PROJECT DOCUMENTATION

CODEBASE

EXPLORATORY DATA ANALYSIS

Tuning

Waiting for W&B process to finish, PID 15074  
Program ended successfully.

VBox(children=(Label(value=' 0.00MB of 0.00MB uploaded (0.00MB deduped)\r'), FloatProgress(value=1.0, max=1.0)…

Find user logs for this run at: /home/phil/Dropbox/Python/Paper To Data/paper-2-data/wandb/run-20210519\_113123-2p400pba/logs/debug.log

Find internal logs for this run at: /home/phil/Dropbox/Python/Paper To Data/paper-2-data/wandb/run-20210519\_113123-2p400pba/logs/debug-internal.log

### Run summary:

|  |  |
| --- | --- |
| train/loss | 0.0145 |
| train/learning\_rate | 2e-05 |
| train/epoch | 8.0 |
| train/global\_step | 160 |
| \_runtime | 58 |
| \_timestamp | 1621420341 |
| \_step | 8 |
| train/train\_runtime | 57.3977 |
| train/train\_samples\_per\_second | 2.788 |
| train/total\_flos | 0 |

### Run history:

|  |  |
| --- | --- |
| train/loss | █▂▁▁▁▁▁▁▁▁ |
| train/learning\_rate | ▁▃▆██▇▆▅▃▂ |
| train/epoch | ▁▁▂▂▃▃▃▃▄▄▅▅▆▆▆▆▇▇████ |
| train/global\_step | ▁▁▂▂▃▃▃▃▄▄▅▅▆▆▆▆▇▇████ |
| \_runtime | ▁▁▂▂▂▃▃▃▄▄▅▅▆▆▆▆▇▇████ |
| \_timestamp | ▁▁▂▂▂▃▃▃▄▄▅▅▆▆▆▆▇▇████ |
| \_step | ▁▁▂▂▂▃▃▃▄▄▄▅▅▅▆▆▆▇▇▇██ |
| eval/loss | █▂▂▂▁▂▁▁▂▂▁ |
| eval/precision | ▁▇▇▇█▇█████ |
| eval/recall | ▁▇█▇▇▇█████ |
| eval/f1 | ▁▇█▇█▇█████ |
| eval/accuracy | ▁█▇▇▇▇█████ |
| eval/runtime | ▂▄█▅▃▁▄▃▄▆▅ |
| eval/samples\_per\_second | ▇▅▁▄▆█▅▆▅▃▄ |
| train/train\_runtime | ▁ |
| train/train\_samples\_per\_second | ▁ |
| train/total\_flos | ▁ |

CODEBASE

## Paper 2 Data Workflow for Data Extraction - CUADv1 - Prepare Dataset

import re, json, os, itertools

import numpy as np

import pandas as pd

from tqdm import tqdm

import matplotlib.pyplot as plt

import seaborn as sns

import spacy

from spacy.lang.en import English

from spacy.training import offsets\_to\_biluo\_tags # requires spaCy 3.0

# download transformer model for spaCy if required

# !python -m spacy download en\_core\_web\_md

!python -m spacy validate # Ensure minimum v3.0.0

### 1. File handling - CUADv1

MASTER\_PATH = "../CUAD-v1/"

JSONL\_FILE = 'project\_6\_dataset.jsonl'

JSONL\_FILE\_INS = 'project\_7\_dataset.jsonl'

FEATURE\_CLASS\_LABELS = "feature\_class\_labels.json"

DATA\_FILE = 'cuad-v1-annotated.json'

### 2. Text Data Preprocessing - CUADv1 - Continued

#### Using Doccano to tag the text file dataset:

- Install doccano at the command line: pip install doccano

- At the command line change the directory to this directory

- run doccano at the command line by typing 'doccano'

- Application will be running at http://0.0.0.0:8000/

- Username is 'admin', passowrd is 'password'

- Use ctrl-c to end application

#### Prepare updated dataset for fine-tuning Transformers with HuggingFace

# JSONL is a multi-line json file and requires lines=True parameter

# Bring in both sets of annotations and concatenate vertically

df1 = pd.read\_json (JSONL\_FILE, lines=True)

df2 = pd.read\_json (JSONL\_FILE\_INS, lines=True)

df = pd.concat([df1, df2], axis=0)

df = df1 # Use this line to exclude the additional manually checked data

df = df.drop(['meta', 'annotation\_approver', 'comments'], axis=1)

df.head()

# Check the information and number of samples

df.info()

# Some samples were not annotated as they were not suitable samples.

# Eliminate any samples which were not annotated.

df\_cut = df[df['labels'].map(lambda d: len(d)) > 0].copy()

df\_cut.info()

# We tokenize each agreement prior to bringing into the transformer model

# Create tokens using spaCy

nlp = English()

df\_cut['tokens'] = df\_cut['text'].apply(lambda x: nlp(x))

df\_cut.head()

