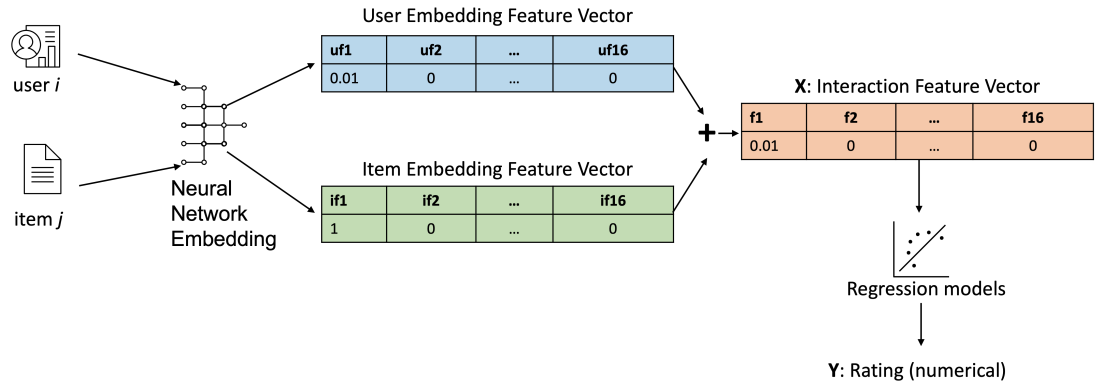
Regression-based Rating Score Prediction using Embedding Features

Estimated time needed: **45** minutes

In our previous lab, you have trained a neural network to predict the user-item interactions while simultaneously extracting the user and item embedding features. In the neural network, extends this by using two embedding vectors as an input into a Neural Network to predict the rating.



Another way to make rating predictions is to use the embedding as an input to a neural network by aggregating them into a single feature vector as input data `X`.

With the interaction label `Y` such as a rating score or an enrollment mode, we can build our other standalone predictive models to approximate the mapping from `X` to `Y`, as shown in the above flowchart.

In this lab, you will be given the course interaction feature vectors as input data `X` and consider label `Y` as the numerical rating scores. As such, we turn the recommender system into a common regression task and you can apply what you have learned about regression modeling to predict the ratings.

Objectives

After completing this lab you will be able to:

- Build regression models to predict ratings using the combined embedding vectors

---

Prepare and setup lab environment

First install and import required libraries:

In [5]:
```
%pip install scikit-learn
%pip install pandas
```

```
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.12/site-pac
kages (1.8.0)
Requirement already satisfied: numpy>=1.24.1 in /opt/conda/lib/python3.12/site-pa
ckages (from scikit-learn) (2.4.1)
Requirement already satisfied: scipy>=1.10.0 in /opt/conda/lib/python3.12/site-pa
ckages (from scikit-learn) (1.17.0)
Requirement already satisfied: joblib>=1.3.0 in /opt/conda/lib/python3.12/site-pa
ckages (from scikit-learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.2.0 in /opt/conda/lib/python3.12/
site-packages (from scikit-learn) (3.6.0)
Note: you may need to restart the kernel to use updated packages.
Collecting pandas
  Downloading pandas-3.0.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_6
4.whl.metadata (79 kB)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-pa
ckages (from pandas) (2.4.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.1
2/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-package
s (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading pandas-3.0.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.
whl (10.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.9/10.9 MB 125.6 MB/s eta 0:00:00
Installing collected packages: pandas
Successfully installed pandas-3.0.0
Note: you may need to restart the kernel to use updated packages.
```

