Content-based Course Recommender System Using User Profile and Course Genres

Estimated time needed: **60** minutes

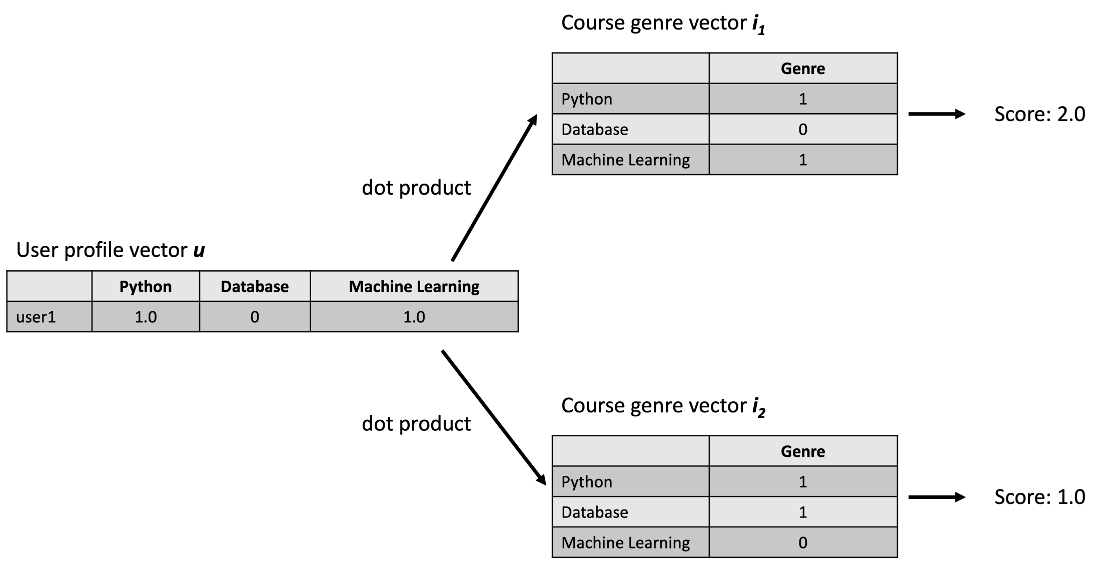
The most common type of content-based recommendation system is to recommend items to users based on their profiles. The user's profile revolves around that user's preferences and tastes. It is shaped based on user ratings, including the number of times a user has clicked on different items or liked those items.

The recommendation process is based on the similarity between those items. The similarity or closeness of items is measured based on the similarity in the content of those items. When we say content, we're talking about things like the item's category, tag, genre, and so on. Essentially the features about an item.

For online course recommender systems, we already know how to extract features from courses (such as genres or BoW features). Next, based on the course genres and users' ratings, we want to further build user profiles (if unknown).

A user profile can be seen as the user feature vector that mathematically represents a user's learning interests.

With the user profile feature vectors and course genre feature vectors constructed, we can use several computational methods, such as a simple dot product, to compute or predict an interest score for each course and recommend those courses with high interest scores.



User profile vector u

	Python	Database	Machine Learning
user1	1.0	0	1.0

Course genre vector i_1

	Genre
Python	1
Database	0
Machine Learning	1

Course genre vector i_2

	Genre
Python	1
Database	1
Machine Learning	0

Objectives

After completing this lab you will be able to:

- Generate a user profile based on course genres and rating
- Generate course recommendations based on a user's profile and course genres

---

Prepare and setup the lab environment

First, let's install and import the required packages.

In [1]:
```
%pip install scikit-learn
%pip install pandas
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.8.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28
_x86_64.whl.metadata (11 kB)
Collecting numpy>=1.24.1 (from scikit-learn)
  Downloading numpy-2.4.1-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_6
4.whl.metadata (6.6 kB)
Collecting scipy>=1.10.0 (from scikit-learn)
  Downloading scipy-1.17.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_6
4.whl.metadata (62 kB)
Collecting joblib>=1.3.0 (from scikit-learn)
  Downloading joblib-1.5.3-py3-none-any.whl.metadata (5.5 kB)
Collecting threadpoolctl>=3.2.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.8.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x
86_64.whl (8.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 92.9 MB/s eta 0:00:00
Downloading joblib-1.5.3-py3-none-any.whl (309 kB)
Downloading numpy-2.4.1-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.w
hl (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 186.6 MB/s eta 0:00:00
Downloading scipy-1.17.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.
whl (35.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.0/35.0 MB 56.1 MB/s eta 0:00:00:0
0:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, numpy, joblib, scipy, scikit-learn
Successfully installed joblib-1.5.3 numpy-2.4.1 scikit-learn-1.8.0 scipy-1.17.0 t
hreadpoolctl-3.6.0
Note: you may need to restart the kernel to use updated packages.
Collecting pandas
  Downloading pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_6
4.whl.metadata (91 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: pytz>=2020.1 in /opt/conda/lib/python3.12/site-pac
kages (from pandas) (2024.2)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2025.3-py2.py3-none-any.whl.metadata (1.4 kB)
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-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.
whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 90.8 MB/s eta 0:00:00
Downloading tzdata-2025.3-py2.py3-none-any.whl (348 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.3.3 tzdata-2025.3
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
```

In [3]:
```python
# also set a random state
rs = 123
```

Lets generate user profiles using course genres and ratings

Suppose we have a very simple course genre dataset that contains only three genres: `Python`, `Database`, and `MachineLearning`.

