Homework_5_Release

March 15, 2022

0.1 Homework 5

0.1.1 NLP Pipelines

Welcome to Homework 5!

The homework contains several tasks. You can find the amount of points that you get for the correct solution in the task header. Maximum amount of points for each homework is *four*.

The **grading** for each task is the following: - correct answer - **full points** - insufficient solution or solution resulting in the incorrect output - **half points** - no answer or completely wrong solution - **no points**

Even if you don't know how to solve the task, we encourage you to write down your thoughts and progress and try to address the issues that stop you from completing the task.

When working on the written tasks, try to make your answers short and accurate. Most of the times, it is possible to answer the question in 1-3 sentences.

When writing code, make it readable. Choose appropriate names for your variables (`a = 'cat'` - not good, `word = 'cat'` - good). Avoid constructing lines of code longer than 100 characters (79 characters is ideal). If needed, provide the commentaries for your code, however, a good code should be easily readable without them :)

Finally, all your answers should be written only by yourself. If you copy them from other sources it will be considered as an academic fraud. You can discuss the tasks with your classmates but each solution must be individual.

Important!: before sending your solution, do the `Kernel -> Restart & Run All` to ensure that all your code works.

[1]:
```
!pip install --quiet datasets stanza conllu networkx torchmetrics
```

[2]:
```
from datasets import load_dataset
import stanza
import re
import matplotlib.pyplot as plt

import torch
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import TensorDataset, DataLoader, random_split
import torch.nn as nn
```

```
from torchmetrics.functional import f1_score
from tqdm.notebook import tqdm
import json
```

Load Relation Extraction on SemEval-2010 Task 8 dataset.

In this dataset, two words are marked with `<e1></e1>` and `<e2></e2>` symbols, and the label is a relation between them.

For example:

`The <e1>student</e1> <e2>association</e2> is the voice of the undergraduate student population of the State University of New York at Buffalo.`

`(Member-Collection(e1,e2))`

[3]:
```
dataset = load_dataset("sem_eval_2010_task_8")
```

```
Using custom data configuration default
Reusing dataset sem_eval2010_task8 (/root/.cache/huggingface/datasets/sem_eval20
10_task8/default/1.0.0/8545d1995bbbade386acf5c4e2bef5589d8387ae0a93356407dfb54cd
b234416)
```

```
  0%|          | 0/2 [00:00<?, ?it/s]
```

Load the Stanza pipeline.

[4]:
```
stanza.download('en')
```

```
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/
↪resources_1.3.0.json:   0%|   …
```

```
2022-03-14 21:49:30 INFO: Downloading default packages for language: en
(English)…
2022-03-14 21:49:34 INFO: File exists: /root/stanza_resources/en/default.zip.
2022-03-14 21:49:43 INFO: Finished downloading models and saved to
/root/stanza_resources.
```

[5]:
```
nlp = stanza.Pipeline('en', processors='tokenize,pos,lemma,depparse',␣
↪tokenize_no_ssplit=True)
```

```
2022-03-14 21:49:48 INFO: Loading these models for language: en (English):
========================
| Processor | Package  |
------------------------
| tokenize  | combined |
| pos       | combined |
| lemma     | combined |
| depparse  | combined |
========================

2022-03-14 21:49:48 INFO: Use device: gpu
2022-03-14 21:49:48 INFO: Loading: tokenize
```

```
2022-03-14 21:49:51 INFO: Loading: pos
2022-03-14 21:49:51 INFO: Loading: lemma
2022-03-14 21:49:51 INFO: Loading: depparse
2022-03-14 21:49:52 INFO: Done loading processors!
```

0.1.2 Task 1. Replace the entities (0.5 points)

Write a function to replace that entity tags and the text between them into `ENTITY1` and `ENTITY2` accordingly.

For example:

Input: `The <e1>student</e1> <e2>association</e2> is the voice of the undergraduate student population of the State University of New York at Buffalo.`

Output: `The ENTITY1 ENTITY2 is the voice of the undergraduate student population of the State University of New York at Buffalo.`

[7]:
```
def replace_entities(text):
    ### YOUR CODE HERE
    import re
    pattern = re.compile(r"</?e\d+>")
    sent = text.split()
    for i in range(len(sent)):
      if pattern.search(sent[i]):
        digit=re.findall(r'\d+',sent[i])
        sent[i] =f'ENTITY{digit[0]}'
    text=" ".join(sent)
    return text
```

