import gc

import random

from clac\_metric import cv\_model\_evaluate

from utils import \*

from model import GATModel

from opt import Optimizer

tf.compat.v1.disable\_eager\_execution()

def PredictScore(train\_drug\_dis\_matrix, drug\_matrix, dis\_matrix, seed, epochs, emb\_dim, dp, lr, adjdp):

np.random.seed(seed)

tf.compat.v1.reset\_default\_graph()

tf.compat.v1.set\_random\_seed(seed)

adj = constructHNet(train\_drug\_dis\_matrix, drug\_matrix, dis\_matrix) # adj.shape=(867,867)

adj = sp.csr\_matrix(adj)

association\_nam = train\_drug\_dis\_matrix.sum()

X = constructNet(train\_drug\_dis\_matrix) # X.shape=(867,867)

features = sparse\_to\_tuple(sp.csr\_matrix(X)) # 稀疏矩阵转换为元组(特征矩阵)

num\_features = features[2][1]

features\_nonzero = features[1].shape[0]

adj\_orig = train\_drug\_dis\_matrix.copy()

adj\_orig = sparse\_to\_tuple(sp.csr\_matrix(adj\_orig))

adj\_norm = preprocess\_graph(adj)#邻接矩阵

adj\_nonzero = adj\_norm[1].shape[0]

placeholders = \

{

'features': tf.compat.v1.sparse\_placeholder(tf.compat.v1.float32),

'adj': tf.compat.v1.sparse\_placeholder(tf.compat.v1.float32),

'adj\_orig': tf.compat.v1.sparse\_placeholder(tf.compat.v1.float32),

'dropout': tf.compat.v1.placeholder\_with\_default(0., shape=()),

'adjdp': tf.compat.v1.placeholder\_with\_default(0., shape=())

}

model = GATModel(placeholders, num\_features, emb\_dim,

features\_nonzero, adj\_nonzero, train\_drug\_dis\_matrix.shape[0], name='LAGAT')

with tf.compat.v1.name\_scope('optimizer'):

opt = Optimizer(

preds=model.reconstructions,

labels=tf.compat.v1.reshape(tf.compat.v1.sparse\_tensor\_to\_dense(

placeholders['adj\_orig'], validate\_indices=False), [-1]),

model=model,

lr=lr, num\_u=train\_drug\_dis\_matrix.shape[0], num\_v=train\_drug\_dis\_matrix.shape[1],

association\_nam=association\_nam)

sess = tf.compat.v1.Session()

sess.run(tf.compat.v1.global\_variables\_initializer())

for epoch in range(epochs):

feed\_dict = dict()

feed\_dict.update({placeholders['features']: features})

feed\_dict.update({placeholders['adj']: adj\_norm})

feed\_dict.update({placeholders['adj\_orig']: adj\_orig})

feed\_dict.update({placeholders['dropout']: dp})

feed\_dict.update({placeholders['adjdp']: adjdp})

\_, avg\_cost = sess.run([opt.opt\_op, opt.cost], feed\_dict=feed\_dict)

if epoch % 100 == 0:

feed\_dict.update({placeholders['dropout']: 0})

feed\_dict.update({placeholders['adjdp']: 0})

res = sess.run(model.reconstructions, feed\_dict=feed\_dict)

print("Epoch:", '%04d' % (epoch + 1),

"train\_loss=", "{:.5f}".format(avg\_cost))

print('Optimization Finished!')

feed\_dict.update({placeholders['dropout']: 0})

feed\_dict.update({placeholders['adjdp']: 0})

res = sess.run(model.reconstructions, feed\_dict=feed\_dict)

sess.close()

return res

def cross\_validation\_experiment(drug\_dis\_matrix, drug\_matrix, dis\_matrix, seed, epochs, emb\_dim, dp, lr, adjdp):

index\_matrix = np.mat(np.where(drug\_dis\_matrix == 1))

association\_nam = index\_matrix.shape[1]

random\_index = index\_matrix.T.tolist()

random.seed(seed)

random.shuffle(random\_index)

k\_folds = 5

CV\_size = int(association\_nam / k\_folds)

temp = np.array(random\_index[:association\_nam - association\_nam %

k\_folds]).reshape(k\_folds, CV\_size, -1).tolist()

temp[k\_folds - 1] = temp[k\_folds - 1] + \

random\_index[association\_nam - association\_nam % k\_folds:]

random\_index = temp

metric = np.zeros((1, 7))

print("seed=%d, evaluating drug-disease...." % (seed))

for k in range(k\_folds):

print("------this is %dth cross validation------" % (k + 1))

train\_matrix = np.matrix(drug\_dis\_matrix, copy=True)

train\_matrix[tuple(np.array(random\_index[k]).T)] = 0

drug\_len = drug\_dis\_matrix.shape[0]

dis\_len = drug\_dis\_matrix.shape[1]

drug\_disease\_res = PredictScore(

train\_matrix, drug\_matrix, dis\_matrix, seed, epochs, emb\_dim, dp, lr, adjdp)

predict\_y\_proba = drug\_disease\_res.reshape(drug\_len, dis\_len)

metric\_tmp = cv\_model\_evaluate(

drug\_dis\_matrix, predict\_y\_proba, train\_matrix)

print(metric\_tmp)

metric += metric\_tmp

del train\_matrix

gc.collect()

print(metric / k\_folds)

metric = np.array(metric / k\_folds)

return metric

if \_\_name\_\_ == "\_\_main\_\_":

drug\_sim = np.loadtxt('../data/drug\_sim.csv', delimiter=',')

dis\_sim = np.loadtxt('../data/dis\_sim.csv', delimiter=',')

drug\_dis\_matrix = np.loadtxt('../data/drug\_dis.csv', delimiter=',')

epoch = 4000

emb\_dim = 64

lr = 0.01

adjdp = 0.6#原：0.6

dp = 0.4

simw = 6

result = np.zeros((1, 7), float)

average\_result = np.zeros((1, 7), float)

circle\_time = 1

for i in range(circle\_time):

result += cross\_validation\_experiment(

drug\_dis\_matrix, drug\_sim \* simw, dis\_sim \* simw, i, epoch, emb\_dim, dp, lr, adjdp)

average\_result = result / circle\_time

print(average\_result)

# 1.AUPR（Area Under the Precision-Recall Curve）

# 2.AUC（Area Under the ROC Curve）

# 3.F1-Score

# 4.准确率（Accuracy）

# 5.召回率（Recall）

# 6.特异度（Specificity）

# 7.精确率（Precision）