We also have two courses: `Machine Learning with Python` and `SQL Learning with Python` and their genres as follows:

In [4]:
```
course_genres = ['Python', 'Database', 'MachineLearning']
courses = [['Machine Learning with Python', 1, 0, 1], ["SQL with Python", 1, 1,
courses_df = pd.DataFrame(courses, columns = ['Title'] + course_genres)
courses_df
```

Out[4]:

	Title	Python	Database	MachineLearning
0	Machine Learning with Python	1	0	1
1	SQL with Python	1	1	0

As we can see from the dataset:

- Course `Machine Learning with Python` has `Python` and `MachineLearning` genres
- Course `SQL with Python` has `Python` and `Database` genres

Then let's create another simple user rating dataframe containing the ratings from two users.

In [5]:
```
users = [['user0', 'Machine Learning with Python', 3], ['user1', 'SQL with Pytho
users_df = pd.DataFrame(users, columns = ['User', 'Title', 'Rating'])
users_df
```

Out[5]:

	User	Title	Rating
0	user0	Machine Learning with Python	3
1	user1	SQL with Python	2

Suppose **user0** rated `Machine Learning with Python` as 3 (completed with a certificate) and **user1** rated `SQL with Python` as 2 (just audited or not completed).

Based on their course ratings and course genres. Can we generate a profile vector for each user?

Intuitively, since user0 has completed the course `Machine Learning with Python`, they should be interested in those genres associated with the course, i.e.,Machine Learning and Python.

On the other hand, user0 has not taken the `SQL with Python` so it is likely they are not interested in the database genre.

To quantify such user interests, we could multiply user0's rating vector with a course genre matrix and get the weighted genre's vector for the courses:

In [6]:
```
# User 0 rated course 0 as 3 and course 1 as 0/NA (unknown or not interested)
u0 = np.array([[3, 0]])
```

In [7]:
```
# The course genre's matrix
C = courses_df[['Python', 'Database', 'MachineLearning']].to_numpy()
```

Before multiple them, let's first print their shapes:

In [8]:
```
print(f"User profile vector shape {u0.shape} and course genre matrix shape {C.sh
```

```
User profile vector shape (1, 2) and course genre matrix shape (2, 3)
```

If we multiple a $1 x 2$ vector with a $2 x 3$ matrix, we will get a 1 x 3 vector representing the user profile vector.

$$u_0C = \begin{bmatrix} 3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\\\\\\\ 1 & 1 & 0 \end{bmatrix}$$

In [9]:
```
u0_weights = np.matmul(u0, C)
u0_weights
```

Out[9]:
```
array([[3, 0, 3]])
```

In [10]:
```
course_genres
```

Out[10]:
```
['Python', 'Database', 'MachineLearning']
```

Let's take a look at the result. This `u0_weights` is also called the weighted genre vector and represents the interests of the user for each genre based on the courses they have rated. As we can see from the results, user0 seems interested in `Python` and `MachineLearning` with a rating of 3.

Similarly, we can calculate the weighted genre matrix for user 1:

$$u_1C = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\\\\\\\ 1 & 1 & 0 \end{bmatrix}$$

In [11]:
```
# User 1 rated course 0 as 0 (unknown or not interested) and course 1 as 2
u1 = np.array([[0, 2]])
```

In [12]:
```
u1_weights = np.matmul(u1, C)
u1_weights
```

Out[12]:
```
array([[2, 2, 0]])
```

As we can see from the `u1_weights` vector, user1 seems very interested in `Python` and `Database` with a value 2.

Let's combine the two weighted genre vectors and create a user profile dataframe:

In [13]:
```
weights = np.concatenate((u0_weights.reshape(1, 3), u1_weights.reshape(1, 3)), a
profiles_df = pd.DataFrame(weights, columns=['Python', 'Database', 'MachineLearn
profiles_df.insert(0, 'user', ['user0', 'user1'])
```

In [14]:
```
profiles_df
```

Out[14]:

	user	Python	Database	MachineLearning
0	user0	3	0	3
1	user1	2	2	0

Now this `profiles_df` clearly shows the user profiles or course interests.

Generate recommendation scores for some new courses

With the user profiles generated, we can see that `user0` is very interested in Python and machine learning, and `user1` is very interested in Python and database.

Now, suppose we published some new courses titled as `Python 101`, `Database 101`, and `Machine Learning with R`:

In [15]:
```
new_courses = [['Python 101', 1, 0, 0], ["Database 101", 0, 1, 0], ["Machine Lea
new_courses_df = pd.DataFrame(new_courses, columns = ['Title', 'Python', 'Databa
new_courses_df
```

Out[15]:

	Title	Python	Database	MachineLearning
0	Python 101	1	0	0
1	Database 101	0	1	0
2	Machine Learning with R	0	0	1

Next, how can we calculate a recommendation score for each new course with respect to `user0` and `user1`, using user profile vectors and genre vectors?

