```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

GitHub Link:

https://github.com/genggeng88/dl4h_team61

To TA who is grading this notebook. We were originally registered with the paper of GD-RNN. However, we can't find the files for dataset generation in that paper. So, we switch to the paper "Prediction of drug-drug interaction events using graph neural networks based feature extraction (GNN-DDI)". (We already notified Sayantani who is the TA for our original paper.)

Introduction

- Background of the problem
  - Type of problem
    - Nowadays, many difficult diseases are treated using drug mixes (Polyparmacy), which is a good approach that utilizes the synergistic effects of drug interactions. However, unplanned DDIs could risk a patient's life because they may cause side effects or perhaps dangerous toxicity. The detection of DDIs becomes much more necessary, but diagnosing DDI on a large number of drug pairs, both in vitro and in vivo, is costly and time-consuming.
  - Importance/meaning of solving the problem
    - Detecting possible DDIs decreases the incidence of unexpected drug interactions and reduces drug production costs. It also can optimize the drug creation process.
  - Difficulty of the problem
    - DDI network can provide vital information about drugs interactions. Furthermore, using an attributed heterogeneous DDIs network that presents the drug's interaction types along with the drug features can better demonstrate the intrinsic characteristics of a drug. However, it is challenging to integrate various features effectively because the drug features might be correlated and contain redundant information.
  - State of the art methods and effectiveness
    - There are four popular approaches in the DDI prediction field: Similarity-based methods, Matrix Factorization-based methods, network analysis-based methods and Deep Learning-based methods. Most of current methods are developed to predict whether drugs interact or not, but not to predict the DDI events. Even though few researches created strong efforts in event prediction but there is a space for advancement.
- Paper explanation
  - what did the paper propose
    - The paper proposed a method for predicting DDI and their type (event) based on attributed heterogeneous graph embedding and a deep learning approach.
  - what is the innovations of the method
    - This paper first built a heterogeneous drug network by integrating the drug properties in each type of DDI. It then made predictions on what kind of interaction is between drugs.
  - how well the proposed method work (in its own metrics)
    - Models like DDIMDL, CNN-DDI, DANN-DDI, MDNN, RF, KNN, LR were used for performance comparison with GNN-DDI base on the metrics like ACC, AUPR, AUC, F1 Score, Precision, and Recall. GNN-DDI showed the best performance among different models.
  - what is the contribution to the reasearch regime
    - This paper proposed a new approach in generating heterogeneous drug network. It also helped make predictions on not only drug interaction, but also drug interaction events which can be beneficial to patients and drug production.

```
# code comment is used as inline annotations for your coding
```

Scope of Reproducibility:

1. Hypothesis 1: Embedding dimension size impact the model's performance that dimention size of 32 led to the best accuracy while dimension size of 16 and 64 led to slightly inferior performance.
2. Hypothesis 2: Different integration schemas of drug vectors impact the model's performance. The method concatenates each drug embedding vector in all event types and then multiplies two vectors of drugs pair shows the best performance.
3. Using different drug feature matrix (similarity matrices) displays different performance on the proposed model. The combined feature matrix show the best performance.
4. The proposed method shows the best performance among approaches like MDNN, DDIMDL, CNN-DDI, DANN-DDI, MDNN, DeepDDI, DNN, RF, KNN, and LR. (We may only test with models DNN, RF, KNN, and LR since the rest models are not standard and can't be called from common libraries)

Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

```
# import  packages you need
import numpy as np
from google.colab import drive
```

Data

In this project, we obtained the data from a sqlite3 database which was compiled from DrugBank 5.1.3 verision. It has 4 tables:

1. **drug** contains 572 kinds of drugs and their features.
2. **event** contains the 37264 DDIs between the 572 kinds of drugs.
3. **extraction** is the process result of NLPProcess. Each interaction is transformed to a tuple: {mechanism, action, drugA, drugB}
4. **event_numer** lists the kinds of DDI events and their occurence frequency.

```
# Upload the tables from sqlite3 database called "Event.db"

import sqlite3

#Connection to the DB
conn = sqlite3.connect('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/event.db')

cursor = conn. cursor()
cursor. execute("SELECT name FROM sqlite_master WHERE type='table';") # [('event_number',), ('event',), ('drug',), ('extraction'
print("Tables : ",cursor. fetchall())
cursor. execute("SELECT name FROM PRAGMA_TABLE_INFO('drug');") # [('index',), ('id',), ('target',), ('enzyme',), ('pathway',), (
print("drug : ",cursor. fetchall())
cursor. execute("SELECT name FROM PRAGMA_TABLE_INFO('event');") # [('index',), ('id',), ('target',), ('enzyme',), ('pathway',),
print("event : ",cursor. fetchall())
cursor. execute("SELECT COUNT(*) FROM event") # [('index',), ('id',), ('target',), ('enzyme',), ('pathway',), ('smile',), ('name
print("event row COUNT: ",cursor. fetchall())

