In this assignment, we want to explore whether GraphSage algorithms can help in improving predictions for training machine learning models on Yelp dataset.

We are going to train traditional Machine Learning and Deep Learning models by categories of restaurants and attributes of users and predict results of 5 ordinal labels: 1, 2, 3, 4, 5. At first, we predict rating by traditional Machine Learning and Deep Learning models and get (test) accuracy of 40% and 45$ separately. Then we convert training data with over 1000 categorical features into graph embeddings of 40 dimensions, pass into previous models and accuracy decreases to (test)48% and 62%.In order to train graph model, we treat reviews as nodes and link those with same user_id and business_id.

So GraphSage help the predictions a little bit since it compress over 1000 sparse categorical features into 40 dimensions. This is especially true when we amount of nodes is very limited. Besides, decrease from training accuracy to test accuracy also reduces a lot(ML model: 53%/40% -> 49%/45%, DL model: 53%/48%->63%/62%) with graph embeddings, which means algorithm has also been stabilized.

In [ ]:
```
import pandas as pd
import os
```

Load the 5000 data

In [ ]:
```
business = pd.read_json('./data/yelp_academic_dataset_business.json', lines=True, nrow
checkin = pd.read_json('./data/yelp_academic_dataset_checkin.json', lines=True, nrows=
review = pd.read_json('./data/yelp_academic_dataset_review.json', lines=True, nrows=50
tip=pd.read_json('./data/yelp_academic_dataset_tip.json', lines=True, nrows=5000)
user = pd.read_json('./data/yelp_academic_dataset_user.json', lines=True, nrows=5000)
```

check the data we load

In [ ]:
```
review.head()
```

Out[ ]:

	review_id	user_id	business_id	stars	useful	f
0	lWC-xP3rd6obsecCYsGZRg	ak0TdVmGKo4pwqdJSTLwWw	buF9druCkbuXLX526sGELQ	4	3	
1	8bFej1QE5LXp4O05qjGqXA	YoVfDbnISlW0f7abNQACIg	RA4V8pr014UyUbDvI-LW2A	4	1	
2	NDhkzczKjLshODbqDoNLSg	eC5evKn1TWDyHCyQAwguUw	_sS2LBIGNT5NQb6PD1Vtjw	5	0	
3	T5fAqjjFooT4V0OeZyuk1w	SFQ1jcnGguO0LYWnbbftAA	0AzLzHfOJgL7ROwhdww2ew	2	1	
4	sjm_uUcQVxab_EeLCqsYLg	0kA0PAJ8QFMeveQWHFqz2A	8zehGz9jnxPqXtOc7KaJxA	4	0	

In [ ]:
```
business.head()
```

Out[ ]:

	business_id	name	address	city	state	postal_code	latitude	longitu
0	6iYb2HFDywm3zjuRg0shjw	Oskar Blues Taproom	921 Pearl St	Boulder	CO	80302	40.017544	-105.2833
1	tCbdrRPZA0oiIYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-122.5933
2	bvN78flM8NLprQ1a1y5dRg	The Reclaimory	4720 Hawthorne Ave	Portland	OR	97214	45.511907	-122.6136
3	oaepsyvc0J17qwi8cfrOWg	Great Clips	2566 Enterprise Rd	Orange City	FL	32763	28.914482	-81.2959
4	PE9uqAjdw0E4-8mjGl3wVA	Crossfit Terminus	1046 Memorial Dr SE	Atlanta	GA	30316	33.747027	-84.3534

In [ ]:
```
user.head()
```

Out[ ]:

	user_id	name	review_count	yelping_since	useful	funny	cool	
0	q_QQ5kBBwlCcbL1s4NVK3g	Jane	1220	2005-03-14 20:26:35	15038	10030	11291	2006,20(
1	dIIKEfOgo0KqUfGQvGikPg	Gabi	2136	2007-08-10 19:01:51	21272	10289	18046	2007,2008,2
2	D6ErcUnFALnCQN4b1W_TlA	Jason	119	2007-02-07 15:47:53	188	128	130	
3	JnPIjvC0cmooNDfsa9BmXg	Kat	987	2009-02-09 16:14:29	7234	4722	4035	
4	37Hc8hr3cw0iHLoPzLK6Ow	Christine	495	2008-03-03 04:57:05	1577	727	1124	

5 rows × 22 columns

In [ ]:

```
checkin.head()
```

Out[ ]:

	business_id	date
0	--0r8K_AQ4FZfLsX3ZYRDA	2017-09-03 17:13:59
1	--0zrn43LEaB4jUWTQH_Bg	2010-10-08 22:21:20, 2010-11-01 21:29:14, 2010...
2	--164t1nclzzmca7eDiJMw	2010-02-26 02:06:53, 2010-02-27 08:00:09, 2010...
3	--2aF9NhXnNVpDV0KS3xBQ	2014-11-03 16:35:35, 2015-01-30 18:16:03, 2015...
4	--2mEJ63SC_8_08_jGgVlg	2010-12-15 17:10:46, 2013-12-28 00:27:54, 2015...