One simple but effective way is to apply the dot product to the user profile vector and course genre vector (as they always have the same shape). Since we have two users and three courses, we need to perform a matrix multiplication:

In [16]:
```
profiles_df
```

Out[16]:

	user	Python	Database	MachineLearning
0	user0	3	0	3
1	user1	2	2	0

Let's convert the course genre dataframe into a 2-D numpy array:

In [17]:
```
# Drop the title column
new_courses_df = new_courses_df.loc[:, new_courses_df.columns != 'Title']
course_matrix = new_courses_df.values
course_matrix
```

Out[17]:
```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

In [18]:
```
# course matrix shape
course_matrix.shape
```

Out[18]:
```
(3, 3)
```

As we can see from the above output, the course matrix is a `3 x 3` matrix and each row vector is a course genre vector.

Then we can convert the user profile dataframe into another 2-d numpy array:

In [19]:
```
# Drop the user column
profiles_df = profiles_df.loc[:, profiles_df.columns != 'user']
profile_matrix = profiles_df.values
profile_matrix
```

Out[19]:
```
array([[3, 0, 3],
       [2, 2, 0]])
```

In [20]:
```
profile_matrix.shape
```

Out[20]:
```
(2, 3)
```

The profile matrix is a 2 x 3 matrix and each row is a user profile vector:

If we multiply the course matrix and the user profile matrix, we can get the 2 x 3 course recommendation matrix with each element `(i, j)` representing a recommendation score of course `i` to user `j`. Intuitively, if a user `j` is interested in some topics(genres) and if a course `i` also has the same topics(genres), it means the user profile vector and course genre vector share many common dimensions and a dot product is likely to have a large value.

In [21]:
```
scores = np.matmul(course_matrix, profile_matrix.T)
scores
```

Out[21]:
```
array([[3, 2],
       [0, 2],
       [3, 0]])
```

Now let's add the course titles and user ids back to make the results more clear:

In [22]:
```
scores_df = pd.DataFrame(scores, columns=['User0', 'User1'])
scores_df.index = ['Python 101', 'Database 101', 'Machine Learning with R']
```

In [23]:
```
# recommendation score dataframe
scores_df
```

Out[23]:

	User0	User1
Python 101	3	2
Database 101	0	2
Machine Learning with R	3	0

From the score results, we can see that:

- For user0, the recommended courses are `Python 101` and `Machine Learning with R` because user0 is very interested in Python and machine learning
- For user1, the recommended courses are `Python 101` and `Database 101` because user1 seems very interested in topics like Python and database

TASK: Generate course recommendations based on user profile and course genre vectors

By now you have learned how to calculate recommendation scores using a user profile vector and a course genre vector. Now, let's work on some real-world datasets to generate real personalized courses recommendations.

First, we will load a user's profile dataframe and a course genre dataframe:

In [24]:
```
course_genre_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain
course_genres_df = pd.read_csv(course_genre_url)
```

In [25]:
```
course_genres_df.head()
```

Out[25]:

	COURSE_ID	TITLE	Database	Python	CloudComputing	DataAnalysis	Cont
0	ML0201EN	robots are coming build iot apps with watson ...	0	0	0	0	
1	ML0122EN	accelerating deep learning with gpu	0	1	0	0	
2	GPXX0ZG0EN	consuming restful services using the reactive ...	0	0	0	0	
3	RP0105EN	analyzing big data in r using apache spark	1	0	0	1	
4	GPXX0Z2PEN	containerizing packaging and running a sprin...	0	0	0	0	

In [26]:
```
profile_genre_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomai
profile_df = pd.read_csv(profile_genre_url)
```

In [27]:
```
profile_df.head()
```

Out[27]:

	user	Database	Python	CloudComputing	DataAnalysis	Containers	MachineLearnin
0	2	52.0	14.0	6.0	43.0	3.0	33.
1	4	40.0	2.0	4.0	28.0	0.0	14.
2	5	24.0	8.0	18.0	24.0	0.0	30.
3	7	2.0	0.0	0.0	2.0	0.0	0.
4	8	6.0	0.0	0.0	4.0	0.0	0.

The profile dataframe contains the course interests for each user, for example, user 8 is very interested in R, data analysis, database, and big data:

In [28]:
```
profile_df[profile_df['user'] == 8]
```

Out[28]:

	user	Database	Python	CloudComputing	DataAnalysis	Containers	MachineLearnin
4	8	6.0	0.0	0.0	4.0	0.0	0.

Next, let's load a test dataset, containing test users to whom we want to make course recommendations:

In [29]:
```
test_users_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c
test_users_df = pd.read_csv(test_users_url)
```

In [30]:
```
test_users_df.head()
```

Out[30]:

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

Let's look at how many test users we have in the dataset.

In [31]:
```
# Group the test users DataFrame by the 'user' column and find the maximum value
# then reset the index and drop the old index to obtain a DataFrame with unique
test_users = test_users_df.groupby(['user']).max().reset_index(drop=False)

# Extract the 'user' column from the test_users DataFrame and convert it to a li
test_user_ids = test_users['user'].to_list()
```

```
# Print the total number of test users by obtaining the length of the test_user_
print(f"Total numbers of test users {len(test_user_ids)}")
```