# close the DB connection
conn.close()
```

```
Tables :  [('event_number',), ('event',), ('drug',), ('extraction',)]
drug :  [('index',), ('id',), ('target',), ('enzyme',), ('pathway',), ('smile',), ('name',)]
event :  [('index',), ('id1',), ('name1',), ('id2',), ('name2',), ('interaction',)]
event row COUNT:  [(37264,)]
```

```
# Generate similarity matrices
import csv
import numpy as np
import pandas as pd
from pandas import DataFrame
from sklearn.decomposition import PCA

def save_f(name_mat):
```

```
def save_f(name,mat):
  print(name)
  print(mat.shape)
  df = pd.DataFrame(mat)
  df.to_csv('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/'+str(name)+'.csv', header=None, index=False)

# construct similarity matrices from adjacency matrices of the properties useing the Jacquard similarity function
def Jaccard(matrix):
  matrix = np.mat(matrix)
  numerator = matrix * matrix.T
  denominator = np.ones(np.shape(matrix)) * matrix.T + matrix * np.ones(np.shape(matrix.T)) - matrix * matrix.T
  return numerator / denominator


# generate the feature vector
def feature_vector(df, feature_name):
  all_feature = []
  drug_list = np.array(df[feature_name]).tolist()
  # Features for each drug, for example, when feature_name is target, drug_list=["P30556|P05412","P28223|P46098|……"]
  for i in drug_list:
    for each_feature in i.split('|'):
      if each_feature not in all_feature:
        all_feature.append(each_feature)  # obtain all the features
  feature_matrix = np.zeros((len(drug_list), len(all_feature)), dtype=float)
  df_feature = DataFrame(feature_matrix, columns=all_feature)  # Consrtuct feature matrices with key of dataframe
  for i in range(len(drug_list)):
    for each_feature in df[feature_name].iloc[i].split('|'):
      df_feature[each_feature].iloc[i] = 1

  sim_matrix = np.asarray(Jaccard(np.array(df_feature)))
  sim_matrix1 = np.array(sim_matrix)
  pca = PCA(n_components=len(sim_matrix1))  # PCA dimension
  pca.fit(sim_matrix)
  sim_matrix = pca.transform(sim_matrix)

  return sim_matrix

conn = sqlite3.connect('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/event.db')
df_drug = pd.read_sql('select * from drug;', conn)

feature_list = ['target', 'enzyme', 'pathway', 'smile']

print(df_drug[feature_list[0]][:2])
print(df_drug[:][:2])

drugs = df_drug[:]
drugs = np.array(np.vstack((df_drug['index'],df_drug['name'])))
drugs = drugs.T

# save the four individual feature matrix
for feature in feature_list:
  mat = feature_vector(df_drug, feature)
  save_f(feature+"_PCA", mat)

conn.close()
```

```
0    P14780|Q00653|P01375|P01579|P33673
1                         Q02641
Name: target, dtype: object
   index       id                              target         enzyme  \
0      0  DB01296  P14780|Q00653|P01375|P01579|P33673  P33261|P05181
1      1  DB09230                              Q02641         P08684

                              pathway  \
0  hsa:4318|hsa:4791|hsa:7124|hsa:3458
1                            hsa:782

                                               smile          name
0  9|10|14|18|19|20|178|181|283|284|285|286|299|3...   Glucosamine
1  9|10|11|12|13|14|15|16|18|19|20|129|131|132|17...  Azelnidipine
target_PCA
(572, 572)
enzyme_PCA
(572, 572)
pathway_PCA
(572, 572)
smile_PCA
(572, 572)
```

```
# retrieve records from the extraction table in the database
# and onstruct the DDI Matrix

import time
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import itertools

conn = sqlite3.connect('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/event.db')
extraction = pd.read_sql('select * from extraction;', conn)
mechanism = extraction['mechanism']
action = extraction['action']
drugA = extraction['drugA']
drugB = extraction['drugB']
d_label = {}
d_event=[]
for i in range(len(mechanism)):
  d_event.append(mechanism[i]+" "+action[i])
count={}
for i in d_event:
  if i in count:
    count[i]+=1
  else:
    count[i]=1
list1 = sorted(count.items(), key=lambda x: x[1],reverse=True)
for i in range(len(list1)):
  d_label[list1[i][0]]=i