In [ ]:

```
tip.head()
```

Out[ ]:

	user_id	business_id	text	date	compliment_count
0	WCjg0jdHXMlwbqS9tZUx8Q	ENwBByjpoa5Gg7tKgxqwLg	Carne asada chips...	2011-07-22 19:07:35	0
1	42-Z02y9bABShAGZhuSzrQ	jKO4Og6ucdX2-YCTKQVYjg	Best happy hour from 3pm to 6pm! $1 off martin...	2014-09-10 07:33:29	0
2	5u7E3LYp_3eB8dLuUBazXQ	9Bto7mky640ocgezVKSfVg	Nice people, skilled staff, clean location - b...	2013-12-13 23:23:41	0
3	wDWoMG5N9ol4DJ-p7z8EBg	XWFjKtRGZ9khRGtGg2ZvaA	1/2-price bowling & the "Very" Old Fashion are...	2017-07-11 23:07:16	0
4	JmuFlorjjRshHTKzTwNtgg	mkrx0VhSMU3p3uhyJGCoWA	Solid gold's. Great sauna. Great staff, too. E...	2016-11-30 08:46:36	0

save data in csv

In [ ]:

```
business.to_csv('./csv_data/businesses.csv')
review.to_csv('./csv_data/review.csv')
user.to_csv('./csv_data/user.csv')
tip.to_csv('./csv_data/tip.csv')
checkin.to_csv('./csv_data/checkin.csv')
```

merge data together; combine business and review into one data framework by business id; combine user and new data framework by user_id

In [ ]:

```
df = pd.merge(business, review, how='left', on='business_id')
df = pd.merge(df, checkin, how='left', on='business_id')
df = pd.merge(df, user, how='left', on='user_id')
df = pd.merge(df, tip, how='left', on='business_id')
df
```

Out[ ]:

	business_id	name_x	address	city	state	postal_code	latitude	
0	6iYb2HFDywm3zjuRg0shjw	Oskar Blues Taproom	921 Pearl St	Boulder	CO	80302	40.017544	-1
1	tCbdrRPZA0oiIYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-1
2	tCbdrRPZA0oiIYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-1
3	tCbdrRPZA0oiIYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-1
4	bvN78flM8NLprQ1a1y5dRg	The Reclaimory	4720 Hawthorne Ave	Portland	OR	97214	45.511907	-1
...	...	...	...	...	...	...	...	
5806	GJR0oG4vA8ZGfRJRI-NCiA	First Nails	14827 108 Avenue	Surrey	BC	V3R 1W2	49.199423	-1
5807	DL3Xk2nM9cKgE5bVLmPqKA	Lefty's Gourmet Pizza	364 2nd Ave	Niwot	CO	80503	40.102041	-1
5808	dXNfGbh2otsAxLGlpDenGA	Happy Garden	2367 SE 122nd Ave	Portland	OR	97233	45.505157	-1
5809	UnDW3a9VVo_TcvUFLSV0EQ	Patrice Vinci Salon	91 Newbury St	Boston	MA	02116	42.351946	-
5810	AV3Foa7I7T0NX5WRtOH04A	Sushi Town	18033 NW Evergreen Pkwy, Ste M	Beaverton	OR	97006	45.536265	-1

5811 rows × 48 columns

print all features we have

In [ ]:

```
df.columns
```

Out[ ]:
```
Index(['business_id', 'name_x', 'address', 'city', 'state', 'postal_code',
       'latitude', 'longitude', 'stars_x', 'review_count_x', 'is_open',
       'attributes', 'categories', 'hours', 'review_id', 'user_id_x',
       'stars_y', 'useful_x', 'funny_x', 'cool_x', 'text_x', 'date_x',
       'date_y', 'name_y', 'review_count_y', 'yelping_since', 'useful_y',
       'funny_y', 'cool_y', 'elite', 'friends', 'fans', 'average_stars',
       'compliment_hot', 'compliment_more', 'compliment_profile',
       'compliment_cute', 'compliment_list', 'compliment_note',
       'compliment_plain', 'compliment_cool', 'compliment_funny',
       'compliment_writer', 'compliment_photos', 'user_id_y', 'text_y', 'date',
       'compliment_count'],
      dtype='object')
```