0.2 Task 2. Preprocess the text. (1 point)

For each sentece in the dataset, create five outputs: - Lowercased tokens - Deprels for each token - Relative position of each token to the first entity (as string) - Relative position of each token to the second entity (as string) - Labels

Example:

Input: `The system as described above has its greatest application in an arrayed <e1>configuration</e1> of antenna <e2>elements</e2>`

Tokens: `['the', 'system', 'as', 'described', 'above', 'has', 'its', 'greatest', 'application', 'in', 'an', 'arrayed', 'entity1', 'of', 'antenna', 'entity2', '.']`

Deprels: `['det', 'nsubj', 'mark', 'acl', 'advmod', 'root', 'nmod:poss', 'amod', 'obj', 'case', 'det', 'amod', 'obl', 'case', 'nmod', 'flat', 'punct']`

Position 1: `['-12', '-11', '-10', '-9', '-8', '-7', '-6', '-5', '-4', '-3', '-2', '-1', '0', '1', '2', '3', '4']`

Position 2: ['-15', '-14', '-13', '-12', '-11', '-10', '-9', '-8', '-7', '-6', '-5', '-4', '-3', '-2', '-1', '0', '1']

[8]:
```
def preprocess(dataset, pipeline):
    doc = pipeline('\n\n'.join([replace_entities(item['sentence']) for item in␣
 ↪dataset]))
    sentences = []
    deprels = []
    pos_1 = []
    pos_2 = []
    relations = [item['relation'] for item in dataset]
    for sentence in doc.sentences:
        try:
            ent1_id = [word.id for word in sentence.words if 'ENTITY1' in word.
 ↪text][0] # Position of the first entity
            ent2_id = [word.id for word in sentence.words if 'ENTITY2' in word.
 ↪text][0] # Position of the second entity
        except IndexError as e:
            print([word.text for word in sentence.words])
            raise e

        ### YOUR CODE STARTS HERE
        sentences.append([word['text'] for word in sentence.to_dict()])
        deprels.append([word['deprel'] for word in sentence.to_dict()])
        pos_1.append([str( int(word['id'])-int(ent1_id)) for word in sentence.
 ↪to_dict()])
        pos_2.append([str(int(word['id'])-int(ent2_id)) for word in sentence.
 ↪to_dict()])
        ### YOUR CODE ENDS HERE

    return sentences, deprels, pos_1, pos_2, relations
```

Preprocess train and test sets.

[9]:
```
train_sents, train_deprels, train_pos_1, train_pos_2, train_rels =␣
 ↪preprocess(dataset['train'], nlp)
test_sents, test_deprels, test_pos_1, test_pos_2, test_rels =␣
 ↪preprocess(dataset['test'], nlp)
```

Download and unzip word vectors.

[10]:
```
!wget https://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2022-03-14 21:53:05--  https://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)… 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443…
connected.
HTTP request sent, awaiting response… 301 Moved Permanently
```

```
Location: http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2022-03-14 21:53:05--  http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)… 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:80… connected.
HTTP request sent, awaiting response… 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip.1'

glove.6B.zip.1      100%[===================>] 822.24M  5.08MB/s    in 2m 40s

utime(glove.6B.zip.1): No such file or directory
2022-03-14 21:55:45 (5.14 MB/s) - 'glove.6B.zip.1' saved [862182613/862182613]

Archive:  glove.6B.zip
replace glove.6B.50d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: glove.6B.50d.txt
replace glove.6B.100d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

[11]:
```
# Load the embeddings into the memory
glove_path = 'glove.6B.300d.txt'
glove_vecs = []
idx2token = []
with open(glove_path, encoding='utf-8') as f:
    for line in tqdm(f):
        line = line.strip().split()
        word = line[0]
        vec = [float(x) for x in line[1:]]
        glove_vecs.append(vec)
        idx2token.append(word)

# Convert to tensor
glove_vecs = torch.tensor(glove_vecs)

# Put zero vector for padding and mean for unknown
glove_vecs = torch.vstack(
    [
        torch.zeros(1, glove_vecs.size(1)),
        torch.mean(glove_vecs, dim=0).unsqueeze(0),
        torch.rand(1, glove_vecs.size(1)),
        torch.rand(1, glove_vecs.size(1)),
        glove_vecs,
    ]
)
```

```
# Save the embeddings in Pytorch format
torch.save(glove_vecs, 'glove.6B.300d.pt')