DDI=[]
for i in range(len(d_event)):
  DDI.append(np.hstack((d_label[d_event[i]],drugA[i], drugB[i])))

mat_DDI = np.array(DDI)
key = drugs[:,1]
val = drugs[:,0]
dic = dict(zip(key,val))
postive1 = [dic[item] for item in mat_DDI[:,1]]
postive2 = [dic[item] for item in mat_DDI[:,2]]
full_pos = np.array(np.vstack((mat_DDI[:,0],postive1,postive2))).astype('int32')
full_pos = full_pos.T

df = pd.DataFrame(np.array(full_pos).tolist())
df.to_csv('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/full_pos2.txt', header=None, index=None, sep=' ')

conn.close()
```

```
# generate negative pairs (non-interacting drug pairs) from a given matrix of positive pairs (interacting drug pairs).

def make_neg_pairs(matrix):
  all_pos = np.array(matrix)
  s1=np.unique(all_pos[:,0])
  s2=np.unique(all_pos[:,1])
  s3=set(s1).union(s2)
  conncted_drug = sorted(s3)
  print("there are ", len(s3), " drugs have connaction out of 572")
  ss = [ii for ii in range(572) if not ii in s3]
  print("there are ", len(ss), " drugs without connaction out of 572")
  pairs_false = list()
  pairs = list()
  comparing = all_pos
  print("start callcolate combinations ... ")
  for dr1,dr2 in itertools.combinations(conncted_drug,2):
    d1=np.array([dr1,dr2])
    d2=np.array([dr1,dr2])
    if dr1 == dr2: continue
    else: pairs.append((dr1,dr2))
  print("all pairs : ",len(pairs))
  for dr in tqdm(pairs, desc="pairs_false generating : "):
    d1=np.array([dr[0],dr[1]])
    d2=np.array([dr[1],dr[0]])
    if not (dr[0]==dr[1]):
      if not ((d2 == comparing).all(axis=1).any() or (d1 == comparing).all(axis=1).any()):
        pairs_false.append([dr[0],dr[1]])
  base=[]
  base2=[]
  for o in tqdm(pairs_false, "all_neg generating : "):
    if (not any(o[0] in h for h in base)) or (not any(o[1] in h for h in base)):
      base.append(o)
    else:
      base2.append(o)
  if len(base) > len(conncted_drug) :
    print("less base .... !")
  pairs_f1 = np.array(base2)
  np.random.shuffle(pairs_f1)
  all_neg = np.concatenate((base,pairs_f1[:len(all_pos)-len(base)]),axis=0)
  np.random.shuffle(all_neg)
  print("all_neg.shape : ", all_neg.shape, "all_pos.shape : ", all_pos.shape)
  df = pd.DataFrame(np.array(all_neg).tolist())
  df.to_csv('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/all_neg2.txt', header=None, index=None, sep=' ')
  return all_neg

all_neg = make_neg_pairs(full_pos[:,1:])
```

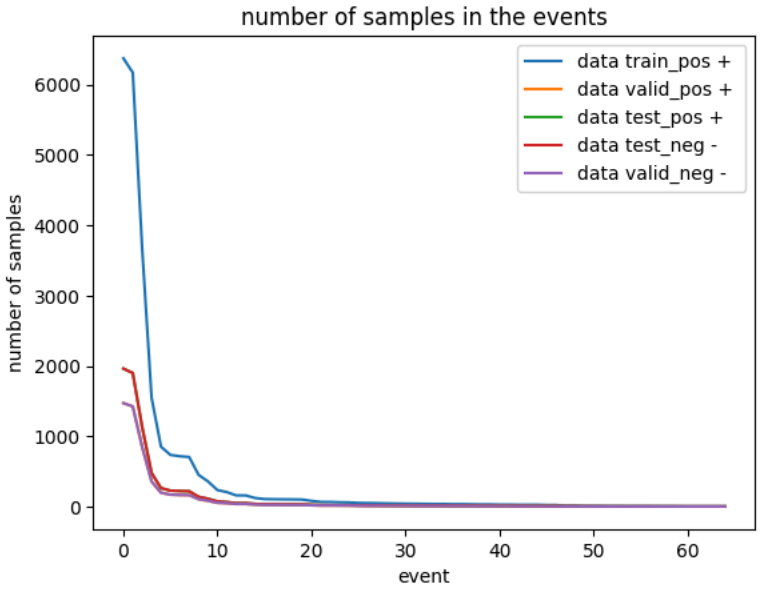
```
    there are  570  drugs have connaction out of 572
    there are  2  drugs without connaction out of 572
    start callcolate combinations ...
    all pairs :  162165
    pairs_false generating : 100%|██████████| 162165/162165 [00:12<00:00, 13471.39it/s]
    all_neg generating : 100%|██████████| 124901/124901 [00:20<00:00, 5970.87it/s]
    all_neg.shape :  (37264, 2) all_pos.shape :  (37264, 2)
```

