Message to the reviewer – The *user\_course\_views* table does not contain the *course\_name* column as given in the instructions but contains the *course\_id* column which is the same as the *course\_id* column in the course\_tags table (the total unique count matches).

## Python packages required –

pandas, numpy, sklearn, scikit-surprise, psycopg2, sqlalchemy, flask, flask-cors

## Instructions for running the API –

Only the 3 files (controller.py, db\_setup.py, similarity\_model.py) in the [main](https://github.com/aatishsuman/pluralsight-exercise/tree/master/main) folder are needed.

1. Creating the PostgreSQL database and tables –
   1. Download PostgreSQL from [here](https://www.postgresql.org/download/).
   2. Create 2 INI files – server.ini and database.ini containing server and database parameters as shown [here](https://www.postgresqltutorial.com/postgresql-python/connect/). Store these files in the same location as the other files.
   3. Run the db\_setup.py file.
2. Training the model –
   1. Run the similarity\_model.py file.
3. Running the API –
   1. Run the controller.py file.

## API –

*GET* */getSimilarUsers/{user\_handle}*

Parameters: *user\_handle* (required) – the unique identifier for the user

Response:

{

‘*user\_top\_assessment\_tags*’: *list* – Top 3 assessment tags for the user by the normalized assessment score,

‘*user\_highest\_course\_views*’: *list* – Top 3 courses for the user with the highest normalized view time,

‘*user\_interest\_tags*’: *list* – All interest tags of the user,

‘*similar\_score\_users*’: {

‘*users*’: *list* – Top 10 most similar users by assessments scores in order,

‘*similarities*’: *list* – Similarity values (-1 to 1)

},

‘*similar\_course\_views\_users*’: {

‘*users*’: *list* – Top 10 most similar users by course views in order,

‘*similarities*’: *list* – Similarity values (-1 to 1)

},

‘*similar\_interest\_users*’: {

‘*users*’: *list* – Top 10 most similar users by interest tags in order,

‘*similarities*’: *list* – Similarity values (-1 to 1)

},

‘*similar\_overall\_users*’: {

‘*users*’: *list* – Top 10 most similar users by the weighted average of the 3 similarity measures in order,

‘*similarities*’: *list* – Similarity values (-1 to 1)

}

}

## Questions

1. **Tell us about your similarity calculation and why you chose it.**

The similarity between the users can be measured using 3 sets of information – assessment scores (users with similar scores can be considered to be similar to each other), course view times (users viewing the same courses for the same durations can be considered to be similar to each other) and interest tags (users with similar interests are likely to be similar to each other). The API returns the top 10 most similar users by each of these similarity measures along with the top 10 most similar users by an overall similarity measure which is calculated by taking the weighted average of the 3 similarity values –

*similarity\_overall = 0.5 \* similarity\_assessment + 0.2 \* similarity\_course\_views +*

* 1. *\* similarity\_interest*
* *similarity\_assessment* – cosine similarity between assessment scores grouped and normalized by assessment tag for each user,
* *similarity\_course\_views –* cosine similarity between user factors obtained using collaborative filtering (SVD) of course view times grouped and normalized by course for each user,
* *similarity\_interest –* cosine similarity between TF-IDF values of the user interest tags.

Cosine similarity is a common technique for measuring the similarity between vectors with continuous values. It is also better suited for high-dimensional vectors (as in this case) than some other measures like Euclidean distance.

Amongst the three attributes of the user used for calculating the similarity, both assessment scores and interests can be considered explicit features of the user, whereas the user features associated with course view times are more implicit in nature and therefore, a collaborative filtering algorithm (SVD) was used to extract the latent user factors from the course view times data.

The weights were chosen intuitively based on the degree of overlap between the users for the 3 properties. Assessment tags (54) have the least overlap, followed by the interest tags (748) and course IDs (5942). Less overlap implies lower similarity values, which should therefore be weighted more heavily. In a real-world scenario, these heuristics would need to be estimated more carefully by analyzing their effect using online metrics.

1. **We have provided you with a relatively small sample of users. At true scale, the number of users, and their associated behavior, would be much larger. What considerations would you make to accommodate that?**

The similarity calculations can be parallelized and therefore frameworks leveraging distributed systems like Spark and Kafka would be used to maintain a low latency of the API. A clustering algorithm can also be used to group similar users and therefore reduce the search space during runtime.

1. **Given the context for which you might assume an API like this would be used, is there anything else you would think about? (e.g. other data you would like to collect)**

Other information that could be valuable are user’s ratings or feedback for the courses. User demographic information like years of experience, age, current position, goals, etc. can also be helpful in identifying similar users and courses to recommend.