INFORMATICS INSTITUTE OF TECHNOLOGY
In Collaboration with
UNIVERSITY OF WESTMINSTER (UOW)

A dissertation by

Selvakumaran Thayaan-2017432

Supervised by

Mr. John Sriskandarajah

Submitted in partial fulfillment of the requirements for the

BEng (Hons) Software Engineering Degree

Department of Computing

April 2021

TABLE OF CONTENT

1.Create Nodes

```
graphdb=GraphDatabase.driver(uri="bolt://localhost:7687",auth=("neo4j","thayaan
1998"))
print(graphdb)
session=graphdb.session()
with open('datas.csv', 'r+') as f:
 for l in parse("../datasets/meta_Clothing_Shoes_and_Jewelry.json.gz"):
  json_object = json.loads(l)
  x = "title" in json_object
  if(i<100000):
   if x:
      docx = nfx.TextFrame(json_object['title'])
      docx.remove_special_characters()
      print(i,"nodes added success")
      print(docx)
      q1 = "CREATE
(:Item{ids:'"+json_object['asin']+"',title:'"+str(docx)+"'})"
      f.writelines(f'\n{json_object["asin"]}')
      nodes = session.run(q1)
      i=i+1
  else:
    sys.exit()
```

2.Making Interaction

```
graphdb=GraphDatabase.driver(uri="bolt://localhost:7687",auth=("neo4j","thayaan
1998"))
print(graphdb)
session=graphdb.session()
f = open("output.strict", 'w')
i=0
for l in parse("../datasets/meta_Clothing_Shoes_and_Jewelry.json.gz"):
  json_object = json.loads(l)
  if(i<100000):
   if "related" in json_object:
     if i>64534:
       print(i)
       if "also_bought" in json_object['related']:
         for x in json_object['related']["also_bought"]:
             print(x)
             with open('datas.csv', 'r+') as f:
                 myDataList = f.readlines()
                 for data in myDataList:
                     if x in data:
                         print("have also_bought relationships")
                         q1 = "match (n:Item),(p:Item) where n.ids='" +
json_object[
                             'asin'] + "' and p.ids='" + x + "' create (n)-
[r:also_bought]->(p)"
                         session.run(q1)
       i=i+1
     else:
       i=i+1

  else:
      i = i + 1
      sys.exit()
```

3.Load all the interacted data

```
%%time
q1='''
MATCH (n)-[interaction:also_viewed]->(p) return n.ids as id1,n.title as
title1,interaction,p.ids as id2,p.title as title2
'''
results = session.run(q1)
data=results.data()
df=pd.DataFrame(data)
df["interaction"]=1
df.head()
```

4.Get most Interacted Data

```
no_of_iteracted_item =
df.groupby(by='id1')['interaction'].count().sort_values(ascending=False)
```

```
no_of_iteracted_item.head(300)
```

```
item_interactioncount=(

df.groupby(by='id1')['interaction'].count().reset_index().rename(columns={'interaction':'interactioncount'})
    [['id1','interactioncount']]
)
item_interactioncount.head()
```

```
interaction_with_count=df.merge(item_interactioncount,left_on="id1",right_on="id1",how='left')
interaction_with_count.head()
```

5.Create train and test data

```
%%time
interaction_popular_items.to_csv("../datasets/alldatas.csv")
df=interaction_popular_items
#import train_test_split module
from sklearn.model_selection import train_test_split
#take 67% as the training set and 33% as the test set
df_train, df_test= train_test_split(df,test_size=0.33)
print(len(df_train))
print(len(df_test))
```

```
df_train.to_csv("../datasets/train.csv",index=False)
```

```
df_test.to_csv("../datasets/test.csv",index=False)
```

6.Make dataset to Binary format

```
train_item_matrix=df_train_all.pivot_table(index='id1',
columns='id2',values='interaction').fillna(0)
train_item_matrix
```

```
#collapse
for dataset in [test_item_matrix.values,train_item_matrix.values]:
    dataset= (dataset > 0).astype("int8")

# Make the ratings binary
print("Interaction matrix:")
print(train_item_matrix.values[:10, :10])

print("\nInteractions:")
train_item_matrix=train_item_matrix.astype("int8")
test_item_matrix=test_item_matrix.astype("int8")

unique_ratings = np.unique(train_item_matrix.values)
print(unique_ratings)
```