```
Total numbers of test users 33901
```

Then for each test user in the test dataset, you need to first find out which courses are unknown/unselected to them. For example, suppose we have a user `1078030` with profile:

In [32]:
```
test_user_profile = profile_df[profile_df['user'] == 1078030]
test_user_profile
```

Out[32]:

	user	Database	Python	CloudComputing	DataAnalysis	Containers	Machin
18204	1078030	0.0	12.0	0.0	9.0	0.0	

In [33]:
```
# Now let's get the test user vector by excluding the `user` column
test_user_vector = test_user_profile.iloc[0, 1:].values
test_user_vector
```

Out[33]:
```
array([ 0., 12.,  0.,  9.,  0., 12.,  0.,  6.,  0.,  0.,  0.,  0.,  0.,
        0.])
```

We can first find their enrolled courses in `test_users_df`:

In [34]:
```
enrolled_courses = test_users_df[test_users_df['user'] == 1078030]['item'].to_li
enrolled_courses = set(enrolled_courses)
```

In [35]:
```
enrolled_courses
```

Out[35]:
```
{'DA0101EN',
 'DV0101EN',
 'ML0101ENv3',
 'ML0115EN',
 'ML0120ENv2',
 'ML0122ENv1',
 'PY0101EN',
 'ST0101EN'}
```

We then print the entire course list:

In [36]:
```
all_courses = set(course_genres_df['COURSE_ID'].values)
all_courses
```