```
# prepare the train and test subsets

for itm in tqdm(range(9)):
  if itm == 0:
    full_pos = np.array(np.array(pd.read_csv("/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/full_pos2.txt", header=
  elif itm == 1:
    all_neg = np.array(np.array(pd.read_csv("/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/all_neg2.txt", header=No
  elif itm == 2:
    target = np.array(np.array(pd.read_csv("/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/target_PCA.csv", header=N
    enzyme = np.array(np.array(pd.read_csv("/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/enzyme_PCA.csv", header=N
    pathway = np.array(np.array(pd.read_csv("/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/pathway_PCA.csv", header
    smile = np.array(np.array(pd.read_csv("/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/smile_PCA.csv", header=Non
  elif itm == 3:
    full_pos = np.array(np.vstack((full_pos[:,0],full_pos[:,1],full_pos[:,2],[1]*len(full_pos)))).astype('int32').T
    all_cat_pos = []
    for i in range(65):
      all_cat_pos.append(([np.array(item).tolist() for item in full_pos if item[0]==i]))
  elif itm == 4:
    l_l = len(all_cat_pos[0])
    f_l = 0
    all_cat_neg = []
    for i in range(65):
      all_cat_neg.append(np.vstack(([i]* len(all_cat_pos[i]),all_neg[f_l:l_l,0].tolist(),all_neg[f_l:l_l,1].tolist(),[0]* len(al
      f_l = l_l
      if i<64:
        l_l += len(all_cat_pos[i+1])
  elif itm == 5:
    train_cat_pos = []
    train_cat_neg = []
    valid_cat_pos = []
    valid_cat_neg = []
    test_cat_pos = []
    test_cat_neg = []
    for i in range(65):
      train_cat_pos.append((all_cat_pos[i][:(len(all_cat_pos[i])*65//100)]))
      train_cat_neg.append((all_cat_neg[i][:(len(all_cat_neg[i])*65//100)]))
      valid_cat_pos.append((all_cat_pos[i][(len(all_cat_pos[i])*65//100):(len(all_cat_pos[i])*80//100)]))
      valid_cat_neg.append((all_cat_neg[i][(len(all_cat_neg[i])*65//100):(len(all_cat_neg[i])*80//100)]))
      test_cat_pos.append((all_cat_pos[i][(len(all_cat_pos[i])*80//100):]))
      test_cat_neg.append((all_cat_neg[i][(len(all_cat_neg[i])*80//100):]))
    train_cat_pos1 = np.array([ii for i in train_cat_pos for ii in i ])
    train_cat_neg1 = np.array([ii for i in train_cat_neg for ii in i ])
    valid_cat_pos1 = np.array([ii for i in valid_cat_pos for ii in i ])
    valid_cat_neg1 = np.array([ii for i in valid_cat_neg for ii in i ])
    test_cat_pos1 = np.array([ii for i in test_cat_pos for ii in i ])
    test_cat_neg1 = np.array([ii for i in test_cat_neg for ii in i ])
  elif itm == 6:
    # train_cat = np.array(np.vstack((train_cat_pos1,train_cat_neg1)))
    valid_cat = np.array(np.vstack((valid_cat_pos1,valid_cat_neg1)))
    test_cat = np.array(np.vstack((test_cat_pos1,test_cat_neg1)))
    # train_final = np.array(train_cat[:,:3])
    train_final = np.array(train_cat_pos1[:,:3])
  elif itm == 7:
    m1 = np.array(target).astype(np.float64)
    m2 = np.array(enzyme).astype(np.float64)
    m3 = np.array(pathway).astype(np.float64)
    m4 = np.array(smile).astype(np.float64)
    # print("m1 : ",len(m1)," m2 : ",len(m2)," m3 : ",len(m3)," m4 : ",len(m4))
    f_all_m1 = np.array(np.column_stack((drugs[:,0],m1)))
    f_all_m2 = np.array(np.column_stack((drugs[:,0],m2)))
    f_all_m3 = np.array(np.column_stack((drugs[:,0],m3)))
    f_all_m4 = np.array(np.column_stack((drugs[:,0],m4)))
    print(len(f_all_m1[:]),len(f_all_m1[0]))
  else:
    print("\n################# DDI copmleted #################")
    print("################# featuers copmleted #################")

print(test_cat_pos1.shape,valid_cat_pos1.shape,train_cat_pos1.shape,train_cat_pos1[0],valid_cat_pos1[0])
tr = []
print(" event >> valid : test >> whate events in valid : whate events in test ")
for i in range(572):
  s1 = len(train_cat_pos1[np.where(train_cat_pos1[:,1:]==i)])
  s2 = len(valid_cat_pos1[np.where(valid_cat_pos1[:,1:]==i)])
  s3 = len(test_cat_pos1[np.where(test_cat_pos1[:,1:]==i)])
  if s1 == 0:
    vid = valid_cat_pos1[np.where(valid_cat_pos1[:,1:]==i)[0],:].tolist()
    tst = test_cat_pos1[np.where(test_cat_pos1[:,1:]==i)[0],:].tolist()
```