Clean the data: (1) delete all rows containing NaN data (2) delete discrete features except 'categories' (3) convert 'categories' into categorical data

In [ ]:
```python
df = df.drop_duplicates(subset=['review_id'])
discrete_feature=['latitude', 'longitude', 'business_id', 'name_x', 'address', 'city',
       'attributes', 'hours', 'review_id', 'user_id_x',
       'text_x', 'date_x',
       'date_y', 'name_y', 'yelping_since','elite', 'friends', 'user_id_y', 'text_y',
df.drop(labels=discrete_feature, axis=1, inplace=True)
```

```
c:\Users\jiaxi\CSCI1420\1xokind_env\lib\site-packages\pandas\core\frame.py:4908: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

In [ ]:
```python
df['categories']
```

Out[ ]:
```
0       Gastropubs, Food, Beer Gardens, Restaurants, B...
1       Salad, Soup, Sandwiches, Delis, Restaurants, C...
2       Salad, Soup, Sandwiches, Delis, Restaurants, C...
3       Salad, Soup, Sandwiches, Delis, Restaurants, C...
4       Antiques, Fashion, Used, Vintage & Consignment...
                              ...
5799                          Beauty & Spas, Hair Salons
5802    Automotive, Car Dealers, Auto Repair, Motorcyc...
5807       Pizza, Italian, Restaurants, Gluten-Free, Food
5809                          Hair Salons, Beauty & Spas
5810                               Japanese, Restaurants
Name: categories, Length: 1539, dtype: object
```

Change feature 'categories' to dummies and drop 'categories'

In [ ]:
```python
#categories_dummies = pd.get_dummies(df['categories'], prefix='category', drop_first=T
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()

df_categories = (df['categories'].str.strip('[]')
                     .str.get_dummies(', ')
                     .rename(columns=lambda x: x.strip('"')))
df.drop(labels='categories', axis=1, inplace=True)
df_categories
```

Out[ ]:

	Acai Bowls	Accessories	Active Life	Acupuncture	African	Airlines	Airport Shuttles	Allergists	Alternative Medicine	A
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
5799	0	0	0	0	0	0	0	0	0	
5802	0	0	0	0	0	0	0	0	0	
5807	0	0	0	0	0	0	0	0	0	
5809	0	0	0	0	0	0	0	0	0	
5810	0	0	0	0	0	0	0	0	0	

1539 rows × 498 columns

(2) fill in all NaN values in continuous data and categories print out if current node has NaN

set val of feature in continuous values that has NaN value

In [ ]:

```
d = {}
for feature, val in df.isna().any().iteritems():
    if val:
        d[feature] = 0
```

In [ ]:

```
df.fillna(d, inplace=True)
df
```

```
c:\Users\jiaxi\CSCI1420\1xokind_env\lib\site-packages\pandas\core\generic.py:6392: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
```

Out[ ]:

	stars_x	review_count_x	stars_y	useful_x	funny_x	cool_x	review_count_y	useful_y	funny_y
0	4.0	86	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	4.0	126	4.0	1.0	0.0	1.0	94.0	278.0	58.0
2	4.0	126	5.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	126	4.0	0.0	0.0	1.0	4880.0	4199.0	2190.0
4	4.5	13	4.0	1.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
5799	4.0	104	5.0	7.0	2.0	1.0	0.0	0.0	0.0
5802	4.5	52	5.0	1.0	0.0	0.0	0.0	0.0	0.0
5807	4.0	52	3.0	0.0	0.0	0.0	0.0	0.0	0.0
5809	4.0	101	5.0	1.0	0.0	0.0	0.0	0.0	0.0
5810	3.5	190	2.0	3.0	0.0	0.0	0.0	0.0	0.0

1539 rows × 24 columns

Set NaN in categories to 0

In [ ]:

```
d = {}
for feature, val in df_categories.isna().any().iteritems():
    if val:
        d[feature] = 0
```

In [ ]:

```
df_categories.fillna(d, inplace=True)
df_categories
```

Out[ ]:

	Acai Bowls	Accessories	Active Life	Acupuncture	African	Airlines	Airport Shuttles	Allergists	Alternative Medicine	A
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
5799	0	0	0	0	0	0	0	0	0	
5802	0	0	0	0	0	0	0	0	0	
5807	0	0	0	0	0	0	0	0	0	
5809	0	0	0	0	0	0	0	0	0	
5810	0	0	0	0	0	0	0	0	0	

1539 rows × 498 columns

Now we can get samples(non_star features) x by merging all non-star features(normalized) and categories together

In [ ]:

```
non_star = []
for (columnName, columnData) in df.iteritems():
    if columnName != 'stars_x':
        non_star += [columnName]
x = df[non_star]
x = (x - x.min()) / (x.max() - x.min())

x = pd.concat([x, df_categories], axis = 1)
x
```

Out[ ]:

	review_count_x	stars_y	useful_x	funny_x	cool_x	review_count_y	useful_y	funny_y	co
0	0.017223	0.0	0.000000	0.000	0.000000	0.000000	0.000000	0.000000	0.000
1	0.025728	0.8	0.045455	0.000	0.052632	0.007307	0.014666	0.004440	0.005
2	0.025728	1.0	0.000000	0.000	0.000000	0.000000	0.000000	0.000000	0.000
3	0.025728	0.8	0.000000	0.000	0.052632	0.379324	0.221513	0.167636	0.193
4	0.001701	0.8	0.045455	0.000	0.000000	0.000000	0.000000	0.000000	0.000
...	...	...	...	...	...	...	...	...	
5799	0.021050	1.0	0.318182	0.125	0.052632	0.000000	0.000000	0.000000	0.000
5802	0.009994	1.0	0.045455	0.000	0.000000	0.000000	0.000000	0.000000	0.000
5807	0.009994	0.6	0.000000	0.000	0.000000	0.000000	0.000000	0.000000	0.000
5809	0.020413	1.0	0.045455	0.000	0.000000	0.000000	0.000000	0.000000	0.000
5810	0.039337	0.4	0.136364	0.000	0.000000	0.000000	0.000000	0.000000	0.000

1539 rows × 521 columns

And label y is the star column. In order to utilize Linear Regression model, we need to convert y into ordinal data. We are going to divide labels into 5 bins to represent 'Very Low', 'Low', 'Medium', 'High', 'Very High' separately

In [ ]:

```
def ordinal_label(df):
    a = pd.cut(df, bins=[-10000, 1, 2, 3, 4, 10000], labels=['Very Low', 'Low', 'Mediu
    return a.replace(to_replace = ['Very Low', 'Low', 'Medium', 'High', 'Very High'],
```

In [ ]:

```
scale_mapper = [1, 2, 3, 4, 5]
y = ordinal_label(df['stars_x'])
y
```

Out[ ]:

```
0       4
1       4
2       4
3       4
4       5
       ..
5799    4
5802    5
5807    4
5809    4
5810    4
Name: stars_x, Length: 1539, dtype: int64
```

Split the Data into Training and Testing Sets

In [ ]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_stat
```

Machine Learning-Logistic regression model

In [ ]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train,y_train)
```

Out[ ]:
```
LinearRegression()
```

accuracy of training data with ML linear regression model

In [ ]:
```
pred = model.predict(x_train)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_train.to_numpy()) / len(pred)
print("training accuracy:", acc)
```

```
training accuracy: 0.5304630381803412
```

accuracy of test data with ML linear regression model

In [ ]:
```
pred = model.predict(x_test)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_test.to_numpy()) / len(pred)
print("test accuracy:", acc)
```

```
test accuracy: 0.4090909090909091
```

Deep Learning- Logistic Regression Model

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [ ]:
```
DL_model = tf.keras.Sequential([
    layers.Flatten(),
    layers.Dense(512),
    layers.Dropout(0.1),
    layers.Dense(256),
    layers.Dropout(0.1),
    layers.Dense(64),
    layers.Dropout(0.1),
    layers.Dense(1)
])
```

In [ ]:
```
DL_model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.005),
    loss='mean_absolute_error')
```

In [ ]:
```
history = DL_model.fit(
    x_train,
    y_train,
    epochs=60,
    batch_size = 200,
    )
```

