	1	1	1	1	1	1	1	1
1	5	1	0	6	6	1	3	4	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	5	1	3	1	1	1	1	1	1	2	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	4	1	1	1	1
1	1	1	2	1	2	1	1	1	1	1	1	1	1	0	6	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	5	1	0	1	1	1	0	6	6	1	1	5	1	5	1
1	5	1	1	5	1	5	1	1	1	2	1	1	1	1	1	1	2	1	1	5	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	2	1	1									

[illegible]

[illegible]







[illegible]







[illegible]

[illegible]

[illegible]



[illegible]





[illegible]





[illegible]

[illegible]









[illegible]

[illegible]



[illegible]

[illegible]





1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,  
1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 8, 1, 5, 1, 1, 1, 1, 1, 1, 3,  
4, 1, 1, 1, 1, 0, 6, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,  
2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 0, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 3,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,  
1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 3, 4, 1, 1, 1, 1, 1,  
0, 1, 1, 2, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1,  
1, 1, 3, 4, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1,  
1, 2, 7, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 5, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0,  
1, 0, 6, 1, 1, 1, 1, 5, 1, 5, 1, 2, 1, 1, 1, 5, 1, 5, 1, 1, 1, 2, 1, 1, 1, 1,  
1, 1, 1, 5, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 5, 1, 5, 1, 1, 5, 8, 8, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 2, 1, 1, 1, 1, 2, 7, 1, 1, 1, 1, 2, 1, 1, 1, 1,  
3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 0, 1, 3, 4, 1, 0, 1, 3, 4, 1, 1,  
3, 4, 1, 3, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 5, 1, 5, 1, 1, 2, 7, 1, 1,  
5, 1, 1, 5, 8, 8, 1, 5, 1, 1, 1, 1, 1, 1, 1, 2, 7, 1, 2, 1, 1, 1, 1, 1, 1,  
3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1,  
1, 1, 1, 5, 1, 2, 7, 1, 1, 1, 5, 1, 5, 1, 1, 1, 1, 1, 1, 1, 2, 7, 1, 1, 1,  
1, 1, 2, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 3, 4, 4, 1, 5, 1, 1, 1,  
1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 3, 4, 4, 1, 5, 1, 1,  
1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1,  
1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1,  
1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1,  
1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4,  
1, 5, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1, 3, 4,  
1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1,  
1, 1, 1, 1, 1, 5, 1, 5, 1, 2, 7, 1, 1, 5, 1, 5, 8, 1, 5, 1, 1, 5, 8, 1, 1, 1,  
1, 1, 1, 2, 7, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 8, 1, 3, 4, 4, 1, 1, 1, 5,  
1, 3, 4, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 5, 1, 1, 1, 1, 1,  
1, 1, 5, 1, 5, 1, 5, 1, 5, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 5, 1, 1, 1, 1, 3, 4, 1, 1, 1, 5, 1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 1, 1,  
1, 5, 1, 1, 1, 1, 1, 5, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 7, 7, 1,  
5, 1, 5, 1, 1, 1, 1, 3, 4, 1, 5, 1, 0, 1, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 0, 1,  
1, 1, 1, 3, 4, 1, 5, 1, 0, 1, 1, 1, 3, 4, 1, 5, 1, 0, 1, 1, 3, 4, 1, 5, 1, 0,  
1, 1, 3, 4, 1, 5, 1, 0, 1, 3, 4, 1, 5, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,  
4, 1, 5, 1, 0, 1, 1, 3, 4, 1, 5, 1, 0, 1, 1, 1, 1, 3, 1, 1, 3, 4, 1, 5, 1, 0, 1,  
1, 3, 4, 1, 5, 1, 0, 1, 1, 3, 1, 1, 3, 1, 1, 1, 3, 4, 4, 1, 5, 1, 0, 1, 1, 3, 1,  
1, 3, 1, 1, 1, 1, 5, 1, 5, 1, 1, 1, 5, 1, 5, 1, 1, 1, 1, 1, 1, 1, 5, 1, 5,  
1, 1, 1, 5, 3, 4, 1, 3, 4, 1, 3, 4, 1, 3, 4, 1, 3, 4, 1, 1, 3, 4, 1, 3,  
4, 1, 1, 3, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 3, 4,