```
Out[36]: {'AI0111EN',
          'BC0101EN',
          'BC0201EN',
          'BC0202EN',
          'BD0101EN',
          'BD0111EN',
          'BD0115EN',
          'BD0121EN',
          'BD0123EN',
          'BD0131EN',
          'BD0133EN',
          'BD0135EN',
          'BD0137EN',
          'BD0141EN',
          'BD0143EN',
          'BD0145EN',
          'BD0151EN',
          'BD0153EN',
          'BD0211EN',
          'BD0212EN',
          'BD0221EN',
          'BD0223EN',
          'BENTEST4',
          'CB0101EN',
          'CB0103EN',
          'CB0105ENv1',
          'CB0201EN',
          'CC0101EN',
          'CC0103EN',
          'CC0120EN',
          'CC0121EN',
          'CC0150EN',
          'CC0201EN',
          'CC0210EN',
          'CC0250EN',
          'CC0271EN',
          'CL0101EN',
          'CNSC02EN',
          'CO0101EN',
          'CO0193EN',
          'CO0201EN',
          'CO0301EN',
          'CO0302EN',
          'CO0401EN',
          'COM001EN',
          'CP0101EN',
          'DA0101EN',
          'DA0151EN',
          'DA0201EN',
          'DAI101EN',
          'DB0101EN',
          'DB0111EN',
          'DB0113EN',
          'DB0115EN',
          'DB0151EN',
          'DE0205EN',
          'DJ0101EN',
          'DP0101EN',
          'DS0101EN',
          'DS0103EN',
          'DS0105EN',
          'DS0107',
          'DS0110EN',
          'DS0132EN',
          'DS0201EN',
          'DS0301EN',
          'DS0321EN',
          'DV0101EN',
          'DV0151EN',
          'DW0101EN',
          'DX0106EN',
          'DX0107EN',
          'DX0108EN',
          'EE0101EN',
          'GPXX01AVEN',
          'GPXX01DCEN',
          'GPXX01RYEN',
          'GPXX03HFEN',
          'GPXX0435EN',
          'GPXX048OEN',
          'GPXX04HEEN',
          'GPXX04MXEN',
          'GPXX04P5EN',
          'GPXX04TNEN',
          'GPXX04V3EN',
          'GPXX04XJEN',
          'GPXX05LMEN',
          'GPXX05P1EN',
          'GPXX05RDEN',
          'GPXX06KEEN',
          'GPXX06RFEN',
          'GPXX06ZLEN',
          'GPXX0725EN',
          'GPXX0742EN',
          'GPXX07REN',
          'GPXX07UGEN',
          'GPXX07YGEN',
          'GPXX08WYEN',
          'GPXX097UEN',
          'GPXX0A1YEN',
          'GPXX0ADEN',
          'GPXX0BSAEN',
          'GPXX0BUBEN',
          'GPXX0D14EN',
          'GPXX0E3QEN',
          'GPXX0FFCEN',
          'GPXX0FTCEN',
          'GPXX0G31EN',
          'GPXX0G3KEN',
          'GPXX0G81EN',
          'GPXX0HAAEN',
          'GPXX0HC7EN',
          'GPXX0HZ2EN',
          'GPXX0I4FEN',
          'GPXX0IBEN',
          'GPXX0IHMEN',
          'GPXX0JGFEN',
          'GPXX0JLHEN',
          'GPXX0JZ4EN',
          'GPXX0KHHEN',
          'GPXX0KV4EN',
          'GPXX0KY1EN',
          'GPXX0LLEEN',
          'GPXX0M6UEN',
          'GPXX0M7ZEN',
          'GPXX0MIIEN',
          'GPXX0MP0EN',
          'GPXX0NHZEN',
          'GPXX0PG8EN',
          'GPXX0PICEN',
          'GPXX0Q8AEN',
          'GPXX0QJFEN',
          'GPXX0QQ3EN',
          'GPXX0QR3EN',
          'GPXX0QS6EN',
          'GPXX0QTEEN',
          'GPXX0QU9EN',
          'GPXX0RL8EN',
          'GPXX0RQLEN',
          'GPXX0SDXEN',
          'GPXX0T0FEN',
          'GPXX0T3CEN',
          'GPXX0TY1EN',
          'GPXX0UMSEN',
          'GPXX0UN5EN',
          'GPXX0W7KEN',
          'GPXX0WRDEN',
          'GPXX0WTIEN',
          'GPXX0XENEN',
          'GPXX0XFQEN',
          'GPXX0XV3EN',
          'GPXX0YBFEN',
          'GPXX0YMEEN',
          'GPXX0YXHEN',
          'GPXX0Z2PEN',
          'GPXX0ZG0EN',
          'GPXX0ZMZEN',
          'GPXX0ZYVEN',
          'HCC104EN',
          'HCC105EN',
          'IT0101EN',
          'LB0101ENv1',
          'LB0103ENv1',
          'LB0105ENv1',
          'LB0107ENv1',
          'LB0109ENv1',
          'LB0111EN',
          'ML0101EN',
          'ML0101ENv3',
          'ML0103EN',
          'ML0109EN',
          'ML0111EN',
          'ML0115EN',
          'ML0120EN',
          'ML0120ENv2',
          'ML0120ENv3',
          'ML0122EN',
          'ML0122ENv1',
          'ML0122ENv3',
          'ML0151EN',
          'ML0201EN',
          'OS0101EN',
          'PA0101EN',
          'PA0103EN',
          'PA0107EN',
          'PA0109EN',
          'PHPM002EN',
          'PY0101EN',
          'QC0101EN',
          'RAVSCTEST1',