7.Make dataset to Long format

```
from typing import List
```

```
def wide_to_long(wide:np.array,possible_ratings=[int]):
    def  _get_ratings(arr:np.array,rating:int):
        idx=np.where(arr==rating)
        return np.vstack(
            (idx[0],idx[1],np.ones(idx[0].size,dtype="int8")*rating)
        ).T
    long_arrays=[]
    for r in possible_ratings:
        long_arrays.append(_get_ratings(wide,r))

    return  np.vstack(long_arrays)
```

```
%%time
long_train = wide_to_long(train_item_matrix.values, unique_ratings)
df_train = pd.DataFrame(long_train, columns=["item1", "item2", "interaction"])
long_test = wide_to_long(test_item_matrix.values, unique_ratings)
df_test = pd.DataFrame(long_test, columns=["item1", "item2", "interaction"])
```

8.Model Creation

```
import tensorflow.keras as keras
from tensorflow.keras.layers import (
    Concatenate,
    Dense,
    Embedding,
    Flatten,
    Input,
    Multiply,
)
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2


def create_ncf( number_of_items1: int, number_of_items2: int, latent_dim_mf: 
int = 4, latent_dim_mlp: int = 32,
                reg_mf: int = 0,reg_mlp: int = 0.01, dense_layers: List[int] = 
[8,4],
                reg_layers: List[int] = [0.01, 0.01],) -> keras.Model:

    # input layer
    item1 = Input(shape=(), dtype="int32", name="item1")
    item2 = Input(shape=(), dtype="int32", name="item2")
    # embedding layers
    mf_item1_embedding = 
Embedding(input_dim=number_of_items1,output_dim=latent_dim_mf,name="mf_item1_em
bedding"

,embeddings_initializer="RandomNormal",embeddings_regularizer=l2(reg_mf),input_
length=1,)
    mf_item2_embedding = Embedding(input_dim=number_of_items2, 
output_dim=latent_dim_mf, name="mf_item2_embedding",
                                   embeddings_initializer="RandomNormal" , 
embeddings_regularizer=l2(reg_mf),input_length=1,)
    mlp_item1_embedding = 
Embedding(input_dim=number_of_items1,output_dim=latent_dim_mlp, 
name="mlp_item1_embedding",

embeddings_initializer="RandomNormal",embeddings_regularizer=l2(reg_mlp),input_
length=1,)
    mlp_item2_embedding = 
Embedding(input_dim=number_of_items2,output_dim=latent_dim_mlp,name="mlp_item2_
embedding",

embeddings_initializer="RandomNormal",embeddings_regularizer=l2(reg_mlp),input_
length=1, )
    # MF vector
    mf_item1_latent = Flatten()(mf_item1_embedding(item1))
    mf_item2_latent = Flatten()(mf_item2_embedding(item2))
    mf_cat_latent = Multiply()([mf_item1_latent, mf_item2_latent])
    # MLP vector
    mlp_item1_latent = Flatten()(mlp_item1_embedding(item1))
    mlp_item2_latent = Flatten()(mlp_item2_embedding(item2))
    mlp_cat_latent = Concatenate()([mlp_item1_latent, mlp_item2_latent])

    mlp_vector = mlp_cat_latent
     # build dense layers for model
    for i in range(len(dense_layers)):
        layer = Dense(
            dense_layers[i],
            activity_regularizer=l2(reg_layers[i]),
            activation="relu",
            name="layer%d" % i,
        )
        mlp_vector = layer(mlp_vector)

    predict_layer = Concatenate()([mf_cat_latent, mlp_vector])

    result = Dense(1, activation="sigmoid", kernel_initializer="lecun_uniform",
name="interaction")
    output = result(predict_layer)
    model = Model(inputs=[item1, item2],outputs=[output],)
    return model
```

9.Loss function creation

```
#collapse
from tensorflow.keras.optimizers import Adam

n_items1, n_items2 = train_item_matrix.shape
ncf_model = create_ncf(n_items1, n_items2)

# ncf_model.compile(
#    optimizer=Adam(),
#     loss="mean_squared_error",
#     metrics=[
#         tf.keras.metrics.MeanSquaredError(name="rmse"),
#
#
#     ],
# )

ncf_model.compile(
    optimizer=Adam(),
    loss="binary_crossentropy",
        metrics=[
        tf.keras.metrics.BinaryAccuracy(name="accuracy"),
        ],
    )
ncf_model._name = "neural_collaborative_filtering"
ncf_model.summary()
```