[illegible]

[illegible]

[illegible]

1, 1, 1, 1, 5, 8, 1, 0, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 1, 1, 3, 4, 1, 1, 3,  
4, 1, 1, 3, 4, 1, 1, 1, 5, 1, 0, 1, 1, 1, 1, 3, 4, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 2, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 0, 4, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 5, 1, 1, 0, 7, 7, 1, 1, 1, 3, 4, 1, 5, 1, 1,  
0, 7, 1, 1, 1, 3, 4, 1, 5, 1, 1, 0, 7, 1, 1, 1, 3, 4, 1, 5, 1, 1, 0, 4, 1, 1, 1,  
3, 4, 1, 5, 1, 1, 0, 6, 1, 1, 1, 3, 4, 4, 1, 5, 1, 1, 0, 6, 1, 1, 1, 3, 4, 1, 5,  
1, 1, 0, 4, 1, 1, 1, 3, 4, 1, 5, 1, 1, 0, 7, 7, 1, 1, 1, 3, 4, 4, 1, 5, 1, 1, 0,  
6, 1, 1, 1, 1, 1, 5, 1, 5, 1, 1, 2, 7, 1, 1, 5, 1, 5, 1, 5, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 1, 1, 1, 1, 1, 5, 1, 3, 1, 1, 1, 1, 1, 1, 3, 4,  
1, 1, 1, 1, 3, 4, 1, 1, 1, 5, 1, 3, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 0, 1,  
1, 1, 1, 1, 1, 3, 4, 5, 1, 5, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5,  
8, 1,  
1, 1, 1, 1, 1, 1, 5, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 1, 1, 1,  
2, 7, 1, 1, 0, 1, 3, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,  
4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 1, 1, 1, 1, 1,  
1, 1, 3, 4, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1,  
1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 3, 1,  
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4,  
1, 1, 1, 1, 3, 1, 1, 1, 1, 2, 1, 3, 4, 1, 3, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 3,  
1, 1, 3, 1, 1, 1, 1, 0, 2, 1, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,  
1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 4, 1, 1, 1, 1, 2, 8, 1, 1, 1, 1, 1, 1, 3, 1, 1,  
1, 1, 1, 3, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 8, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4,  
1, 1, 1, 3, 4, 1, 3, 4, 1, 1, 1, 1, 1, 1, 3, 4, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4,  
1, 1, 1, 3, 4, 4, 4, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 3, 4, 1, 3, 4, 1, 1, 1,  
1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 3, 4, 4, 1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1,  
1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 3, 4, 1, 1, 1, 1, 5, 8, 1, 1, 1, 3, 4, 1, 3, 4, 1,  
1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4,  
1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 3, 4, 1, 1, 1,  
1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 1, 3, 4,  
1, 1, 0, 6, 1, 5, 8, 1, 0, 6, 1, 1, 5, 1, 5, 8, 1, 5, 8, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 5, 8, 1, 1, 1, 1, 1, 5, 8, 1, 1, 1, 3, 4, 4, 4, 1, 1, 1,  
3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 3, 1, 1, 1, 1, 3, 1, 1, 1, 1, 5, 8, 1, 1, 1, 3, 4,  
1, 3, 4, 1, 3, 4, 1, 1, 1, 3, 4, 1, 1, 1, 1, 1, 1, 3, 1, 5, 8, 1, 1, 1, 1, 1,  
1, 1, 5, 1, 1, 1, 1, 7, 1, 1, 5, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 1, 7, 1, 1, 1, 5, 1, 1, 1,  
1, 3, 4, 4, 1,  
1, 1, 2, 7, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 5, 1, 1, 5, 1, 2, 7, 7, 1, 5, 1, 5, 1, 1, 1, 5, 1, 1, 2, 7, 7, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 5, 1, 1, 1, 2, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 6, 6, 6, 1, 0, 1, 0, 6, 6, 6, 1, 0, 1, 0, 6, 6, 6,

[illegible]





[illegible]



[illegible]









[illegible]











[illegible]

```
[57]: precision, recall, f1_score, _ = precision_recall_fscore_support(list1, list2,
    ↪ average='weighted')
print(f'Precision: {precision:.4f}, Recall: {recall:.4f}, F1 score: {f1_score:.
    ↪ 4f}')
```