# Add special pad and unk tokens to the vocab
PAD = '<pad>'
PAD_ID = 0
UNK = '<unk>'
UNK_ID = 1
E1 = 'entity1'
E1_ID = 2
E2 = 'entity2'
E2_ID = 3

idx2token = [PAD, UNK, E1, E2] + idx2token

# Save the vocab
json.dump(idx2token, open('idx2token.json', 'w', encoding='utf-8'))
```

```
0it [00:00, ?it/s]
```

[12]:
```
token2idx = {token: idx for idx, token in enumerate(idx2token)}
```

Build vocabularies for deprels and relative positions.

[13]:
```
deprel_idx2token = list(set([deprel for sent in train_deprels for deprel in␣
↪sent]))
deprel_idx2token = [PAD, UNK] + deprel_idx2token
deprel_token2idx = {token: idx for idx, token in enumerate(deprel_idx2token)}
```

[14]:
```
pos_idx2token = list(set([pos for sent in train_pos_1 + train_pos_2 for pos in␣
↪sent]))
pos_idx2token = [PAD, UNK] + pos_idx2token
pos_token2idx = {token: idx for idx, token in enumerate(pos_idx2token)}
```

[15]:
```
device = torch.device('cuda') if torch.cuda.is_available() else torch.
↪device('cpu')
```

Convert text to ids, transform to pytorch tensors and pad to equal lengths.

[16]:
```
def texts2ids(texts, token2idx, device):
    text_ids = []
    for text in texts:
        text_ids.append(torch.tensor([token2idx.get(token, UNK_ID) for token in␣
↪text], dtype=torch.long))
    text_ids = pad_sequence(text_ids, batch_first=True, padding_value=PAD_ID)
    return text_ids.to(device)
```

[17]:
```
train_sents_ids = texts2ids(train_sents, token2idx, device)
train_deprels_ids = texts2ids(train_deprels, deprel_token2idx, device)
train_pos_1_ids = texts2ids(train_pos_1, pos_token2idx, device)
train_pos_2_ids = texts2ids(train_pos_2, pos_token2idx, device)

test_sents_ids = texts2ids(test_sents, token2idx, device)
test_deprels_ids = texts2ids(test_deprels, deprel_token2idx, device)
test_pos_1_ids = texts2ids(test_pos_1, pos_token2idx, device)
test_pos_2_ids = texts2ids(test_pos_2, pos_token2idx, device)
```

[18]:
```
train_rels = torch.tensor(train_rels, dtype=torch.long, device=device)
test_rels = torch.tensor(test_rels, dtype=torch.long, device=device)
```

Create a Dataset.

[19]:
```
train_dataset = TensorDataset(train_sents_ids, train_deprels_ids,␣
 ↪train_pos_1_ids, train_pos_2_ids, train_rels)
test_dataset = TensorDataset(test_sents_ids, test_deprels_ids, test_pos_1_ids,␣
 ↪test_pos_2_ids, test_rels)
```

Randomly split the train set into 7000 sentences for train and 1000 for validation.

[20]:
```
train_dataset, val_dataset = random_split(train_dataset, [7000, 1000],␣
 ↪generator=torch.Generator().manual_seed(42))
```

[21]:
```
train_dataloader = DataLoader(train_dataset, batch_size=100, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=100)
test_dataloader = DataLoader(test_dataset, batch_size=100)
```

0.2.1 Task 3. Modify the model. (2 points)

Add three more embedding layers that are randomly initialized. First one will encode the deprels, second encodes first relative positions and third encodes second relative position. The input size to each embedding layer is the size of corresponding vocabulary. The embedding size is specified by `deprel_emb_dim` argument.

Modify the `forward()` method. Encode each input with the corresponding embedding layer. After that, concatenate all the embeddings together.

[31]:
```
class RelationClassificationModel(nn.Module):
    def __init__(self, pretrained_emb, deprel_idx2token, pos_idx2token,␣
 ↪deprel_emb_dim, num_filters, kernel_sizes, num_classes):
        super().__init__()
        self.word_emb = nn.Embedding.from_pretrained(pretrained_emb,␣
 ↪padding_idx=PAD_ID, freeze=False)

