train_test_create

June 15, 2021

[1]:
```
import numpy as np
import pandas as pd
import re, os
from pathlib import Path
```

[2]:
```
pdf = pd.read_csv('order_phrases.csv')
```

[3]:
```
pdf.head()
```

[3]:

	spoken_text	split	prod_start_char	prod_end_char	qty	qty_char	c_name
0	i want 1 cappuccino from CCD	train	10	19	1	8	cappuccino
1	I want 2 flat white from CCD	train	10	19	2	8	flat white
2	dark frappe one	train	1	11	one	13	dark frappe
3	Give me espresso 3	train	9	16	3	18	espresso
4	My order is classic filter coffee 45	train	13	26	45	35	classic filter

[4]:
```
train_data = pdf[pdf["split"]=='train']
test_data = pdf[pdf["split"]=='test']
```

[5]:
```
train_data.iloc[0]['prod_start_char']
```

[5]:
```
10
```

[6]:
```
all_text = train_data['spoken_text'].to_list()
all_unseen_text = test_data['spoken_text'].to_list()
```

[7]:
```
all_text[0]
```

[7]:
```
'i want 1 cappuccino from CCD'
```

[8]:
```
TRAIN_DATA = []
TEST_DATA = []
```

[9]:
```
for i in range(len(all_text)):
    data=[0]*2
    data[0] = all_text[i]
    ent_dict = {}
    dict_data_1 = [train_data.iloc[i]['prod_start_char'], train_data.
 ↪iloc[i]['prod_end_char'], "PRODUCT"]
    dict_data_2 = [train_data.iloc[i]['qty_char'], (int(train_data.
 ↪iloc[i]['qty_char'])+(len(str(train_data.iloc[i]['qty']))-1)), "NUMBER"]
    ent_dict["entities"] = [tuple(dict_data_1), tuple(dict_data_2)]
    data[1] = ent_dict
    TRAIN_DATA.append(tuple(data))
TRAIN_DATA
```

[9]:
```
[('i want 1 cappuccino from CCD',
  {'entities': [(10, 19, 'PRODUCT'), (8, 8, 'NUMBER')]}),
 ('I want 2 flat white from CCD',
  {'entities': [(10, 19, 'PRODUCT'), (8, 8, 'NUMBER')]}),
 ('dark frappe one', {'entities': [(1, 11, 'PRODUCT'), (13, 15, 'NUMBER')]}),
 ('Give me espresso 3',
  {'entities': [(9, 16, 'PRODUCT'), (18, 18, 'NUMBER')]}),
 ('My order is classic filter coffee 45',
  {'entities': [(13, 26, 'PRODUCT'), (35, 36, 'NUMBER')]}),
 ('Bring macchiato 9 large',
  {'entities': [(7, 15, 'PRODUCT'), (17, 17, 'NUMBER')]}),
 ('cafe americano two please',
  {'entities': [(1, 14, 'PRODUCT'), (16, 18, 'NUMBER')]}),
 ('cafe mocha 8 for me',
  {'entities': [(1, 10, 'PRODUCT'), (12, 12, 'NUMBER')]}),
 ('aztec coffee 200 cups for me',
  {'entities': [(1, 5, 'PRODUCT'), (14, 16, 'NUMBER')]}),
 ('1 large ethiopian coffee for me',
  {'entities': [(11, 19, 'PRODUCT'), (1, 1, 'NUMBER')]}),
 ('give me 2 cafe latte',
  {'entities': [(11, 20, 'PRODUCT'), (9, 9, 'NUMBER')]}),
 ('make 11 coconut milk latte for me',
  {'entities': [(9, 26, 'PRODUCT'), (6, 7, 'NUMBER')]}),
 ('do you have 7 toffee cappuccino?',
  {'entities': [(15, 31, 'PRODUCT'), (13, 13, 'NUMBER')]}),
 ('I would like 23 vanilla cappuccino',
  {'entities': [(17, 34, 'PRODUCT'), (14, 15, 'NUMBER')]}),
 ('17 vanilla latte for me please',
  {'entities': [(4, 16, 'PRODUCT'), (1, 2, 'NUMBER')]}),
 ('get 54 toffee latte right now',
  {'entities': [(8, 19, 'PRODUCT'), (5, 6, 'NUMBER')]})]
```

[10]:
```
for i in range(len(all_unseen_text)):
    data=[0]*2
    data[0] = all_unseen_text[i]
    ent_dict = {}
    dict_data_1 = [test_data.iloc[i]['prod_start_char'], test_data.
 ↪iloc[i]['prod_end_char'], "PRODUCT"]
    dict_data_2 = [test_data.iloc[i]['qty_char'], int(test_data.
 ↪iloc[i]['qty_char'])+(len(str(test_data.iloc[i]['qty']).strip())-1),␣
 ↪"NUMBER"]
    ent_dict["entities"] = [tuple(dict_data_1), tuple(dict_data_2)]
    data[1] = ent_dict
    TEST_DATA.append(tuple(data))
TEST_DATA
```