```
Epoch 1/60
7/7 [==============================] - 1s 10ms/step - loss: 1.7444
Epoch 2/60
7/7 [==============================] - 0s 8ms/step - loss: 1.0140
Epoch 3/60
7/7 [==============================] - 0s 8ms/step - loss: 0.8191
Epoch 4/60
7/7 [==============================] - 0s 8ms/step - loss: 0.7837
Epoch 5/60
7/7 [==============================] - 0s 8ms/step - loss: 0.6552
Epoch 6/60
7/7 [==============================] - 0s 9ms/step - loss: 0.6743
Epoch 7/60
7/7 [==============================] - 0s 9ms/step - loss: 0.6905
Epoch 8/60
7/7 [==============================] - 0s 8ms/step - loss: 0.6773
Epoch 9/60
7/7 [==============================] - 0s 10ms/step - loss: 0.5804
Epoch 10/60
7/7 [==============================] - 0s 9ms/step - loss: 0.5530
Epoch 11/60
7/7 [==============================] - 0s 8ms/step - loss: 0.5769
Epoch 12/60
7/7 [==============================] - 0s 9ms/step - loss: 0.5447
Epoch 13/60
7/7 [==============================] - 0s 9ms/step - loss: 0.5284
Epoch 14/60
7/7 [==============================] - 0s 9ms/step - loss: 0.5640
Epoch 15/60
7/7 [==============================] - 0s 9ms/step - loss: 0.5405
Epoch 16/60
7/7 [==============================] - 0s 8ms/step - loss: 0.6193
Epoch 17/60
7/7 [==============================] - 0s 8ms/step - loss: 0.5976
Epoch 18/60
7/7 [==============================] - 0s 8ms/step - loss: 0.5270
Epoch 19/60
7/7 [==============================] - 0s 8ms/step - loss: 0.5129
Epoch 20/60
7/7 [==============================] - 0s 8ms/step - loss: 0.5055
Epoch 21/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4873
Epoch 22/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4744
Epoch 23/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4780
Epoch 24/60
7/7 [==============================] - 0s 9ms/step - loss: 0.5064
Epoch 25/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4943
Epoch 26/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4951
Epoch 27/60
7/7 [==============================] - 0s 7ms/step - loss: 0.4790
Epoch 28/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4538
Epoch 29/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4827
Epoch 30/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4735
```

```
Epoch 31/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4741
Epoch 32/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4680
Epoch 33/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4505
Epoch 34/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4664
Epoch 35/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4568
Epoch 36/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4412
Epoch 37/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4788
Epoch 38/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4793
Epoch 39/60
7/7 [==============================] - 0s 8ms/step - loss: 0.5188
Epoch 40/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4926
Epoch 41/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4637
Epoch 42/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4794
Epoch 43/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4429
Epoch 44/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4295
Epoch 45/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4208
Epoch 46/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4297
Epoch 47/60
7/7 [==============================] - 0s 10ms/step - loss: 0.4130
Epoch 48/60
7/7 [==============================] - 0s 12ms/step - loss: 0.4467
Epoch 49/60
7/7 [==============================] - 0s 11ms/step - loss: 0.4609
Epoch 50/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4249
Epoch 51/60
7/7 [==============================] - 0s 10ms/step - loss: 0.4279
Epoch 52/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4261
Epoch 53/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4270
Epoch 54/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4003
Epoch 55/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4492
Epoch 56/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4330
Epoch 57/60
7/7 [==============================] - 0s 9ms/step - loss: 0.4405
Epoch 58/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4714
Epoch 59/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4166
Epoch 60/60
7/7 [==============================] - 0s 8ms/step - loss: 0.4434
```

accuracy of training data with DL linear regression model

In [ ]:
```
pred = DL_model.predict(x_train)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_train.to_numpy()) / len(pred)
print("training accuracy:", acc)
```

```
training accuracy: 0.4979691307879773
```

accuracy of test data with DL linear regression model

In [ ]:
```
pred = DL_model.predict(x_test)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_test.to_numpy()) / len(pred)
print("test accuracy:", acc)
```