        ### YOUR CODE STARTS HERE
        # Embedding layer for the deprels
```

```
        self.deprel_emb = nn.Embedding(len(deprel_idx2token), deprel_emb_dim,␣
↪padding_idx=PAD_ID)
        # Embedding layer for the first positional embeddings
        self.pos_emb_1 = nn.Embedding(len(pos_idx2token), deprel_emb_dim,␣
↪padding_idx=PAD_ID)
        # Embedding layer for the second positional embeddings
        self.pos_emb_2 = nn.Embedding(len(pos_idx2token), deprel_emb_dim,␣
↪padding_idx=PAD_ID)
        # Concatenated embedding size
        emb_size = self.word_emb.weight.size(1) + (deprel_emb_dim *3)
        ### YOUR CODE ENDS HERE

        self.convs = nn.ModuleList(
            [
                    nn.Conv1d(in_channels=emb_size, out_channels=num_filters,␣
↪kernel_size=kernel_size)
                    for kernel_size in kernel_sizes
            ]
        )
        linear_dim = num_filters * len(kernel_sizes)
        self.linear_out = nn.Sequential(nn.Linear(in_features=linear_dim,␣
↪out_features=linear_dim // 2),
                                        nn.LeakyReLU(),
                                        nn.Dropout(0.5),
                                        nn.Linear(linear_dim // 2, num_classes))
        self.drop = nn.Dropout(0.5)

    def forward(self, sent, deprels, pos_1, pos_2):
        # x size is [batch x seq_len]

        x_word = self.word_emb(sent) # [batch x seq_len x emb_dim]

        x_dep = self.deprel_emb(deprels) # [batch x seq_len x deprel_emb_dim]
        x_pos_1 = self.pos_emb_1(pos_1) # [batch x seq_len x deprel_emb_dim]
        x_pos_2 = self.pos_emb_2(pos_2) # [batch x seq_len x deprel_emb_dim]
        # Concatenate the embeddings above
        # Hint: Use torch.cat
        ### YOUR CODE STARTS HERE
        x = torch.cat((x_word,x_dep,x_pos_1,x_pos_2),dim=2)
        ### YOUR CODE ENDS HERE

        x = x.permute(0, 2, 1) # [batch x emb_dim x seq_len]
        xs = [torch.relu(conv(x)) for conv in self.convs] # [batch x␣
↪num_filters x conv_seq_len] x num_kernels
        xs = [torch.nn.functional.max_pool1d(x, x.size(2)).squeeze(2) for x in␣
↪xs] # [batch x num_filters] x num_kernels
```

```
        x = torch.cat(xs, dim=1) # [batch x num_filters * num_kernels]
        x = self.drop(x)
        x = self.linear_out(x) # [batch x num_classes]
        return x
```

[32]:
```
num_filters = 150
deprel_emb_dim = 50
kernel_sizes = [2, 3, 4, 5]
lr = 1e-3
num_classes = dataset['train'].features['relation'].num_classes

num_iters = 100
```

[33]:
```
model = RelationClassificationModel(glove_vecs, deprel_idx2token,␣
↪pos_idx2token, deprel_emb_dim, num_filters, kernel_sizes, num_classes)
model = model.to(device)
```

[34]:
```
print(model)
```

```
RelationClassificationModel(
  (word_emb): Embedding(400004, 300, padding_idx=0)
  (deprel_emb): Embedding(46, 50, padding_idx=0)
  (pos_emb_1): Embedding(164, 50, padding_idx=0)
  (pos_emb_2): Embedding(164, 50, padding_idx=0)
  (convs): ModuleList(
    (0): Conv1d(450, 150, kernel_size=(2,), stride=(1,))
    (1): Conv1d(450, 150, kernel_size=(3,), stride=(1,))
    (2): Conv1d(450, 150, kernel_size=(4,), stride=(1,))
    (3): Conv1d(450, 150, kernel_size=(5,), stride=(1,))
  )
  (linear_out): Sequential(
    (0): Linear(in_features=600, out_features=300, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=300, out_features=19, bias=True)
  )
  (drop): Dropout(p=0.5, inplace=False)
)
```

[35]:
```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

Train the model.