Precision: 0.9842, Recall: 0.9845, F1 score: 0.9843

## 1.2 dev.out file for evaluating on PERL

```
[58]: devOutput = open("dev.out", "w")
k=0
i=0
idx2label = {value: key for key, value in train_dataset.label2idx.items()}

with open('/content/dev', 'r') as f:
    for line in f:

        line = line.strip().split(' ')
        #print(line)
        if len(line)>1:
            idx,word,gold = line[0], line[1],line[2]
            pred=predicted_labels[k][i]
            i=i+1
            key = idx2label[pred]
            devOutput.write(f"{idx} {word} {gold} {key}\n")
        else:
            devOutput.write(f"{line[0]}\n")
            k=k+1
            i=0
f.close()
devOutput.close()
```

[58] :

### 1.3 dev2.out file for submission (in same format as train)

```
[59]: devOutput = open("dev2.out", "w")
      k=0
      i=0

      with open('/content/dev', 'r') as f:
```

```

for line in f:
    line = line.strip().split(' ')
    if len(line)>1:
        idx,word,gold = line[0], line[1],line[2]
        pred=predicted_labels[k][i]
        i=i+1
        key = idx2label[pred]
        devOutput.write(f"{idx} {word} {key}\n")
    else:
        devOutput.write(f"\n")
        k=k+1
        i=0
f.close()
devOutput.close()

```

[59]:

## 2 Test Prediction

```

[60]: class TestNERDataset(Dataset):
    def __init__(self, filename,word2idx):
        self.data = []
        self.word2idx = word2idx
        self.max_sent_len = 0

        with open(filename, "r") as f:
            sentence, boolean = [], []
            for line in f:
                line = line.strip()
                if len(line) == 0:
                    if len(sentence) > self.max_sent_len:
                        self.max_sent_len = len(sentence)
                    self.data.append((sentence, boolean))
                    sentence, boolean = [], []
                else:
                    parts = line.split(" ")
                    word = parts[1]
                    if word[0].isupper():
                        boolean.append(1)
                    else:
                        boolean.append(0)
                    word=word.lower()
                    if word not in self.word2idx:
                        word = '<unkcap>' if word[0].isupper() else '<unk>'
                    sentence.append(self.word2idx[word])

```

```

        if len(sentence) > 0:
            if len(sentence) > self.max_sent_len:
                self.max_sent_len = len(sentence)
            self.data.append((sentence, boolean))

        self.pad_idx = self.word2idx['<pad>']

        self.x, self.mask, self.lengths = [], [], []
        for sentence, boolean in self.data:
            self.lengths.append(len(sentence))
            self.x.append(torch.tensor(sentence))
            self.mask.append(torch.tensor(boolean))

        self.x = pad_sequence(self.x, batch_first=True, padding_value=self.
↪pad_idx)
        self.mask = pad_sequence(self.mask, batch_first=True,
↪padding_value=self.pad_idx)

        print('done')

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.x[index], self.lengths[index], self.mask[index]

```

```

[61]: test_dataset = TestNERDataset('test', train_dataset.word2idx)
      test_loader = DataLoader(test_dataset)

```

done

```

[62]: from sklearn.metrics import precision_recall_fscore_support

model2.eval()
predicted_labels = []
true_labels = []

with torch.no_grad():
    for x, lengths, mask in test_loader:
        x = x.to(device)

        output = model2(x, lengths, mask.to(device))

        predicted = torch.argmax(output, dim=1)
        predicted_labels.extend(predicted.cpu().numpy().tolist())

```

## 2.1 test2.out file for submission

```
[63]: testOutput = open("test2.out", "w")
k=0
i=0

with open('/content/test', 'r') as f:
    for line in f:
        line = line.strip().split(' ')
        if len(line)>1:
            idx,word = line[0], line[1]
            pred=predicted_labels[k][i]
            i=i+1
            key = idx2label[pred]
            testOutput.write(f"{idx} {word} {key}\n")
        else:
            testOutput.write(f"\n")
            k=k+1
            i=0
f.close()
testOutput.close()
```

```
[63]:
```