from .base import AbstractTrainer

from .utils import recalls\_and\_ndcgs\_for\_ks

from .utils import RETURN\_PREDICTION\_AND\_TRUE\_LABELS ##### My code 20201119

class BERTTrainer(AbstractTrainer):

print(' ')

print('meantime / trainers / bert.py / BERTTrainer')

def \_\_init\_\_(self, args, model, train\_loader, val\_loader, test\_loader, export\_root):

super().\_\_init\_\_(args, model, train\_loader, val\_loader, test\_loader, export\_root)

@classmethod

def code(cls):

return 'bert'

def add\_extra\_loggers(self):

pass

def log\_extra\_train\_info(self, log\_data):

pass

def calculate\_loss(self, batch):

print(' ')

print('meantime / trainers / bert.py / BERTTrainer / calculate\_loss')

# loss = self.model(batch, loss=True)

# loss = loss.mean()

# return loss

print('meantime / trainers / bert.py / BERTTrainer / calculate\_loss / d = self.model(batch)')

d = self.model(batch)

print(' ')

print('"d" in meantime / trainers / bert.py / BERTTrainer / calculate\_loss')

print(' ')

#print(d)

print("d['logits'].shape is")

print(' ')

#print(d['logits'].shape)

print(' ')

print("d['loss'].shape is")

print(' ')

#print(d['loss'].shape)

print(' ')

print("d['loss\_cnt'].shape is")

print(' ')

#print(d['loss\_cnt'].shape)

loss, loss\_cnt = d['loss'], d['loss\_cnt']

loss = (loss \* loss\_cnt).sum() / loss\_cnt.sum()

return loss

def calculate\_metrics(self, batch):

print(' ')

print('meantime / trainers / bert.py / calculate\_metrics')

print(' ')

labels = batch['labels']

scores = self.model(batch)['scores'] # B x C

# scores = scores.gather(1, candidates) # B x C

metrics = recalls\_and\_ndcgs\_for\_ks(scores, labels, self.metric\_ks)

return metrics

def NEW\_CODE\_PRINT\_PREDICTION(self, batch): ##### My code 20201119

print(' ')

print('meantime / trainers / bert.py / NEW\_CODE\_PRINT\_PREDICTION')

labels = batch['labels']

scores = self.model(batch)['scores'] # B x C

# scores = scores.gather(1, candidates) # B x C

metrics = RETURN\_PREDICTION\_AND\_TRUE\_LABELS(scores, labels, self.metric\_ks)

return metrics

def calculate\_loss2(self, batch):

print(' ')

print('meantime / trainers / bert.py / BERTTrainer / calculate\_loss')

# loss = self.model(batch, loss=True)

# loss = loss.mean()

# return loss

print('meantime / trainers / bert.py / BERTTrainer / calculate\_loss / d = self.model(batch)')

valid\_index, valid\_scores, valid\_labels = self.model(batch)

return valid\_index, valid\_scores, valid\_labels