[36]:
```
best_f1 = 0.0
for i in range(num_iters):
    current_loss = 0
    model.train()
```

```
    for texts, deprels, pos_1, pos_2, labels in train_dataloader:
        model.zero_grad()
        preds = model(texts, deprels, pos_1, pos_2)
        loss = loss_fn(preds, labels)
        current_loss += loss
        loss.backward()
        optimizer.step()
    avg_train_loss = current_loss.item() / len(train_dataloader)

    current_loss = 0
    current_f1 = 0
    all_preds = []
    all_labels = []
    model.eval()
    for texts, deprels, pos_1, pos_2, labels in val_dataloader:
        with torch.no_grad():
            preds = model(texts, deprels, pos_1, pos_2)
            loss = loss_fn(preds, labels)
            preds = torch.argmax(torch.log_softmax(preds, dim=1), dim=1)
            all_preds.append(preds)
            all_labels.append(labels)
            current_loss += loss
    avg_val_loss = current_loss.item() / len(val_dataloader)
    avg_val_f1 = f1_score(torch.hstack(all_preds), torch.hstack(all_labels),␣
↪average='macro', num_classes=num_classes)

    if avg_val_f1 > best_f1:
        print(f'F1 increased [{best_f1:.4f} --> {avg_val_f1:.4f}]. Saving the␣
↪model...')
        best_f1 = avg_val_f1
        torch.save(model, 'model_best.pt')

    print(f'Epoch: {i}\tTrain loss: {avg_train_loss:.4f}\tVal loss:␣
↪{avg_val_loss:.4f}\tVal acc: {avg_val_f1:.4f}')
```

```
F1 increased [0.0000 --> 0.3226]. Saving the model…
Epoch: 0        Train loss: 2.3667      Val loss: 1.7322        Val acc: 0.3226
F1 increased [0.3226 --> 0.4462]. Saving the model…
Epoch: 1        Train loss: 1.6366      Val loss: 1.3620        Val acc: 0.4462
F1 increased [0.4462 --> 0.5082]. Saving the model…
Epoch: 2        Train loss: 1.3473      Val loss: 1.2123        Val acc: 0.5082
F1 increased [0.5082 --> 0.5703]. Saving the model…
Epoch: 3        Train loss: 1.1178      Val loss: 1.1125        Val acc: 0.5703
Epoch: 4        Train loss: 0.9469      Val loss: 1.0983        Val acc: 0.5699
F1 increased [0.5703 --> 0.6150]. Saving the model…
Epoch: 5        Train loss: 0.7827      Val loss: 1.0533        Val acc: 0.6150
Epoch: 6        Train loss: 0.6585      Val loss: 1.0843        Val acc: 0.6137
```

```
Epoch: 7        Train loss: 0.5215      Val loss: 1.1394        Val acc: 0.6034
Epoch: 8        Train loss: 0.4093      Val loss: 1.1860        Val acc: 0.6108
Epoch: 9        Train loss: 0.3173      Val loss: 1.3324        Val acc: 0.6101
Epoch: 10       Train loss: 0.2657      Val loss: 1.4332        Val acc: 0.5901
F1 increased [0.6150 --> 0.6252]. Saving the model…
Epoch: 11       Train loss: 0.2229      Val loss: 1.4068        Val acc: 0.6252
F1 increased [0.6252 --> 0.6355]. Saving the model…
Epoch: 12       Train loss: 0.1831      Val loss: 1.4850        Val acc: 0.6355
Epoch: 13       Train loss: 0.1495      Val loss: 1.6304        Val acc: 0.6198
Epoch: 14       Train loss: 0.1259      Val loss: 1.6886        Val acc: 0.6257
Epoch: 15       Train loss: 0.1154      Val loss: 1.6236        Val acc: 0.6308