[10]:
```
[('100 aztec please', {'entities': [(5, 9, 'PRODUCT'), (1, 3, 'NUMBER')]}),
 ('30 mocha for me', {'entities': [(4, 8, 'PRODUCT'), (1, 2, 'NUMBER')]}),
 ('get me three frappes',
  {'entities': [(14, 19, 'PRODUCT'), (8, 12, 'NUMBER')]}),
 ('Can I have four espresso please?',
  {'entities': [(17, 24, 'PRODUCT'), (12, 15, 'NUMBER')]}),
 ('5 large cappuccino for me',
  {'entities': [(9, 18, 'PRODUCT'), (1, 1, 'NUMBER')]}),
 ('My order is 10 dark frappe',
  {'entities': [(16, 26, 'PRODUCT'), (13, 14, 'NUMBER')]}),
 ('Get me 22 coconut milk latte',
  {'entities': [(11, 28, 'PRODUCT'), (8, 9, 'NUMBER')]})]
```

0.0.1 Spacy 2 to 3 conversion of training data

[11]:
```
import spacy
from spacy import displacy
```

[12]:
```
import random
from spacy.util import minibatch, compounding
from spacy.training.example import Example
from spacy.tokens import Span
```

nlp = spacy.blank(“en”) nlp.add_pipe(“ner”)

ner=nlp.get_pipe(“ner”)

for _, annotations in TRAIN_DATA: for ent in annotations.get(“entities”): ner.add_label(ent[2])

[13]:
```
from tqdm import tqdm
from spacy.tokens import DocBin

nlp = spacy.blank("en")
```

```
db = DocBin() # create a DocBin object

for text, annot in tqdm(TRAIN_DATA): # data in previous format
    doc = nlp.make_doc(text) # create doc object from text
    ents = []
    for start, end, label in annot["entities"]: # add character indexes
        print (doc, start, end, label)
        span = doc.char_span(start, end, label=label, alignment_mode="expand")
        if span is None:
            print("Skipping entity "+label)
        else:
            ents.append(span)
            print(ents)
    doc.ents = ents # label the text with the ents
    db.add(doc)

db.to_disk("./train.spacy") # save the docbin object
```

```
100%|      | 16/16 [00:00<00:00, 454.97it/s]
```

```
i want 1 cappuccino from CCD 10 19 PRODUCT
[cappuccino]
i want 1 cappuccino from CCD 8 8 NUMBER
[cappuccino, ]
I want 2 flat white from CCD 10 19 PRODUCT
[flat white]
I want 2 flat white from CCD 8 8 NUMBER
[flat white, ]
dark frappe one 1 11 PRODUCT
[dark frappe]
dark frappe one 13 15 NUMBER
[dark frappe, one]
Give me espresso 3 9 16 PRODUCT
[espresso]
Give me espresso 3 18 18 NUMBER
Skipping entity NUMBER
My order is classic filter coffee 45 13 26 PRODUCT
[classic filter]
My order is classic filter coffee 45 35 36 NUMBER
[classic filter, 45]
Bring macchiato 9 large 7 15 PRODUCT
[macchiato]
Bring macchiato 9 large 17 17 NUMBER
[macchiato, ]
cafe americano two please 1 14 PRODUCT
[cafe americano]
cafe americano two please 16 18 NUMBER
[cafe americano, two]
```

```
cafe mocha 8 for me 1 10 PRODUCT
[cafe mocha]
cafe mocha 8 for me 12 12 NUMBER
[cafe mocha, ]
aztec coffee 200 cups for me 1 5 PRODUCT
[aztec]
aztec coffee 200 cups for me 14 16 NUMBER
[aztec, 200]
1 large ethiopian coffee for me 11 19 PRODUCT
[ethiopian coffee]
1 large ethiopian coffee for me 1 1 NUMBER
[ethiopian coffee, ]
give me 2 cafe latte 11 20 PRODUCT
[cafe latte]
give me 2 cafe latte 9 9 NUMBER
[cafe latte, ]
make 11 coconut milk latte for me 9 26 PRODUCT
[coconut milk latte]
make 11 coconut milk latte for me 6 7 NUMBER
[coconut milk latte, 11]
do you have 7 toffee cappuccino? 15 31 PRODUCT
[toffee cappuccino]
do you have 7 toffee cappuccino? 13 13 NUMBER
[toffee cappuccino, ]
I would like 23 vanilla cappuccino 17 34 PRODUCT
[vanilla cappuccino]
I would like 23 vanilla cappuccino 14 15 NUMBER
[vanilla cappuccino, 23]
17 vanilla latte for me please 4 16 PRODUCT
[vanilla latte]
17 vanilla latte for me please 1 2 NUMBER
[vanilla latte, 17]
get 54 toffee latte right now 8 19 PRODUCT
[toffee latte]
get 54 toffee latte right now 5 6 NUMBER
[toffee latte, 54]
```