          'RP0101EN',
          'RP0103',
          'RP0103EN',
          'RP0105EN',
          'RP0151EN',
          'SC0101EN',
          'SC0103EN',
          'SC0105EN',
          'SECM03EN',
          'SN0111EN',
          'ST0101EN',
          'ST0201EN',
          'ST0301EN',
          'SW0101EN',
          'SW0201EN',
          'TA0105',
          'TA0105EN',
          'TA0106EN',
          'TMP0101EN',
          'TMP0105EN',
          'TMP0106',
          'TMP107',
          'WA0101EN',
          'WA0103EN',
          'excourse01',
          'excourse02',
          'excourse03',
          'excourse04',
          'excourse05',
          'excourse06',
          'excourse07',
          'excourse08',
          'excourse09',
          'excourse10',
          'excourse11',
          'excourse12',
          'excourse13',
          'excourse14',
          'excourse15',
          'excourse16',
          'excourse17',
          'excourse18',
          'excourse19',
          'excourse20',
          'excourse21',
          'excourse22',
          'excourse23',
          'excourse24',
          'excourse25',
          'excourse26',
          'excourse27',
          'excourse28',
          'excourse29',
          'excourse30',
          'excourse31',
          'excourse32',
          'excourse33',
          'excourse34',
          'excourse35',
          'excourse36',
          'excourse37',
          'excourse38',
          'excourse39',
          'excourse40',
          'excourse41',
          'excourse42',
          'excourse43',
          'excourse44',
          'excourse45',
          'excourse46',
          'excourse47',
          'excourse48',
          'excourse49',
          'excourse50',
          'excourse51',
          'excourse52',
          'excourse53',
          'excourse54',
          'excourse55',
          'excourse56',
          'excourse57',
          'excourse58',
          'excourse59',
          'excourse60',
          'excourse61',
          'excourse62',
          'excourse63',
          'excourse64',
          'excourse65',
          'excourse66',
          'excourse67',
          'excourse68',
          'excourse69',
          'excourse70',
          'excourse71',
          'excourse72',
          'excourse73',
          'excourse74',
          'excourse75',
          'excourse76',
          'excourse77',
          'excourse78',
          'excourse79',
          'excourse80',
          'excourse81',
          'excourse82',
          'excourse83',
          'excourse84',
          'excourse85',
          'excourse86',
          'excourse87',
          'excourse88',
          'excourse89',
          'excourse90',
          'excourse91',
          'excourse92',
          'excourse93'}
```

Then we can use all courses to subtract the enrolled courses to get a set of all unknown courses for user `1078030`, and we want to find potential interested courses hidden in the unknown course list.

In [37]:
```
unknown_courses = all_courses.difference(enrolled_courses)
unknown_courses
```

Out[37]:
```
{'AI0111EN',
 'BC0101EN',
 'BC0201EN',
 'BC0202EN',
 'BD0101EN',
 'BD0111EN',
 'BD0115EN',
 'BD0121EN',
 'BD0123EN',
 'BD0131EN',
 'BD0133EN',
 'BD0135EN',
 'BD0137EN',
 'BD0141EN',
 'BD0143EN',
 'BD0145EN',
 'BD0151EN',
 'BD0153EN',
 'BD0211EN',
 'BD0212EN',
 'BD0221EN',
 'BD0223EN',
 'BENTEST4',
 'CB0101EN',
 'CB0103EN',
 'CB0105ENv1',
 'CB0201EN',
 'CC0101EN',
 'CC0103EN',
 'CC0120EN',
 'CC0121EN',
 'CC0150EN',
 'CC0201EN',
 'CC0210EN',
 'CC0250EN',
 'CC0271EN',
 'CL0101EN',
 'CNSC02EN',
 'CO0101EN',
 'CO0193EN',
 'CO0201EN',
 'CO0301EN',
 'CO0302EN',
 'CO0401EN',
 'COM001EN',
 'CP0101EN',
 'DA0151EN',
 'DA0201EN',
 'DAI101EN',
 'DB0101EN',
 'DB0111EN',
 'DB0113EN',
 'DB0115EN',
 'DB0151EN',
 'DE0205EN',
 'DJ0101EN',
 'DP0101EN',
 'DS0101EN',
 'DS0103EN',
 'DS0105EN',
 'DS0107',
 'DS0110EN',
 'DS0132EN',
 'DS0201EN',
 'DS0301EN',
 'DS0321EN',
 'DV0151EN',
 'DW0101EN',
 'DX0106EN',
 'DX0107EN',
 'DX0108EN',
 'EE0101EN',
 'GPXX01AVEN',
 'GPXX01DCEN',
 'GPXX01RYEN',
 'GPXX03HFEN',
 'GPXX0435EN',
 'GPXX048OEN',
 'GPXX04HEEN',
 'GPXX04MXEN',
 'GPXX04P5EN',
 'GPXX04TNEN',
 'GPXX04V3EN',
 'GPXX04XJEN',
 'GPXX05LMEN',
 'GPXX05P1EN',
 'GPXX05RDEN',
 'GPXX06KEEN',
 'GPXX06RFEN',
 'GPXX06ZLEN',
 'GPXX0725EN',
 'GPXX0742EN',
 'GPXX07REN',
 'GPXX07UGEN',
 'GPXX07YGEN',
 'GPXX08WYEN',
 'GPXX097UEN',
 'GPXX0A1YEN',