Epoch: 16       Train loss: 0.1022      Val loss: 1.7499        Val acc: 0.6279
Epoch: 17       Train loss: 0.1028      Val loss: 1.8690        Val acc: 0.6124
Epoch: 18       Train loss: 0.0982      Val loss: 1.8508        Val acc: 0.6122
Epoch: 19       Train loss: 0.0763      Val loss: 1.8443        Val acc: 0.6297
Epoch: 20       Train loss: 0.0612      Val loss: 2.0054        Val acc: 0.6237
F1 increased [0.6355 --> 0.6383]. Saving the model…
Epoch: 21       Train loss: 0.0668      Val loss: 2.0351        Val acc: 0.6383
F1 increased [0.6383 --> 0.6423]. Saving the model…
Epoch: 22       Train loss: 0.0630      Val loss: 1.9774        Val acc: 0.6423
Epoch: 23       Train loss: 0.0651      Val loss: 2.1872        Val acc: 0.6245
Epoch: 24       Train loss: 0.0623      Val loss: 2.1309        Val acc: 0.6340
Epoch: 25       Train loss: 0.0550      Val loss: 2.0638        Val acc: 0.6177
Epoch: 26       Train loss: 0.0503      Val loss: 2.3110        Val acc: 0.6149
Epoch: 27       Train loss: 0.0568      Val loss: 2.2987        Val acc: 0.6166
Epoch: 28       Train loss: 0.0482      Val loss: 2.3122        Val acc: 0.6206
Epoch: 29       Train loss: 0.0486      Val loss: 2.2730        Val acc: 0.6267
Epoch: 30       Train loss: 0.0533      Val loss: 2.3106        Val acc: 0.6206
Epoch: 31       Train loss: 0.0548      Val loss: 2.3974        Val acc: 0.6065
Epoch: 32       Train loss: 0.0538      Val loss: 2.4853        Val acc: 0.6088
Epoch: 33       Train loss: 0.0481      Val loss: 2.4573        Val acc: 0.6252
Epoch: 34       Train loss: 0.0490      Val loss: 2.4734        Val acc: 0.6175
Epoch: 35       Train loss: 0.0500      Val loss: 2.5090        Val acc: 0.6271
Epoch: 36       Train loss: 0.0490      Val loss: 2.5106        Val acc: 0.6230
Epoch: 37       Train loss: 0.0368      Val loss: 2.5919        Val acc: 0.6088
Epoch: 38       Train loss: 0.0357      Val loss: 2.6418        Val acc: 0.6269
Epoch: 39       Train loss: 0.0396      Val loss: 2.8978        Val acc: 0.6047
Epoch: 40       Train loss: 0.0331      Val loss: 2.7691        Val acc: 0.6046
Epoch: 41       Train loss: 0.0320      Val loss: 2.7307        Val acc: 0.6108
Epoch: 42       Train loss: 0.0431      Val loss: 2.7075        Val acc: 0.6227
Epoch: 43       Train loss: 0.0346      Val loss: 2.7469        Val acc: 0.6251
Epoch: 44       Train loss: 0.0313      Val loss: 2.7977        Val acc: 0.6130
Epoch: 45       Train loss: 0.0367      Val loss: 2.8091        Val acc: 0.6217
Epoch: 46       Train loss: 0.0409      Val loss: 2.8448        Val acc: 0.6117
Epoch: 47       Train loss: 0.0285      Val loss: 2.9904        Val acc: 0.6178
Epoch: 48       Train loss: 0.0386      Val loss: 3.0929        Val acc: 0.6135
Epoch: 49       Train loss: 0.0419      Val loss: 3.0872        Val acc: 0.6081
Epoch: 50       Train loss: 0.0265      Val loss: 3.1152        Val acc: 0.6215
```

```
Epoch: 51	Train loss: 0.0291	Val loss: 3.2350	Val acc: 0.6227
Epoch: 52	Train loss: 0.0364	Val loss: 3.0413	Val acc: 0.6277