In [36]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
# also set a random state
rs = 123
```

Load datasets

In [4]:
```
rating_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
user_emb_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
item_emb_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
```

The first dataset is the rating dataset that contains a user-item interaction matrix

In [5]:
```
rating_df = pd.read_csv(rating_url)
```

In [6]:
```
rating_df.head()
```

Out[6]:

	user	item	rating
0	1889878	CC0101EN	5
1	1342067	CL0101EN	3
2	1990814	ML0120ENv3	5
3	380098	BD0211EN	5
4	779563	DS0101EN	3

As you can see from the above data, the user and item are just ids, let's substitute them by their embedding vectors:

In [7]:
```
# Load user embeddings
user_emb = pd.read_csv(user_emb_url)
# Load item embeddings
item_emb = pd.read_csv(item_emb_url)
```

In [8]:
```
user_emb.head()
```

Out[8]:

	user	UFeature0	UFeature1	UFeature2	UFeature3	UFeature4	UFeature5	UFeat
0	1889878	0.080721	-0.129561	0.087998	0.030231	0.082691	-0.004176	-0.00
1	1342067	0.068047	-0.112781	0.045208	-0.007570	-0.038382	0.068037	0.11
2	1990814	0.124623	0.012910	-0.072627	0.049935	0.020158	0.133306	-0.03
3	380098	-0.034870	0.000715	0.077406	0.070311	-0.043007	-0.035446	0.03
4	779563	0.106414	-0.001887	-0.017211	-0.042277	-0.074953	-0.056732	0.07

In [9]:
```
item_emb.head()
```

Out[9]:

	item	CFeature0	CFeature1	CFeature2	CFeature3	CFeature4	CFeature5	CF
0	CC0101EN	0.009657	-0.005238	-0.004098	0.016303	-0.005274	-0.000361	-0
1	CL0101EN	-0.008611	0.028041	0.021899	-0.001465	0.006900	-0.017981	0
2	ML0120ENv3	0.027439	-0.027649	-0.007484	-0.059451	0.003972	0.020496	-0
3	BD0211EN	0.020163	-0.011972	-0.003714	-0.015548	-0.007540	0.014847	-0
4	DS0101EN	0.006399	0.000492	0.005640	0.009639	-0.005487	-0.000590	-0

In [10]:
```
# Merge user embedding features
user_emb_merged = pd.merge(rating_df, user_emb, how='left', left_on='user', righ
# Merge course embedding features
merged_df = pd.merge(user_emb_merged, item_emb, how='left', left_on='item', righ
```

In [11]:
```
merged_df.head()
```

Out[11]:

	user	item	rating	UFeature0	UFeature1	UFeature2	UFeature3	UFeatur
0	1889878	CC0101EN	5	0.080721	-0.129561	0.087998	0.030231	0.0826
1	1342067	CL0101EN	3	0.068047	-0.112781	0.045208	-0.007570	-0.0383
2	1990814	ML0120ENv3	5	0.124623	0.012910	-0.072627	0.049935	0.0201
3	380098	BD0211EN	5	-0.034870	0.000715	0.077406	0.070311	-0.0430
4	779563	DS0101EN	3	0.106414	-0.001887	-0.017211	-0.042277	-0.0749

5 rows × 35 columns

Next, we can combine the user features (the column labels starting with `UFeature` and item features (the column labels starting with `CFeature`. In machine learning, there are many ways to aggregate two feature vectors such as element-wise add, multiply, max/min, average, etc. Here we simply add the two sets of feature columns:

In [12]:
```
# Define column names for user and course embedding features
u_features = [f"UFeature{i}" for i in range(16)] # Assuming there are 16 user em
c_features = [f"CFeature{i}" for i in range(16)]  # Assuming there are 16 course

# Extract user embedding features
user_embeddings = merged_df[u_features]
# Extract course embedding features
course_embeddings = merged_df[c_features]
# Extract ratings
ratings = merged_df['rating']