 'GPXX0ADEN',
 'GPXX0BSAEN',
 'GPXX0BUBEN',
 'GPXX0D14EN',
 'GPXX0E3QEN',
 'GPXX0FFCEN',
 'GPXX0FTCEN',
 'GPXX0G31EN',
 'GPXX0G3KEN',
 'GPXX0G81EN',
 'GPXX0HAAEN',
 'GPXX0HC7EN',
 'GPXX0HZ2EN',
 'GPXX0I4FEN',
 'GPXX0IBEN',
 'GPXX0IHMEN',
 'GPXX0JGFEN',
 'GPXX0JLHEN',
 'GPXX0JZ4EN',
 'GPXX0KHHEN',
 'GPXX0KV4EN',
 'GPXX0KY1EN',
 'GPXX0LLEEN',
 'GPXX0M6UEN',
 'GPXX0M7ZEN',
 'GPXX0MIIEN',
 'GPXX0MP0EN',
 'GPXX0NHZEN',
 'GPXX0PG8EN',
 'GPXX0PICEN',
 'GPXX0Q8AEN',
 'GPXX0QJFEN',
 'GPXX0QQ3EN',
 'GPXX0QR3EN',
 'GPXX0QS6EN',
 'GPXX0QTEEN',
 'GPXX0QU9EN',
 'GPXX0RL8EN',
 'GPXX0RQLEN',
 'GPXX0SDXEN',
 'GPXX0T0FEN',
 'GPXX0T3CEN',
 'GPXX0TY1EN',
 'GPXX0UMSEN',
 'GPXX0UN5EN',
 'GPXX0W7KEN',
 'GPXX0WRDEN',
 'GPXX0WTIEN',
 'GPXX0XENEN',
 'GPXX0XFQEN',
 'GPXX0XV3EN',
 'GPXX0YBFEN',
 'GPXX0YMEEN',
 'GPXX0YXHEN',
 'GPXX0Z2PEN',
 'GPXX0ZG0EN',
 'GPXX0ZMZEN',
 'GPXX0ZYVEN',
 'HCC104EN',
 'HCC105EN',
 'IT0101EN',
 'LB0101ENv1',
 'LB0103ENv1',
 'LB0105ENv1',
 'LB0107ENv1',
 'LB0109ENv1',
 'LB0111EN',
 'ML0101EN',
 'ML0103EN',
 'ML0109EN',
 'ML0111EN',
 'ML0120EN',
 'ML0120ENv3',
 'ML0122EN',
 'ML0122ENv3',
 'ML0151EN',
 'ML0201EN',
 'OS0101EN',
 'PA0101EN',
 'PA0103EN',
 'PA0107EN',
 'PA0109EN',
 'PHPM002EN',
 'QC0101EN',
 'RAVSCTEST1',
 'RP0101EN',
 'RP0103',
 'RP0103EN',
 'RP0105EN',
 'RP0151EN',
 'SC0101EN',
 'SC0103EN',
 'SC0105EN',
 'SECM03EN',
 'SN0111EN',
 'ST0201EN',
 'ST0301EN',
 'SW0101EN',
 'SW0201EN',
 'TA0105',
 'TA0105EN',
 'TA0106EN',
 'TMP0101EN',
 'TMP0105EN',
 'TMP0106',
 'TMP107',
 'WA0101EN',
 'WA0103EN',
 'excourse01',
 'excourse02',
 'excourse03',
 'excourse04',
 'excourse05',
 'excourse06',
 'excourse07',
 'excourse08',
 'excourse09',
 'excourse10',
 'excourse11',
 'excourse12',
 'excourse13',
 'excourse14',
 'excourse15',
 'excourse16',
 'excourse17',
 'excourse18',
 'excourse19',
 'excourse20',
 'excourse21',
 'excourse22',
 'excourse23',
 'excourse24',
 'excourse25',
 'excourse26',
 'excourse27',
 'excourse28',
 'excourse29',
 'excourse30',
 'excourse31',
 'excourse32',
 'excourse33',
 'excourse34',
 'excourse35',
 'excourse36',
 'excourse37',
 'excourse38',
 'excourse39',
 'excourse40',
 'excourse41',
 'excourse42',
 'excourse43',
 'excourse44',
 'excourse45',
 'excourse46',
 'excourse47',
 'excourse48',
 'excourse49',
 'excourse50',
 'excourse51',
 'excourse52',
 'excourse53',
 'excourse54',
 'excourse55',
 'excourse56',
 'excourse57',
 'excourse58',
 'excourse59',
 'excourse60',
 'excourse61',
 'excourse62',
 'excourse63',
 'excourse64',
 'excourse65',
 'excourse66',
 'excourse67',
 'excourse68',
 'excourse69',
 'excourse70',
 'excourse71',
 'excourse72',
 'excourse73',
 'excourse74',
 'excourse75',
 'excourse76',
 'excourse77',
 'excourse78',
 'excourse79',
 'excourse80',
 'excourse81',
 'excourse82',
 'excourse83',
 'excourse84',
 'excourse85',
 'excourse86',
 'excourse87',
 'excourse88',
 'excourse89',
 'excourse90',
 'excourse91',
 'excourse92',
 'excourse93'}
```

We can get the genre vectors for those unknown courses as well:

In [38]:
```
unknown_course_genres = course_genres_df[course_genres_df['COURSE_ID'].isin(unkn
# Now let's get the course matrix by excluding `COURSE_ID` and `TITLE` columns:
course_matrix = unknown_course_genres.iloc[:, 2:].values
course_matrix
```

Out[38]:
```
array([[0, 0, 0, ..., 1, 1, 0],
       [0, 1, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 1, 0],
       ...,
       [0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 1, 1, 0],
       [0, 0, 0, ..., 1, 1, 0]], shape=(299, 14))
```

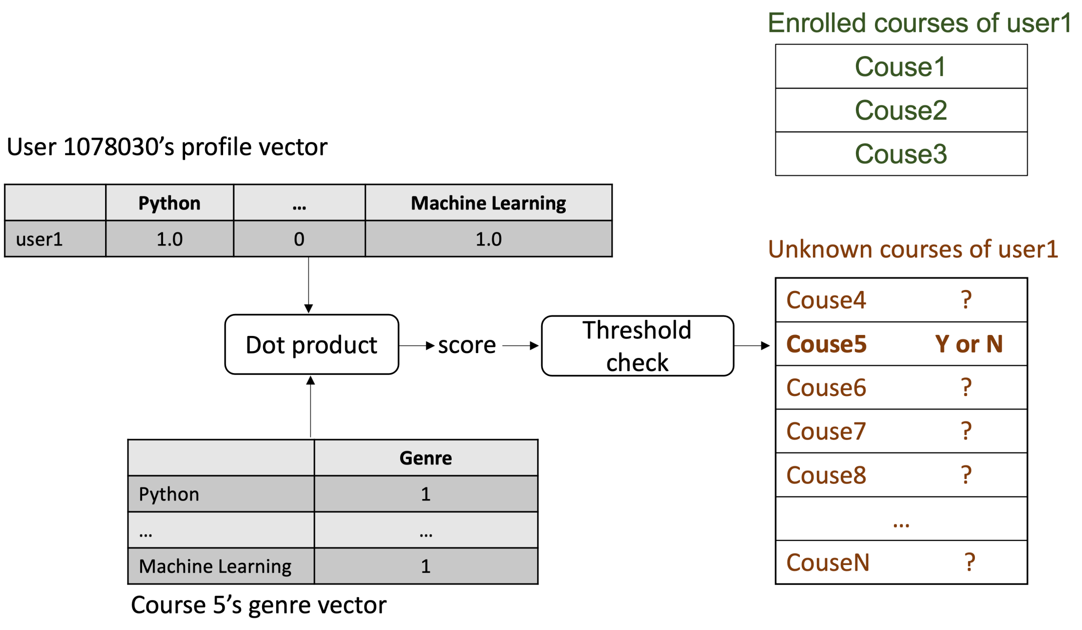
Given the user profile vector for user `1078030` and all the unseen course genres vectors above, you can use the dot product to calculate the recommendation score for each unknown course. e.g., the recommendation score for course `accelerating deep learning with gpu` is:

In [39]:
```
score = np.dot(course_matrix[1], test_user_vector)
score
```

Out[39]:
```
np.float64(30.0)
```

Later, we will need to choose a recommendation score threshold. If the score of any course is above the threshold, we may recommend that course to the user.

The workflow can be summarized in the following flowchart:



Next, let's calculate the recommendation scores of all courses for all the 1000 test users.

In [40]:
```
# Reload the test users dataset from the specified URL using pandas and store it
test_users_df = pd.read_csv(test_users_url)