```
      print(i," >>   ",s2," : ",s3 ," >> "
      ,np.unique(valid_cat_pos1[np.where(valid_cat_pos1[:,1:]==i)[0],0])
      ," : ",np.unique(test_cat_pos1[np.where(test_cat_pos1[:,1:]==i)[0],0])
      ," >>   ",vid," : ",tst )
      if not np.isnan(vid).all():
        for item in vid:
          tr.append(item)
      if not np.isnan(tst).all():
        for item in tst:
          tr.append(item)

  print(" missing sample in train : " , tr)

  print(" event >> test : valid")
  for i in range(572):
    s4 = len(test_cat_neg1[np.where(test_cat_neg1[:,1:]==i)])
    s5 = len(valid_cat_neg1[np.where(valid_cat_neg1[:,1:]==i)])
    if s4 == 0 or s5 == 0:
      print(i," >>   ",s4," : ",s5 )

  s1, s2, s3, s4, s5 ,s_all ,l_all ,persnt ,sal= [], [], [], [], [], [], [], [], 0
  events = np.unique(full_pos[:,0])
  for i in events:
    if i < 6:
      pp = (len(full_pos[np.where(full_pos[:,0]==i)])/len(full_pos))*100
      persnt.append(round(pp,1))
      l_all.append(""+str(round(pp,1))+"%")
      s_all.append(len(full_pos[np.where(full_pos[:,0]==i)]))
    else:
      sal += len(full_pos[np.where(full_pos[:,0]==i)])
    s1.append(len(train_cat_pos1[np.where(train_cat_pos1[:,0]==i)]))
    s2.append(len(valid_cat_pos1[np.where(valid_cat_pos1[:,0]==i)]))
    s3.append(len(test_cat_pos1[np.where(test_cat_pos1[:,0]==i)]))
    s4.append(len(test_cat_neg1[np.where(test_cat_neg1[:,0]==i)]))
    s5.append(len(valid_cat_neg1[np.where(valid_cat_neg1[:,0]==i)]))
    # print(i," >> ",s1," : ",s2 ," : ",s3 ," : ",s4 ," : ",s5 )
  s_all.append(sal)
  persnt.append(round(sal/len(full_pos)*100,1))
  l_all.append(""+str(persnt[-1])+"%")
```

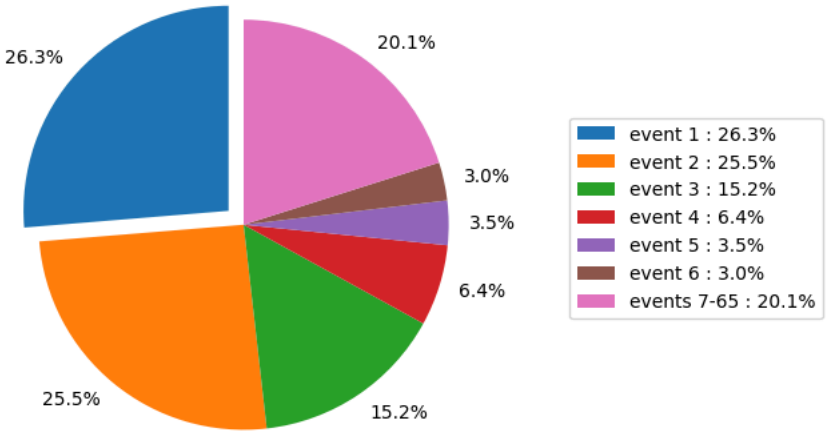
```
      100%|██████████| 9/9 [00:15<00:00,  1.76s/it]
      572 573

      ################## DDI copmleted ##################
      ################## featuers copmleted ##################
      (7474, 4) (5599, 4) (24191, 4) [  0 541 280   1] [  0 441 518   1]
       event >> valid : test >> whate events in valid : whate events in test
      166  >>    0  :  2  >>  []  :  [1]  >>    []  :  [[1, 531, 166, 1], [1, 166, 136, 1]]
      336  >>    0  :  0  >>  []  :  []  >>    []  :  []
      376  >>    0  :  0  >>  []  :  []  >>    []  :  []
      447  >>    0  :  1  >>  []  :  [4]  >>    []  :  [[4, 304, 447, 1]]
      557  >>    0  :  2  >>  []  :  [1]  >>    []  :  [[1, 531, 557, 1], [1, 557, 136, 1]]
       missing sample in train :  [[1, 531, 166, 1], [1, 166, 136, 1], [4, 304, 447, 1], [1, 531, 557, 1], [1, 557, 136, 1]]
       event >> test : valid
      336  >>    0  :  0
      376  >>    0  :  0
```