```
test accuracy: 0.45454545454545453
```

Now we move on to next part: generate graph embeddings with GraphSage algorithm and test whether they can increase accuracy of prediction. We regard each sample as one node set review_id as its index. Then we add business_id and user_id to

In [ ]:
```
import stellargraph as sg
from stellargraph.mapper import GraphSAGENodeGenerator
from stellargraph.mapper import GraphSAGELinkGenerator
from stellargraph.data import UnsupervisedSampler
from stellargraph.layer import GraphSAGE, link_classification
import itertools
```

In [ ]:
```
df = pd.merge(business, review, how='left', on='business_id')
df = pd.merge(df, checkin, how='left', on='business_id')
df = pd.merge(df, user, how='left', on='user_id')
df = pd.merge(df, tip, how='left', on='business_id')
df
```

Out[ ]:

	business_id	name_x	address	city	state	postal_code	latitude	
0	6iYb2HFDywm3zjuRg0shjw	Oskar Blues Taproom	921 Pearl St	Boulder	CO	80302	40.017544	-1
1	tCbdrRPZA0oilYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-1
2	tCbdrRPZA0oilYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-1
3	tCbdrRPZA0oilYSmHG3J0w	Flying Elephants at PDX	7000 NE Airport Way	Portland	OR	97218	45.588906	-1
4	bvN78flM8NLprQ1a1y5dRg	The Reclaimory	4720 Hawthorne Ave	Portland	OR	97214	45.511907	-1
...	...	...	...	...	...	...	...	
5806	GJR0oG4vA8ZGfRJRI-NCiA	First Nails	14827 108 Avenue	Surrey	BC	V3R 1W2	49.199423	-1
5807	DL3Xk2nM9cKgE5bVLmPqKA	Lefty's Gourmet Pizza	364 2nd Ave	Niwot	CO	80503	40.102041	-1
5808	dXNfGbh2otsAxLGlpDenGA	Happy Garden	2367 SE 122nd Ave	Portland	OR	97233	45.505157	-1
5809	UnDW3a9VVo_TcvUFLSV0EQ	Patrice Vinci Salon	91 Newbury St	Boston	MA	02116	42.351946	-
5810	AV3Foa7I7T0NX5WRtOH04A	Sushi Town	18033 NW Evergreen Pkwy, Ste M	Beaverton	OR	97006	45.536265	-1

5811 rows × 48 columns

In [ ]:

```
batch_size = 50
in_samples = [5, 2]
out_samples = [5, 2]
```

In [ ]:

```
review_nodes = pd.concat([x, df['business_id']], axis = 1)
review_nodes = pd.concat([review_nodes, df['user_id_x']], axis = 1)
review_nodes = pd.concat([review_nodes, df['review_id']], axis = 1)
review_nodes = pd.concat([review_nodes, y], axis = 1)
review_nodes = review_nodes.dropna()
review_nodes = review_nodes.drop_duplicates(subset=['review_id'])
```

In [ ]:

```
y = review_nodes['stars_x']
```

In [ ]:
```
d_business = collections.defaultdict(list)
d_user = collections.defaultdict(list)
for r_id, b_id in zip(review_nodes['review_id'], review_nodes['business_id']):
    d_business[b_id] += [r_id]

for r_id, u_id in zip(review_nodes['review_id'], review_nodes['user_id_x']):
    d_user[u_id] += [r_id]
```

In [ ]:
```
review_edges = pd.DataFrame()
d_edges = collections.defaultdict(list)
for b_ids in d_business.values():
    if len(b_ids) >= 2:
        for a, b in itertools.combinations(b_ids, 2):
            d_edges["source"] += [a]
            d_edges["target"] += [b]

for u_ids in d_user.values():
    if len(u_ids) >= 2:
        for a, b in itertools.combinations(u_ids, 2):
            d_edges["source"] += [a]
            d_edges["target"] += [b]
review_edges = pd.DataFrame(d_edges)
```

In [ ]:
```
review_nodes = review_nodes.set_index('review_id')
```

In [ ]:
```
review_nodes.drop(labels=['business_id', 'user_id_x', 'stars_x', 'stars_y'], axis=1, i
review_nodes
```

Out[ ]:

	review_count_x	useful_x	funny_x	cool_x	review_count_y	useful_y
review_id						
sPWRG7i-gwJjo0nDPr87Dw	0.025728	0.045455	0.000	0.052632	0.007307	0.014666
I0q_GX7IkjecNdr4lQDzcQ	0.025728	0.000000	0.000	0.000000	0.000000	0.000000
AJtLSWJsf4E0gVhu8lvbQg	0.025728	0.000000	0.000	0.052632	0.379324	0.221513
xYcbW9MPyLdy8fwbIoAyAQ	0.001701	0.045455	0.000	0.000000	0.000000	0.000000
w3ge0N2w88RY41-0r7zmcw	0.000638	0.000000	0.000	0.000000	0.000000	0.000000
...	...	...	...	...	...	...
ZLIfMmHmeiJWM0gErGaGEg	0.021050	0.318182	0.125	0.052632	0.000000	0.000000
Q_7RI-H7mlX5O9ycqYRhcg	0.009994	0.045455	0.000	0.000000	0.000000	0.000000
ciRlGK11bEaYS_SiWTS5iA	0.009994	0.000000	0.000	0.000000	0.000000	0.000000
LPNWVLwRJ017wQfhVBhbRA	0.020413	0.045455	0.000	0.000000	0.000000	0.000000
lMAjwu1QBclhtsgdWg0MKQ	0.039337	0.136364	0.000	0.000000	0.000000	0.000000

1538 rows × 520 columns

In [ ]:
```
# user_nodes = review_nodes[user_features]
```