# Reload the user profiles dataset from the specified URL containing user profil
```

```
profile_df = pd.read_csv(profile_genre_url)

# Reload the course genres dataset from the specified URL containing course genr
course_genres_df = pd.read_csv(course_genre_url)

# Create an empty dictionary to store the results of the recommendation process
res_dict = {}
```

We only want to recommend courses with very high scores so we may set a score threshold to filter out those courses with low scores.

In [41]:
```
# Only keep the score larger than the recommendation threshold
# The threshold can be fine-tuned to adjust the size of generated recommendation
score_threshold = 10.0
```

We defined a function called `generate_recommendation_scores()` to compute the recommendation scores of all the unknown courses for all test users.

TODO: Complete the generate_recommendation_scores() function blow to generate recommendation score for all users. You may also implement the task with different solutions.

In [42]:
```
def generate_recommendation_scores():
    """
    Generate recommendation scores for users and courses.

    Returns:
    users (list): List of user IDs.
    courses (list): List of recommended course IDs.
    scores (list): List of recommendation scores.
    """

    users = []      # List to store user IDs
    courses = []    # List to store recommended course IDs
    scores = []     # List to store recommendation scores

    # Iterate over each user ID in the test_user_ids list
    for user_id in test_user_ids:
        # Get the user profile data for the current user
        test_user_profile = profile_df[profile_df['user'] == user_id]

        # Get the user vector for the current user id (replace with your method
        test_user_vector = test_user_profile.iloc[0, 1:].values

        # Get the known course ids for the current user
        enrolled_courses = test_users_df[test_users_df['user'] == user_id]['item

        # Calculate the unknown course ids
        unknown_courses = all_courses.difference(enrolled_courses)

        # Filter the course_genres_df to include only unknown courses
        unknown_course_df = course_genres_df[course_genres_df['COURSE_ID'].isin(
        unknown_course_ids = unknown_course_df['COURSE_ID'].values

        # Calculate the recommendation scores using dot product
        recommendation_scores = np.dot(unknown_course_df.iloc[:, 2:].values, tes
        # Append the results into the users, courses, and scores list
        for i in range(0, len(unknown_course_ids)):
            score = recommendation_scores[i]

            # Only keep the courses with high recommendation score
            if score >= score_threshold:
                users.append(user_id)
                courses.append(unknown_course_ids[i])
                scores.append(recommendation_scores[i])

    return users, courses, scores
```

NOTE: Instead of using some absolute score threshold, you may also try sorting the scores for each user and return the top-ranked courses.

After you have completed the function `generate_recommendation_scores()` above, you can test it and generate recommendation scores and save the courses recommendations into a dataframe with three columns: `USER`, `COURSE_ID`, `SCORE`:

```
In [43]: # Call the generate_recommendation_scores function to obtain recommendation scor
         # and assign the returned lists to variables users, courses, and scores
         users, courses, scores = generate_recommendation_scores()

         # Create an empty dictionary named res_dict to store the results of the recommen
         res_dict = {}

         # Store the lists of users, courses, and scores into the res_dict dictionary wit
         res_dict['USER'] = users
         res_dict['COURSE_ID'] = courses
         res_dict['SCORE'] = scores

         # Create a DataFrame named res_df using the res_dict dictionary, specifying the
         res_df = pd.DataFrame(res_dict, columns=['USER', 'COURSE_ID', 'SCORE'])

         # Save the res_df DataFrame to a CSV file named "profile_rs_results.csv" without
         res_df.to_csv("profile_rs_results.csv", index=False)

         # Output the res_df DataFrame
         res_df
```

Out[43]:

	USER	COURSE_ID	SCORE
0	2	ML0201EN	43.0
1	2	GPXX0ZG0EN	43.0
2	2	GPXX0Z2PEN	37.0
3	2	DX0106EN	47.0
4	2	GPXX06RFEN	52.0
...	...	...	...
1500419	2102680	excourse62	15.0
1500420	2102680	excourse69	14.0
1500421	2102680	excourse77	14.0
1500422	2102680	excourse78	14.0
1500423	2102680	excourse79	14.0

1500424 rows × 3 columns

Your recommendation results may look like the following screenshot:

With the course recommendation list generated for each test user, you also need to perform some analytic tasks to answer the following two questions:

- On average, how many new courses have been recommended per test user?
- What are the most frequently recommended courses? Return the top-10 commonly recommended courses across all test users.

For example, suppose we have only 3 test users, each user receives the following course recommendations:

- User1: ['course1', 'course2']
- User2: ['course3', 'course4']
- User3: ['course3', 'course4', 'course5']

Then, the average recommended courses per user is: $(2 + 2 + 3) / 3 = 2.33$. The top-2 recommended courses are: `course3`: 2 times, and `course4`: 2 times.

Note that the answers may depend on your score threshold. A lower score threshold yields more recommended courses but with smaller confidence so that some test users may receive very long course recommendation lists and feel overwhelmed.

Ideally, we should limit the maximum course recommendations to be less than 20 courses per user. As such, the average course recommendations per user should also be less than 20 or so. This makes sure we only recommend relevant courses with high confidence (score).

Summary

In this lab, you first learned how to generate a user profile vector based on the user's course ratings and course genre vectors. Then, with the user profile generated, you applied a simple dot product between the user profile vector and the course genre vector to generate a course recommendation score.

The idea is if a user is interested in certain topics (genres) and if a course also has similar topics (genres), which means the user vector and course genre vector share common dimensions, and a dot product is able to capture such similarity.

Authors

Yan Luo

Other Contributors

```
toggle##


toggle|Date

toggle|-|-|-|-|

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