```
  # Statistics of datasets
  plt.title("number of samples in the events")
  plt.plot(events, s1, label='data train_pos + ')
  plt.plot(events, s2, label='data valid_pos + ')
  plt.plot(events, s3, label='data test_pos + ')
  plt.plot(events, s4, label='data test_neg - ')
  plt.plot(events, s5, label='data valid_neg - ')
  plt.xlabel('event')
  plt.ylabel('number of samples')
  plt.legend()
  plt.show()
```




```
# print('\n',tr,'\n',vid,'\n',tst)
print(train_final.shape)
tr1 = np.array(tr)

# plt.title("number of samples in the events")
# plt.bar(l_all, s_all)
print(s_all," | sum : ",sum(s_all)," all : ",len(full_pos),"\n",persnt," | ",sum(persnt))

ddd = np.zeros((7)).tolist()
ddd[0] = 0.1
myexplode = ddd

labels = ["event "+str(i+1)+" : "+str(j) for i,j in enumerate(l_all)]
labels[-1] = "events 7-65 : "+str(l_all[-1])
# title = plt.title("number of samples in the events")
# title.set_ha("center")
plt.gca().axis("equal")
pie = plt.pie(persnt, labels = l_all, explode = myexplode, startangle=90)
plt.legend(pie[0],labels, bbox_to_anchor=(0.83,0.5), loc="center", fontsize=10, bbox_transform=plt.gcf().transFigure)
plt.subplots_adjust(left=0.0, bottom=0.1, right=0.6)
# image_format = 'svg' # e.g .png, .svg, etc.
# image_name = 'event_pie.svg'
# plt.savefig(image_name, format=image_format, dpi=1200)

print(tr1.shape, tr1)
train_final1 = np.concatenate((train_final,tr1[:,:3]))
print(train_final1.shape,train_final1[-1])
train_final = train_final1
```

```
(24191, 3)
[9810, 9496, 5646, 2386, 1312, 1132, 7482]  | sum :  37264  all :  37264
 [26.3, 25.5, 15.2, 6.4, 3.5, 3.0, 20.1]  |  100.0
(5, 4) [[  1 531 166   1]
 [  1 166 136   1]
 [  4 304 447   1]
 [  1 531 557   1]
 [  1 557 136   1]]
(24196, 3) [  1 557 136]
```




```
# Save all the proprecessed data into .txt files for next step using
df = pd.DataFrame(np.array(train_final))
df.to_csv('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/train.txt', header=None, index=None, sep=' ')
df = pd.DataFrame(np.array(valid_cat))
df.to_csv('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/valid.txt', header=None, index=None, sep=' ')
df = pd.DataFrame(np.array(test_cat))
df.to_csv('/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/test.txt', header=None, index=None, sep=' ')

def write_f(a_f,path_f):
  print(a_f.shape)
  a,b = a_f.shape
  b = b-1
  with open(path_f, "w") as txt_file:
    csv.writer(txt_file, delimiter=' ').writerow([a,b])
    csv.writer(txt_file, delimiter=' ').writerows(a_f)

print(len(f_all_m1[:]),len(f_all_m1[0]), f_all_m1.shape)
f1 = write_f(f_all_m1,'/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/featuers_m1.txt')
f2 = write_f(f_all_m2,'/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/featuers_m2.txt')
f3 = write_f(f_all_m3,'/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/featuers_m3.txt')
f4 = write_f(f_all_m4,'/content/drive/MyDrive/Colab Notebooks/GNN-DDI/GNN_DDI/DDI/data5/featuers_m4.txt')
```

```
572 573 (572, 573)
(572, 573)
(572, 573)
(572, 573)
(572, 573)
```

Model

The proposed model contains two stages:

1. Collect the drugs information from different sources and then integrate them through the formation of an attributed heterogeneous network and generate a drug embedding vector based on different drug interaction types and drug attributes. The following figure provides an overview on the first stage.

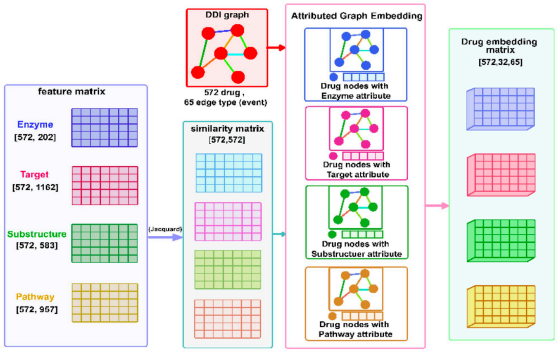


Figure 8. A view of the first step of the proposed method.

2. Aggregate the representation vectors then predictions of the DDIs and their events are performed through a deep multi-model framework. The following figure provides an overview on the second stage.

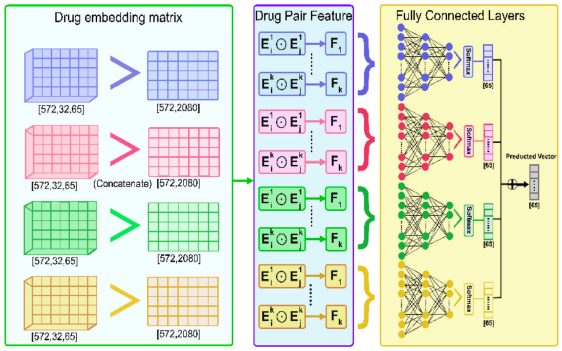


Figure 10. A view of the second stage of the proposed method.