```
# user_nodes = user_nodes.drop_duplicates(subset=['source'])
# user_nodes = user_nodes.set_index('source')
# user_nodes
```

In [ ]:
```
# review_edges = review_nodes[['source', 'target']]
# review_edges
```

In [ ]:
```
G = sg.StellarDiGraph(nodes=review_nodes, edges=review_edges)
print(G.info())
```

```
StellarDiGraph: Directed multigraph
 Nodes: 1538, Edges: 2390

 Node types:
  default: [1538]
    Features: float32 vector, length 520
    Edge types: default-default->default

 Edge types:
    default-default->default: [2390]
        Weights: all 1 (default)
        Features: none
```

In [ ]:
```
number_of_walks = 1
length = 5

batch_size = 200
epochs = 2
num_samples = [20, 10]
```

In [ ]:
```
nodes = list(G.nodes())
```

In [ ]:
```
unsupervised_samples = UnsupervisedSampler(G, nodes=nodes, length=length, number_of_wa
```

In [ ]:
```
generator = GraphSAGELinkGenerator(G, batch_size, num_samples)
```

In [ ]:
```
train_gen = generator.flow(unsupervised_samples)
```

In [ ]:
```
layer_sizes = [40, 40]
graphsage = GraphSAGE(layer_sizes=layer_sizes, generator=generator, bias=True, dropout
```

In [ ]:
```
x_inp, x_out = graphsage.in_out_tensors()
prediction = link_classification(output_dim=10, output_act="sigmoid", edge_embedding_m
```

```
link_classification: using 'ip' method to combine node embeddings into edge embedding
s
```

```
c:\Users\jiaxi\CSCI1420\1xokind_env\lib\site-packages\stellargraph\layer\link_inferen
ce.py:340: UserWarning: For inner product link method the output_dim will be ignored
as it is fixed to be 1.
  name="link_classification",
```

In [ ]:
```
G_model = keras.Model(inputs=x_inp, outputs=prediction)

G_model.compile(
    optimizer=keras.optimizers.Adam(lr=1e-3),
    loss=keras.losses.binary_crossentropy,
```

```
    metrics=[keras.metrics.binary_accuracy],
)
```

```
c:\Users\jiaxi\CSCI1420\1xokind_env\lib\site-packages\keras\optimizer_v2\adam.py:105:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [ ]:

```
history = G_model.fit(
    train_gen,
    epochs=4,
    verbose=1,
    use_multiprocessing=False,
    workers=4,
    shuffle=True,
)
```

```
Epoch 1/4
40/40 [==============================] - 19s 411ms/step - loss: 0.5370 - binary_accur
acy: 0.7004
Epoch 2/4
40/40 [==============================] - 17s 403ms/step - loss: 0.5112 - binary_accur
acy: 0.7669
Epoch 3/4
40/40 [==============================] - 18s 407ms/step - loss: 0.5060 - binary_accur
acy: 0.7739
Epoch 4/4
40/40 [==============================] - 17s 398ms/step - loss: 0.5059 - binary_accur
acy: 0.7752
```

Generate node embeddings

In [ ]:

```
x_inp_src = x_inp[0::2]
x_out_src = x_out[0]
embedding_model = keras.Model(inputs=x_inp_src, outputs=x_out_src)
```

In [ ]:

```
node_ids = review_nodes.index
node_gen = GraphSAGENodeGenerator(G, batch_size, num_samples).flow(node_ids)
```

In [ ]:

```
review_embeddings = embedding_model.predict(node_gen, workers=4, verbose=1)
```

```
8/8 [==============================] - 2s 129ms/step
```

In [ ]:

```
review_embeddings.shape
```

Out[ ]:

```
(1538, 40)
```

Now we have graph_embeddings and we can pass it along with labels into ML, DL linear regression model to test if it is useful

In [ ]:

```
x = review_embeddings
x
```

Out[ ]:
```
array([[ 0.04784697, -0.06349546, -0.17156482, ...,  0.21288216,
        -0.18572491,  0.17904122],
       [ 0.00930625, -0.07710905, -0.16803794, ...,  0.22547264,
        -0.1913685 ,  0.18751994],
       [ 0.08226829, -0.05988624, -0.11528733, ...,  0.21549037,
        -0.16205576,  0.17071465],
       ...,
       [-0.24701577,  0.00255168, -0.14631896, ..., -0.00592949,
        -0.06461843,  0.03287771],
       [-0.36126   ,  0.08186973,  0.09246553, ..., -0.00808264,
        -0.08808298,  0.04481641],
       [-0.12055721, -0.10928923,  0.16476761, ..., -0.00734045,
        -0.07999483,  0.04070118]], dtype=float32)
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_stat
```