[14]:
```python
# load a new spacy model
db = DocBin() # create a DocBin object

for text, annot in tqdm(TEST_DATA): # data in previous format
    doc = nlp.make_doc(text) # create doc object from text
    ents = []
    for start, end, label in annot["entities"]: # add character indexes
        span = doc.char_span(start, end, label=label, alignment_mode="expand")
        if span is None:
```

```
            print("Skipping entity "+label)
        else:
            ents.append(span)
    doc.ents = ents # label the text with the ents
    db.add(doc)

db.to_disk("./test.spacy") # save the docbin object
```

```
100%|      | 7/7 [00:00<00:00, 621.30it/s]
```

0.1 To train, run:

python -m spacy train config.cfg –output ./output –paths.train ./train.spacy –paths.dev ./test.spacy

1 Only after you run the above command, check below output

[15]:
```
nlp1 = spacy.load(Path(os.getcwd()+"/output/model-best"))
```

[17]:
```
doc = nlp1("i want 11 cappuccino and 22 flat white from CCD") # input sample␣
↪text

displacy.render(doc, style="ent", jupyter=True) # display in Jupyter
```

```
<IPython.core.display.HTML object>
```

[19]:
```
doc = nlp1("I would like 23 machhiato")
displacy.render(doc, style='ent', jupyter=True)
```

```
<IPython.core.display.HTML object>
```

[ ]:
```

```

[ ]:
```

```

1.1 Below are some experimental tests

[20]:
```
nlp = spacy.load("en_core_web_lg", exclude=["ner", "tok2vec"])
nlp.add_pipe("ner", source=nlp1)
nlp.add_pipe("tok2vec", source=nlp1)
```

[20]:
```
<spacy.pipeline.tok2vec.Tok2Vec at 0x7f0aa5011680>
```

[27]:
```
ner=nlp.get_pipe("ner")
for _, annotations in TRAIN_DATA:
    for ent in annotations.get("entities"):
        ner.add_label(ent[2])
```

[30]:
```
def get_entity_options():
    col_dict = {}
    list_colours = ['#ffe119', '#3cb44b']
    for label, colour in zip(nlp.pipe_labels['ner'], list_colours):
        col_dict[label] = colour
    options = {"ents": nlp.pipe_labels['ner'], "colors": col_dict}
    return options
get_entity_options()
```

[30]:
```
{'ents': ['NUMBER', 'PRODUCT'],
 'colors': {'NUMBER': '#ffe119', 'PRODUCT': '#3cb44b'}}
```

[31]:
```
doc = nlp("I want 6 flat white")
print("Entities", [(ent.text, ent.label_) for ent in doc.ents])
```

```
Entities []
```

1.2 Somehow build a dependency graph and link the prodcut to its number, you may also check nearest node.

[32]:
```
numbers = [token for token in doc if token.like_num]
```

[33]:
```
for token in doc:
    print(token.text, token.dep_, token.head.text, token.head.pos_,
            [child for child in token.children])
```

```
I nsubj want ADP []
want nsubj 6 ADP [I]
6 nsubj flat ADP [want]
flat ROOT flat ADP [6]
white ROOT white ADP []
```

pipes_to_disable

nlang = spacy.load(“en_core_web_sm”)

ner=nlang.get_pipe(“ner”)

pipes_disable_except = [“ner”, “trf_wordpiecer”, “trf_tok2vec”] pipes_to_disable = [pipe for pipe in nlang.pipe_names if pipe not in pipes_disable_except]

with nlang.disable_pipes(*pipes_to_disable): for i in range(40): random.shuffle(TRAIN_DATA) losses = {} for batch in minibatch(TRAIN_DATA, size=2): for text, annotations in batch: doc =

nlang.make_doc(text) example = Example.from_dict(doc, annotations) nlang.update([example], drop=0.5, # drop half losses=losses,) print("Loss:", losses)

ner=nlp.get_pipe("ner") for _, annotations in TRAIN_DATA: for ent in annotations.get("entities"): print(ent[2]) ner.add_label(ent[2])

[ ]:

examples = [] for text, annots in TRAIN_DATA: spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), annots["entities"]) examples.append(Example.from_dict(nlp.make_doc(text), annots))

nlp.initialize(lambda: examples)

losses = {} for i in range(20): random.shuffle(examples) for batch in minibatch(examples, size=2): nlp.update(batch, drop=0.5, # dropout - make it harder to memorise data losses=losses,) print("Losses", losses)