10.Make tensorflow dataset for training

```
def make_tf_dataset( df: pd.DataFrame, targets: List[str],   val_split: float =
0.1,
    batch_size: int = 100,
    seed=40,
):

    n_val = round(df.shape[0] * val_split)
    if seed:
        # shuffle all the rows
        x = df.sample(frac=1, random_state=seed).to_dict("series")
    else:
        x = df.to_dict("series")
    y = dict()
    for t in targets:
        y[t] = x.pop(t)
    ds = tf.data.Dataset.from_tensor_slices((x, y))

    ds_val = ds.take(n_val).batch(batch_size)
    ds_train = ds.skip(n_val).batch(batch_size)
    return ds_train, ds_val
```

11.Train function

```
%%time
import os
import datetime
N_EPOCHS = 10
# define logs and callbacks
@profile
def train():
  logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-
%H%M%S"))
  tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir,
histogram_freq=1)
  early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=0
  )

  train_hist = ncf_model.fit(
    ds_train,
    validation_data=ds_val,
    epochs=N_EPOCHS,
    callbacks=[tensorboard_callback, early_stopping_callback],
    verbose=1
  )

train()
```

12.Prediction View

```
%%time
long_test = wide_to_long(test_item_matrix.values, unique_ratings)
df_test = pd.DataFrame(long_test, columns=["item1", "item2", "interaction"])
ds_test, _ = make_tf_dataset(df_test, ["interaction"], val_split=0, seed=None)
ncf_predictions = ncf_model.predict(ds_test)
df_test["ncf_predictions"] = ncf_predictions
```

```
%%time
std = df_test.describe().loc["std", "ncf_predictions"]
print(std)
if std < 0.01:
    raise ValueError("Model predictions have standard deviation of less than
1e-2."
```

13.Final Recommendation

```
def make_model_matrix(df_test):
    test_item_matrix = create_matrix(df_test, 'title1', 'title2',
'interaction')
    test_item1_indices = pd.DataFrame(sorted(list(set(df_test['title1']))),
columns=['title1'])
    test_item1_indices['item1'] = test_item1_indices.index
    test_item_matrix = test_item_matrix.astype("int8")
    unique_ratings = np.unique(test_item_matrix.values)
    long_test = wide_to_long(test_item_matrix.values, unique_ratings)
    df_test = pd.DataFrame(long_test, columns=["item1", "item2",
"interaction"])
    df_test=df_test[df_test["interaction"] > 0]
    ds_test, _ = make_tf_dataset(df_test, ["interaction"], val_split=0,
seed=None)
    ncf_model =
keras.models.load_model('../datasets/neural_collaborative_filtering.h5')
    ncf_predictions = ncf_model.predict(ds_test)
    df_test["ncf_predictions"] = ncf_predictions
    df_test = pd.merge(df_test, test_item1_indices, on='item1')
    df_test_matrix = create_matrix(df_test, 'title1', 'item2',
'ncf_predictions')
    return df_test_matrix


neighbours=51
@app.route('/getrecommendations',methods=['GET','POST'])
def getrecommendations():
    title = request.json.get('title', None)
    # creating list
    list = []
    df_test_matrix = make_model_matrix(df_test)
    model_knn = NearestNeighbors(metric='cosine', algorithm="brute")
    products_sparse = csr_matrix(df_test_matrix.values)
    model_knn.fit(products_sparse)
    distances, indices =
model_knn.kneighbors(df_test_matrix.loc[title].values.reshape(1, -1),
n_neighbors = neighbours)
    for i in range(0, len(distances.flatten())):
        if distances.flatten()[i]<1:
         distance=distances.flatten()[i]
         i = df_images[df_images["title"] ==
df_test_matrix.index[indices.flatten()[i]]]
         list.append(item(str(i["title"].iloc[0]), i["imUrl"].iloc[0],
str(distance),neighbours))
    return  MyEncoder().encode(list)
```