from datetime import datetime

from functools import reduce

import os

import pandas as pd

import re

from sklearn.metrics import average\_precision\_score, classification\_report, precision\_recall\_curve, PrecisionRecallDisplay

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

import matplotlib.pyplot as plt

import re

from twitter.cuad.representation.models.optimization import create\_optimizer

from twitter.cuad.representation.models.text\_encoder import TextEncoder

pd.set\_option('display.max\_colwidth', None)

pd.set\_option('display.expand\_frame\_repr', False)

print(tf.\_\_version\_\_)

print(tf.config.list\_physical\_devices())

log\_path = os.path.join('pnsfwtweettext\_model\_runs', datetime.now().strftime('%Y-%m-%d\_%H.%M.%S'))

tweet\_text\_feature = 'text'

params = {

'batch\_size': 32,

'max\_seq\_lengths': 256,

'model\_type': 'twitter\_bert\_base\_en\_uncased\_augmented\_mlm',

'trainable\_text\_encoder': True,

'lr': 5e-5,

'epochs': 10,

}

REGEX\_PATTERNS = [

r'^RT @[A-Za-z0-9\_]+: ',

r"@[A-Za-z0-9\_]+",

r'https:\/\/t\.co\/[A-Za-z0-9]{10}',

r'@\?\?\?\?\?',

]

EMOJI\_PATTERN = re.compile(

"(["

"\U0001F1E0-\U0001F1FF"

"\U0001F300-\U0001F5FF"

"\U0001F600-\U0001F64F"

"\U0001F680-\U0001F6FF"

"\U0001F700-\U0001F77F"

"\U0001F780-\U0001F7FF"

"\U0001F800-\U0001F8FF"

"\U0001F900-\U0001F9FF"

"\U0001FA00-\U0001FA6F"

"\U0001FA70-\U0001FAFF"

"\U00002702-\U000027B0"

"])"

)

def clean\_tweet(text):

for pattern in REGEX\_PATTERNS:

text = re.sub(pattern, '', text)

text = re.sub(EMOJI\_PATTERN, r' \1 ', text)

text = re.sub(r'\n', ' ', text)

return text.strip().lower()

df['processed\_text'] = df['text'].astype(str).map(clean\_tweet)

df.sample(10)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(df[['processed\_text']], df['is\_nsfw'], test\_size=0.1, random\_state=1)

def df\_to\_ds(X, y, shuffle=False):

ds = tf.data.Dataset.from\_tensor\_slices((

X.values,

tf.one\_hot(tf.cast(y.values, tf.int32), depth=2, axis=-1)

))

if shuffle:

ds = ds.shuffle(1000, seed=1, reshuffle\_each\_iteration=True)

return ds.map(lambda text, label: ({ tweet\_text\_feature: text }, label)).batch(params['batch\_size'])

ds\_train = df\_to\_ds(X\_train, y\_train, shuffle=True)

ds\_val = df\_to\_ds(X\_val, y\_val)

X\_train.values

inputs = tf.keras.layers.Input(shape=(), dtype=tf.string, name=tweet\_text\_feature)

encoder = TextEncoder(

max\_seq\_lengths=params['max\_seq\_lengths'],

model\_type=params['model\_type'],

trainable=params['trainable\_text\_encoder'],

local\_preprocessor\_path='demo-preprocessor'

)

embedding = encoder([inputs])["pooled\_output"]

predictions = tf.keras.layers.Dense(2, activation='softmax')(embedding)

model = tf.keras.models.Model(inputs=inputs, outputs=predictions)

model.summary()

optimizer = create\_optimizer(

params['lr'],

params['epochs'] \* len(ds\_train),

0,

weight\_decay\_rate=0.01,

optimizer\_type='adamw'

)

bce = tf.keras.losses.BinaryCrossentropy(from\_logits=False)

pr\_auc = tf.keras.metrics.AUC(curve='PR', num\_thresholds=1000, from\_logits=False)

model.compile(optimizer=optimizer, loss=bce, metrics=[pr\_auc])

callbacks = [

tf.keras.callbacks.EarlyStopping(

monitor='val\_loss',

mode='min',

patience=1,

restore\_best\_weights=True

),

tf.keras.callbacks.ModelCheckpoint(

filepath=os.path.join(log\_path, 'checkpoints', '{epoch:02d}'),

save\_freq='epoch'

),

tf.keras.callbacks.TensorBoard(

log\_dir=os.path.join(log\_path, 'scalars'),

update\_freq='batch',

write\_graph=False

)

]

history = model.fit(

ds\_train,

epochs=params['epochs'],

callbacks=callbacks,

validation\_data=ds\_val,

steps\_per\_epoch=len(ds\_train)

)

model.predict(["xxx 🍑"])

preds = X\_val.processed\_text.apply(apply\_model)

print(classification\_report(y\_val, preds >= 0.90, digits=4))

precision, recall, thresholds = precision\_recall\_curve(y\_val, preds)

fig = plt.figure(figsize=(15, 10))

plt.plot(precision, recall, lw=2)

plt.grid()

plt.xlim(0.2, 1)

plt.ylim(0.3, 1)

plt.xlabel("Recall", size=20)

plt.ylabel("Precision", size=20)

average\_precision\_score(y\_val, preds)