Epoch: 53	Train loss: 0.0383	Val loss: 3.0064	Val acc: 0.6204
Epoch: 54	Train loss: 0.0408	Val loss: 3.0481	Val acc: 0.6201
Epoch: 55	Train loss: 0.0423	Val loss: 2.9397	Val acc: 0.6033
Epoch: 56	Train loss: 0.0419	Val loss: 3.1233	Val acc: 0.6129
Epoch: 57	Train loss: 0.0259	Val loss: 3.3820	Val acc: 0.6156
Epoch: 58	Train loss: 0.0308	Val loss: 3.3622	Val acc: 0.6118
Epoch: 59	Train loss: 0.0345	Val loss: 3.2145	Val acc: 0.6217
Epoch: 60	Train loss: 0.0270	Val loss: 3.2716	Val acc: 0.6305
Epoch: 61	Train loss: 0.0341	Val loss: 3.2785	Val acc: 0.6128
Epoch: 62	Train loss: 0.0321	Val loss: 3.3106	Val acc: 0.6235
Epoch: 63	Train loss: 0.0256	Val loss: 3.4519	Val acc: 0.6067
Epoch: 64	Train loss: 0.0278	Val loss: 3.3679	Val acc: 0.6252
Epoch: 65	Train loss: 0.0312	Val loss: 3.3292	Val acc: 0.6047
Epoch: 66	Train loss: 0.0253	Val loss: 3.2519	Val acc: 0.6322
Epoch: 67	Train loss: 0.0280	Val loss: 3.3405	Val acc: 0.6217
F1 increased [0.6423 --> 0.6440]. Saving the model…
Epoch: 68	Train loss: 0.0267	Val loss: 3.4262	Val acc: 0.6440
Epoch: 69	Train loss: 0.0228	Val loss: 3.4212	Val acc: 0.6152
Epoch: 70	Train loss: 0.0214	Val loss: 3.5861	Val acc: 0.6290
Epoch: 71	Train loss: 0.0359	Val loss: 3.5163	Val acc: 0.6288
Epoch: 72	Train loss: 0.0454	Val loss: 3.4489	Val acc: 0.6373
Epoch: 73	Train loss: 0.0317	Val loss: 3.4768	Val acc: 0.6187
Epoch: 74	Train loss: 0.0224	Val loss: 3.6582	Val acc: 0.6146
Epoch: 75	Train loss: 0.0274	Val loss: 3.6070	Val acc: 0.6084
Epoch: 76	Train loss: 0.0345	Val loss: 3.5943	Val acc: 0.6196
Epoch: 77	Train loss: 0.0294	Val loss: 3.5629	Val acc: 0.6119
Epoch: 78	Train loss: 0.0301	Val loss: 3.4640	Val acc: 0.6204
Epoch: 79	Train loss: 0.0251	Val loss: 3.8435	Val acc: 0.6175
Epoch: 80	Train loss: 0.0259	Val loss: 4.0090	Val acc: 0.6224
Epoch: 81	Train loss: 0.0289	Val loss: 3.8883	Val acc: 0.6132
Epoch: 82	Train loss: 0.0300	Val loss: 3.8476	Val acc: 0.6205
Epoch: 83	Train loss: 0.0335	Val loss: 3.7654	Val acc: 0.6114
Epoch: 84	Train loss: 0.0235	Val loss: 3.9045	Val acc: 0.6256
Epoch: 85	Train loss: 0.0291	Val loss: 3.9680	Val acc: 0.6184
Epoch: 86	Train loss: 0.0267	Val loss: 3.8183	Val acc: 0.6130
Epoch: 87	Train loss: 0.0223	Val loss: 4.0031	Val acc: 0.6315
Epoch: 88	Train loss: 0.0303	Val loss: 3.9609	Val acc: 0.6308
Epoch: 89	Train loss: 0.0248	Val loss: 4.0004	Val acc: 0.6164
Epoch: 90	Train loss: 0.0215	Val loss: 4.0533	Val acc: 0.6163
Epoch: 91	Train loss: 0.0242	Val loss: 4.1823	Val acc: 0.6195
Epoch: 92	Train loss: 0.0293	Val loss: 4.0160	Val acc: 0.6202
Epoch: 93	Train loss: 0.0305	Val loss: 4.0792	Val acc: 0.6130
Epoch: 94	Train loss: 0.0252	Val loss: 4.0926	Val acc: 0.6038
Epoch: 95	Train loss: 0.0186	Val loss: 4.1293	Val acc: 0.6087
Epoch: 96	Train loss: 0.0185	Val loss: 4.1227	Val acc: 0.6039
Epoch: 97	Train loss: 0.0270	Val loss: 4.1515	Val acc: 0.5837
```

```
Epoch: 98       Train loss: 0.0201      Val loss: 4.3088        Val acc: 0.6053
Epoch: 99       Train loss: 0.0219      Val loss: 4.3876        Val acc: 0.6026
```