# Aggregate the two feature columns using element-wise add
regression_dataset = user_embeddings + course_embeddings.values
# Rename the columns of the resulting DataFrame
regression_dataset.columns = [f"Feature{i}" for i in range(16)]# Assuming there
# Add the 'rating' column from the original DataFrame to the regression dataset
regression_dataset['rating'] = ratings
# Display the first few rows of the regression dataset
regression_dataset.head()
```

Out[12]:

	Feature0	Feature1	Feature2	Feature3	Feature4	Feature5	Feature6	Feature7
0	0.090378	-0.134799	0.083900	0.046534	0.077417	-0.004537	-0.018561	0.079236
1	0.059437	-0.084740	0.067107	-0.009036	-0.031482	0.050057	0.125847	0.066517
2	0.152061	-0.014739	-0.080112	-0.009516	0.024130	0.153802	-0.048061	-0.119888
3	-0.014707	-0.011257	0.073692	0.054763	-0.050547	-0.020599	0.027146	-0.067012
4	0.112812	-0.001395	-0.011572	-0.032638	-0.080440	-0.057321	0.064595	-0.020880

By now, we have built the input dataset `X` and the output vector `y`:

In [13]:
```
X = regression_dataset.iloc[:, :-1]
y = regression_dataset.iloc[:, -1]
print(f"Input data shape: {X.shape}, Output data shape: {y.shape}")
```

```
Input data shape: (233306, 16), Output data shape: (233306,)
```

TASK: Perform regression on the interaction dataset

Now our input data `X` and output `y` are ready, let's build regression models to map X to y and predict ratings.

y.unique()

You may use `sklearn` to train and evaluate various regression models.

TODO: First split dataset into training and testing datasets

In [14]:
```
### WRITE YOUR CODE HERE
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_sta
```

▶ Click here for Hints

TODO: Create a basic linear regression model

In [18]:
```
### WRITE YOUR CODE HERE
model = LinearRegression()
```

▶ Click here for Hints

TODO: Train the basic regression model with training data

In [19]:
```
### WRITE YOUR CODE HERE
model.fit(X_train,y_train)
```

Out[19]:

▾ LinearRegression ⓘ ?

▶ Parameters

▶ Click here for Hints

TODO: Evaluate the basic regression model

In [23]:
```
### WRITE YOUR CODE HERE
y_pred = model.predict(X_test)
rsme = mean_squared_error(y_test,y_pred)
print(rsme)
```

```
0.6623267903328539
```

▶ Click here for Hints

TODO: Try different regression models such as Ridge, Lasso, ElasticNet and tune their hyperparameters to see which one has the best performance

In [37]:
```
### WRITE YOUR CODE HERE
### Lasso
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lasso', Lasso(max_iter=10000, random_state=42))
])

# 3. Hyperparameter grid
param_grid = {
    'lasso__alpha': [0.001, 0.01, 0.1, 1, 10]
}

# 4. GridSearchCV
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

# 5. Train model
grid_search.fit(X_train, y_train)

# 6. Best model
best_lasso = grid_search.best_estimator_

print("Best alpha:", grid_search.best_params_['lasso__alpha'])

# 7. Predict on test data
y_pred = best_lasso.predict(X_test)

# 8. Evaluation
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Test MSE :", mse)
print("Test RMSE:", rmse)
print("Test R2  :", r2)

# 9. Feature selection info (Optional but useful)
lasso_model = best_lasso.named_steps['lasso']
num_selected_features = np.sum(lasso_model.coef_ != 0)

print("Number of selected features:", num_selected_features)
```

```
Best alpha: 0.01
Test MSE : 0.6622993912533642
Test RMSE: 0.8138177875012098
Test R2  : -1.4840360538448394e-05
Number of selected features: 0
```

Summary

In this lab, you have built regression models to predict numerical course ratings using the embedding feature vectors extracted from neural networks. In the next lab, we can treat the prediction problem as a classification problem as rating only has two categorical values so classification can be a more natural problem statement.

Authors

Yan Luo

Other Contributors

```
toggle##


toggle|Date

toggle|-|-|-|-|

toggle|2021-10-25|1.0|Yan|Created
```