```
import tensorflow as tf
import numpy as np
from tqdm import tqdm


class MyModel:
    def __init__(self, num_nodes, embedding_size, edge_type_count, feature_dim=None):
        self.num_nodes = num_nodes
        self.embedding_size = embedding_size
        self.edge_type_count = edge_type_count
        self.feature_dim = feature_dim

        self.build_model()

    def build_model(self):
        self.node_embeddings = tf.Variable(tf.random_uniform([self.num_nodes, self.embedding_size], -1.0, 1.0))
        self.trans_weights = tf.Variable(tf.truncated_normal([self.edge_type_count, self.embedding_size, self.embedding_size], s

        if self.feature_dim is not None:
            self.feature_weights = tf.Variable(tf.truncated_normal([self.feature_dim, self.embedding_size], stddev=1.0 / np.sqrt

    def DNN(self):
        model = tf.keras.Sequential([
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(256, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(self.num_nodes, activation='softmax')
        ])

        model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

        return model

    def forward(self, inputs, node_neigh):
        node_embed = tf.nn.embedding_lookup(self.node_embeddings, inputs)
        # Add additional forward steps depending on the model complexity and operations
        return node_embed
```

Training

In the training process, the Noise-Contrastive Estimation (NCE) loss is used as loss function, and the Adam Optimizer is uesd as optimizer function.

```
model = MyModel()

def loss_func(labels, logits):
    nce_weights = tf.Variable(tf.truncated_normal([model.num_nodes, model.embedding_size], stddev=1.0 / tf.sqrt(model.embedding_
    nce_biases = tf.Variable(tf.zeros([model.num_nodes]))

    loss = tf.reduce_mean(
        tf.nn.nce_loss(weights=nce_weights,
                       biases=nce_biases,
                       labels=labels,
                       inputs=logits,
                       num_sampled=10,  # Number of negative examples to sample
                       num_classes=model.num_nodes))
    return loss

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

def train_model_one_iter(model, loss_func, optimizer, data):
    with tf.GradientTape() as tape:
        logits = model.call(data, training=True)
        loss = loss_func(data[1], logits)  # Assuming data[1] is train_labels

    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss

num_epochs = 10
for epoch in range(num_epochs):
    # data was not processed yet
    data = (train_inputs, train_labels, train_types, node_neigh)
    loss = train_model_one_iter(model, loss_func, optimizer, data)
    print(f"Epoch {epoch+1}, Loss: {loss.numpy():.2f}")
```

Evaluation

In this paper, the model is evaluated with 5-fold cross validation techique with this 4 subsets were used for training and 1 subset was used for testing. The metrics used in this project include ACC, AUPR, AUC, F1 Score, Precision, and Recall. Among these metrics, AUPR and AUC use the micro metrics and the rest use macro metrics.

```
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import precision_recall_curve

# CV approach
def cross_validation(feature_matrix, label_matrix, clf_type, event_num, seed, CV):
    all_eval_type = 11
    result_all = np.zeros((all_eval_type, 1), dtype=float)
    each_eval_type = 6
    result_eve = np.zeros((event_num, each_eval_type), dtype=float)
    y_true = np.array([])
    y_pred = np.array([])
    y_score = np.zeros((0, event_num), dtype=float)
    index_all_class = get_index(label_matrix, event_num, seed, CV)
    matrix = []
    if type(feature_matrix) != list:
        matrix.append(feature_matrix)
        feature_matrix = matrix

    for k in range(CV):
        print("k : ",k)
        train_index = np.where(index_all_class != k)
        test_index = np.where(index_all_class == k)
        pred = np.zeros((len(test_index[0]), event_num), dtype=float)
        # dnn=DNN()
        for i in range(len(feature_matrix)):
            print("f : ",i)
            xx = bring_f(str(feature_matrix[i]))
            xx = np.array(xx)
            x_train = xx[train_index]
            x_test = xx[test_index]
            xx = 0
            y_train = label_matrix[train_index]
            # one-hot encoding
            y_train_one_hot = np.array(y_train)
            y_train_one_hot = (np.arange(y_train_one_hot.max() + 1) == y_train[:, None]).astype(dtype='float32')
            y_test = label_matrix[test_index]
            # one-hot encoding
            y_test_one_hot = np.array(y_test)
            y_test_one_hot = (np.arange(y_test_one_hot.max() + 1) == y_test[:, None]).astype(dtype='float32')