Load the best model.

[37]:
```python
model = torch.load('model_best.pt')
model = model.to(device)
```

Test the model.

[38]:
```python
current_loss = 0
current_f1 = 0
all_preds = []
all_labels = []
model.eval()
for texts, deprels, pos_1, pos_2, labels in test_dataloader:
    with torch.no_grad():
        preds = model(texts, deprels, pos_1, pos_2)
        loss = loss_fn(preds, labels)
        preds = torch.argmax(torch.log_softmax(preds, dim=1), dim=1)
        all_preds.append(preds)
        all_labels.append(labels)
        current_loss += loss
avg_test_loss = current_loss.item() / len(test_dataloader)
all_preds = torch.hstack(all_preds)
all_labels = torch.hstack(all_labels)
avg_test_f1 = f1_score(all_preds, all_labels, average='macro',␣
 ↪num_classes=num_classes)

print(f'Test loss: {avg_test_loss:.4f}\tTest F1: {avg_test_f1:.4f}')
```

```
Test loss: 3.3558       Test F1: 0.5950
```

0.2.2 Task 4. Interpret the results. (0.5 points)

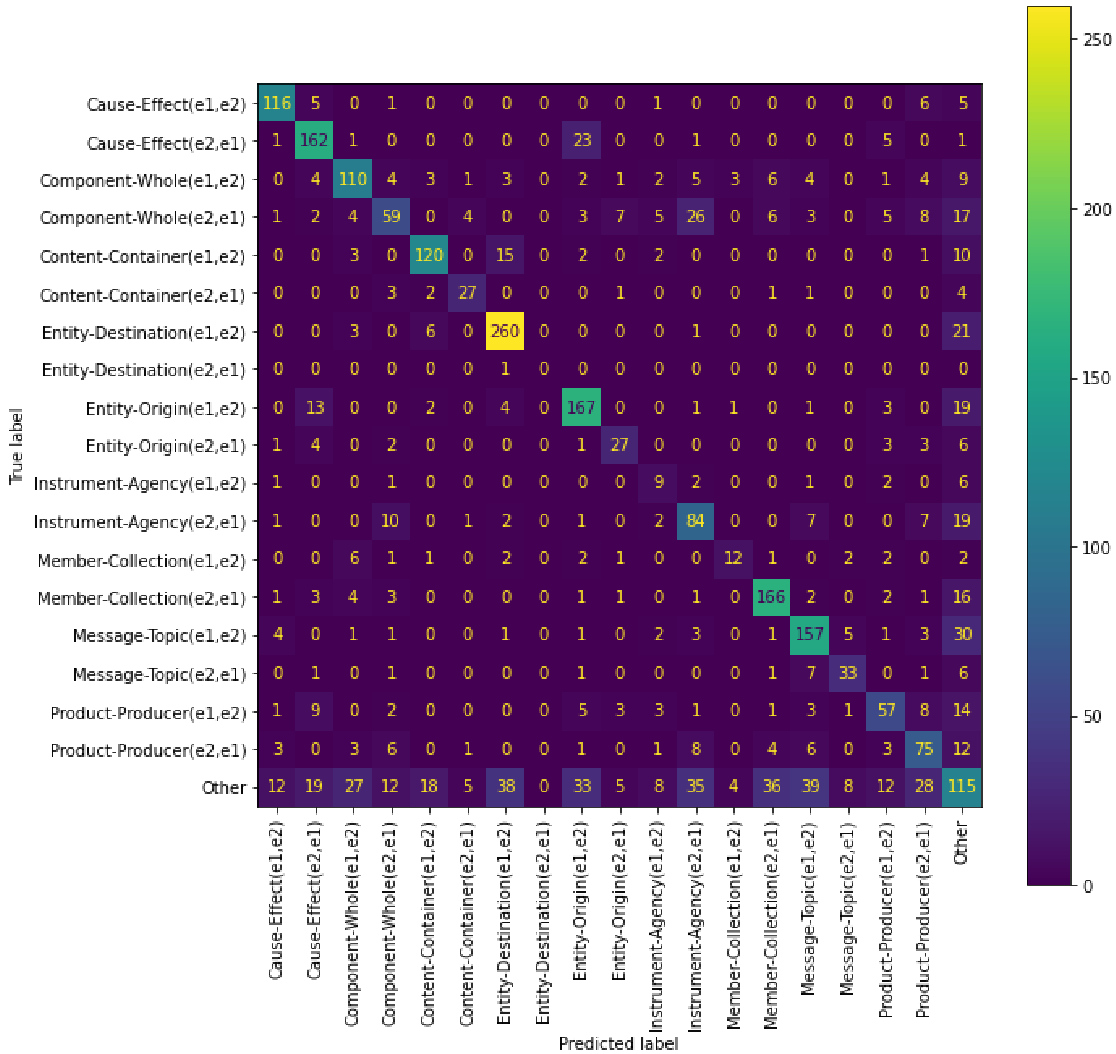
Look at the confusion matrix. Briefly describe what tendencies do you see. Which class got confused the most? What can you do to make the performance better?

YOUR ANSWER HERE:

(A) : The `Entity-Detination` has best performane. There is high posibility that model prodict labe as `other`. Maybe can optimize the label to increase the accuracy of prediction.

[39]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
fig, ax = plt.subplots(figsize=(10, 10))
cm = confusion_matrix(all_labels.cpu().numpy(), all_preds.cpu().numpy())
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=dataset['train'].
 ↪features['relation'].names)
disp.plot(xticks_rotation='vertical', ax=ax)
```

```
plt.show();
```



True label
Predicted label
Cause-Effect(e1,e2)
Cause-Effect(e2,e1)
Component-Whole(e1,e2)
Component-Whole(e2,e1)
Content-Container(e1,e2)
Content-Container(e2,e1)
Entity-Destination(e1,e2)
Entity-Destination(e2,e1)
Entity-Origin(e1,e2)
Entity-Origin(e2,e1)
Instrument-Agency(e1,e2)
Instrument-Agency(e2,e1)
Member-Collection(e1,e2)
Member-Collection(e2,e1)
Message-Topic(e1,e2)
Message-Topic(e2,e1)
Product-Producer(e1,e2)
Product-Producer(e2,e1)
Other
250
200
150
100
50
0


```
[ ]:
```