# Check an example of the text indices and labels

row = df\_cut.iloc[4]

doc = row['tokens']

for start, end, label in row['labels']:

print(start, end, label)

print("\n")

print(doc)

# Count and visualise the amount of labels

DOC\_NAME\_COUNT = 0

DATE\_COUNT = 0

PARTIES\_COUNT = 0

for index, row in df\_cut.iterrows():

for l in row['labels']:

if l[2] == "DOC\_NAME":

DOC\_NAME\_COUNT += 1

if l[2] == "AGMT\_DATE":

DATE\_COUNT += 1

if l[2] == "PARTY":

PARTIES\_COUNT += 1

# Create DataFrame for the bar plot

data=pd.DataFrame.from\_dict({"Document Name":[DOC\_NAME\_COUNT],

"Date of Agreement":[DATE\_COUNT],

"Parties":[PARTIES\_COUNT]})

# Use Seaborn for the bar plot

splot = sns.barplot(palette="pastel", data=data)

splot.set(title='Number of labels in dataset', ylabel='Count')

# Annotate the bars with the count of labels

for p in splot.patches:

splot.annotate(format(p.get\_height(), '.0f'),

(p.get\_x() + p.get\_width() / 2.,p.get\_height()),

ha = 'center', va = 'center',

size=10,

xytext = (0, -12),

textcoords = 'offset points')

# Show plot

plt.show

print("The total number of labels in the dataset is:", DOC\_NAME\_COUNT+DATE\_COUNT+PARTIES\_COUNT)

# Check how the entity labels match up with the tokens

ents=[]

for start, end, label in row['labels']:

if doc.char\_span(start, end, label) != None:

ent = doc.char\_span(start, end, label)

ents.append(ent)

elif doc.char\_span(start, end+1, label) != None:

ent = doc.char\_span(start, end+1, label)

ents.append(ent)

elif doc.char\_span(start+1, end, label) != None:

ent = doc.char\_span(start+1, end, label)

ents.append(ent)

elif doc.char\_span(start, end-1, label) != None:

ent = doc.char\_span(start, end-1, label)

ents.append(ent)

doc.ents = ents

doc.ents

# Each word must be seperated for the transformer using the IOB format

# Create tags using token.ent\_iob\_ and add to the DataFrame

# Allow for any character misalignment between spaCy tokenization and Doccano character indices

tags\_list\_iob = []

for index, row in df\_cut.iterrows():

doc = row['tokens']

ents=[]

for start, end, label in row['labels']:

if doc.char\_span(start, end, label) != None:

ent = doc.char\_span(start, end, label)

ents.append(ent)

elif doc.char\_span(start, end+1, label) != None:

ent = doc.char\_span(start, end+1, label)

ents.append(ent)

elif doc.char\_span(start+1, end, label) != None:

ent = doc.char\_span(start+1, end, label)

ents.append(ent)

elif doc.char\_span(start, end-1, label) != None:

ent = doc.char\_span(start, end-1, label)

ents.append(ent)

elif doc.char\_span(start-1, end, label) != None:

ent = doc.char\_span(start-1, end, label)

ents.append(ent)

doc.ents = ents

iob\_tags = [f"{t.ent\_iob\_}-{t.ent\_type\_}" if t.ent\_iob\_ != "O" else "O" for t in doc]

tags\_list\_iob.append(iob\_tags)

df\_cut['tags'] = tags\_list\_iob

# Check to ensure we have all the data (all non-null)

df\_cut.info()

# Generate list of the IOB feature class labels from tags

all\_tags = list(itertools.chain.from\_iterable(tags\_list\_iob))

def unique(list1):

# insert the list to the set

list\_set = set(list1)

# convert the set to the list

unique\_list = (list(list\_set))

unique\_list.sort()

return unique\_list

feature\_class\_labels = unique(all\_tags)

print(feature\_class\_labels)

# Generate the NER index tags for each token

df\_cut['ner\_tags'] = df\_cut['tags'].apply(lambda x: [feature\_class\_labels.index(tag) for tag in x])

# Split tokens into a list ready for CSV

df\_cut['split\_tokens'] = df\_cut['tokens'].apply(lambda x: [tok.text for tok in x])

# Check dataframe head

df\_cut.head()

# Export relevant columns only:

export\_columns = ['id', 'ner\_tags', 'split\_tokens']

export\_df = df\_cut[export\_columns]

export\_df.to\_json(DATA\_FILE, orient="table", index=False)

# Export Feature Class Labels for use in Transformer fine tuning

with open(FEATURE\_CLASS\_LABELS, 'w') as f:

json.dump(feature\_class\_labels, f, indent=2)

#### The dataset is now ready for any transformer model!