            # CV for other models
            '''
            if clf_type == 'DDIMDL':
                dnn = DNN()
                early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='auto')
                dnn.fit(x_train, y_train_one_hot, batch_size=128, epochs=100,
                        validation_data=(x_test, y_test_one_hot),
                        callbacks=[early_stopping])
                x_train = 0
                pred += dnn.predict(x_test)
                x_test = 0
                continue
            elif clf_type == 'RF':
                clf = RandomForestClassifier(n_estimators=100)
            elif clf_type == 'GBDT':
                clf = GradientBoostingClassifier()
            elif clf_type == 'SVM':
                clf = SVC(probability=True)
            elif clf_type == 'FM':
                clf = GradientBoostingClassifier()
            elif clf_type == 'KNN':
                clf = KNeighborsClassifier(n_neighbors=4)
            else:
                clf = LogisticRegression()
            '''

            clf.fit(x_train, y_train)
            pred += clf.predict_proba(x_test)

        dnn = 0
        pred_score = pred / len(feature_matrix)
        pred_type = np.argmax(pred_score, axis=1)
```

```
            y_true = np.hstack((y_true, y_test))
            y_pred = np.hstack((y_pred, pred_type))
            y_score = np.row_stack((y_score, pred_score))
        return y_pred, y_score, y_true


    # Calculate Different Metrics
    def calculate_metric_score(real_labels,predict_score):
        # Evaluate the prediction performance
        precision, recall, pr_thresholds = precision_recall_curve(real_labels, predict_score)
        aupr_score = auc(recall, precision)
        all_F_measure = np.zeros(len(pr_thresholds))
        for k in range(0, len(pr_thresholds)):
           if (precision[k] + recall[k]) > 0:
               all_F_measure[k] = 2 * precision[k] * recall[k] / (precision[k] + recall[k])
           else:
               all_F_measure[k] = 0
        print("all_F_measure: ")
        print(all_F_measure)
        max_index = all_F_measure.argmax()
        threshold = pr_thresholds[max_index]
        fpr, tpr, auc_thresholds = roc_curve(real_labels, predict_score)
        auc_score = auc(fpr, tpr)

        f = f1_score(real_labels, predict_score)
        print("F_measure:"+str(all_F_measure[max_index]))
        print("f-score:"+str(f))
        accuracy = accuracy_score(real_labels, predict_score)
        precision = precision_score(real_labels, predict_score)
        recall = recall_score(real_labels, predict_score)
        print('results for feature:' + 'weighted_scoring')
        print(    '************************AUC score:%.3f, AUPR score:%.3f, precision score:%.3f, recall score:%.3f, f score:%.3f,ac
            auc_score, aupr_score, precision, recall, f, accuracy))
        results = [auc_score, aupr_score, precision, recall,  f, accuracy]

        return results
```

Plan for Completion

For now, we already obtained the database and preprocessed it to get the the attributed heterogeneous graph of drugs and feature matrices. We also have part of the model class, same as training and evaluation components. Our next step will be further developing the model class and adding other models for comparison, listed as bellow:

1. Focus on the embedding process for each drug in each event type to generate the embedding matrix. In this matrix, each vector represents the embedding of that drug in a particular event type.
2. Use the concatenation method to reduce the embedding matrices' dimensions into a one-dimensional feature vector which will be used as an input of a multifully connected deep learning model to predict the DDI types.
3. Replenish the training and evaluation process to make it runnable with all metircs mentioned in the paper.
4. Apply other models for comparison on performance.

Results - Not Yet Completed!

In this section, you should finish training your model training or loading your trained model. That is a great experiment! You should share the results with others with necessary metrics and figures.

Please test and report results for all experiments that you run with:

- specific numbers (accuracy, AUC, RMSE, etc)
- figures (loss shrinkage, outputs from GAN, annotation or label of sample pictures, etc)

Model comparison - Not Yet Completed!

Discussion

Reproducing the paper's results was somewhat challenging. While the provided code was helpful, there were some areas where we faced challenges in reproducing. As we picked up the paper late, we still need to work on it to completely reproduce the results. On the positive side, the core GNN model implementation was relatively straightforward to understand and reproduce. However, integrating the DNN into the model and ensuring compatibility with the provided training and evaluation pipelines proved to be challenging.