Pass the graph into ML linear regression model and check its training & test accuracy

In [ ]:
```
model_G = LinearRegression()
model_G.fit(x_train,y_train)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
pred = model_G.predict(x_train)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_train.to_numpy()) / len(pred)
print("training accuracy:", acc)
```

```
training accuracy: 0.46747967479674796
```

In [ ]:
```
pred = model_G.predict(x_test)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_test.to_numpy()) / len(pred)
print("testing accuracy:", acc)
```

```
testing accuracy: 0.4512987012987013
```

Pass the graph into DL linear regression model and check its training & test accuracy

In [ ]:
```
DL_model_G = tf.keras.Sequential([
    layers.Flatten(),
    layers.Dense(512),
    layers.Dropout(0.1),
    layers.Dense(256),
    layers.Dropout(0.1),
    layers.Dense(64),
    layers.Dropout(0.1),
    layers.Dense(1)
])
```

In [ ]:
```
DL_model_G.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.006),
    loss='mean_absolute_error')
```

In [ ]:
```
history = DL_model_G.fit(
    x_train,
    y_train,
    epochs=60,
```

```
    batch_size = 200,
    )
```

```
Epoch 1/60
7/7 [==============================] - 0s 5ms/step - loss: 2.8512
Epoch 2/60
7/7 [==============================] - 0s 5ms/step - loss: 1.5003
Epoch 3/60
7/7 [==============================] - 0s 5ms/step - loss: 0.8555
Epoch 4/60
7/7 [==============================] - 0s 6ms/step - loss: 0.7131
Epoch 5/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6338
Epoch 6/60
7/7 [==============================] - 0s 6ms/step - loss: 0.6491
Epoch 7/60
7/7 [==============================] - 0s 6ms/step - loss: 0.6312
Epoch 8/60
7/7 [==============================] - 0s 6ms/step - loss: 0.6433
Epoch 9/60
7/7 [==============================] - 0s 6ms/step - loss: 0.6545
Epoch 10/60
7/7 [==============================] - 0s 6ms/step - loss: 0.6067
Epoch 11/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5790
Epoch 12/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6154
Epoch 13/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6419
Epoch 14/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6269
Epoch 15/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6203
Epoch 16/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6073
Epoch 17/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5646
Epoch 18/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5577
Epoch 19/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5722
Epoch 20/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5785
Epoch 21/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5749
Epoch 22/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5755
Epoch 23/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6025
Epoch 24/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5563
Epoch 25/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5747
Epoch 26/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6296
Epoch 27/60
7/7 [==============================] - 0s 5ms/step - loss: 0.6208
Epoch 28/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5551
Epoch 29/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5972
Epoch 30/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5830
```

```
Epoch 31/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5691
Epoch 32/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5874
Epoch 33/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5497
Epoch 34/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5686
Epoch 35/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5520
Epoch 36/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5451
Epoch 37/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5930
Epoch 38/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5747
Epoch 39/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5519
Epoch 40/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5674
Epoch 41/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5480
Epoch 42/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5673
Epoch 43/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5503
Epoch 44/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5587
Epoch 45/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5738
Epoch 46/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5558
Epoch 47/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5840
Epoch 48/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5642
Epoch 49/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5493
Epoch 50/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5817
Epoch 51/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5806
Epoch 52/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5342
Epoch 53/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5377
Epoch 54/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5374
Epoch 55/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5510
Epoch 56/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5504
Epoch 57/60
7/7 [==============================] - 0s 5ms/step - loss: 0.5469
Epoch 58/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5594
Epoch 59/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5489
Epoch 60/60
7/7 [==============================] - 0s 6ms/step - loss: 0.5550
```

In [ ]:
```
pred = DL_model_G.predict(x_train)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_train.to_numpy()) / len(pred)
print("training accuracy:", acc)
```

```
training accuracy: 0.6341463414634146
```

In [ ]:
```
pred = DL_model_G.predict(x_test)
pred = ordinal_label(pd.DataFrame(pred)[0])
acc = sum(pred.to_numpy()== y_test.to_numpy()) / len(pred)
print("test accuracy:", acc)
```

```
test accuracy: 0.6233766233766234
```