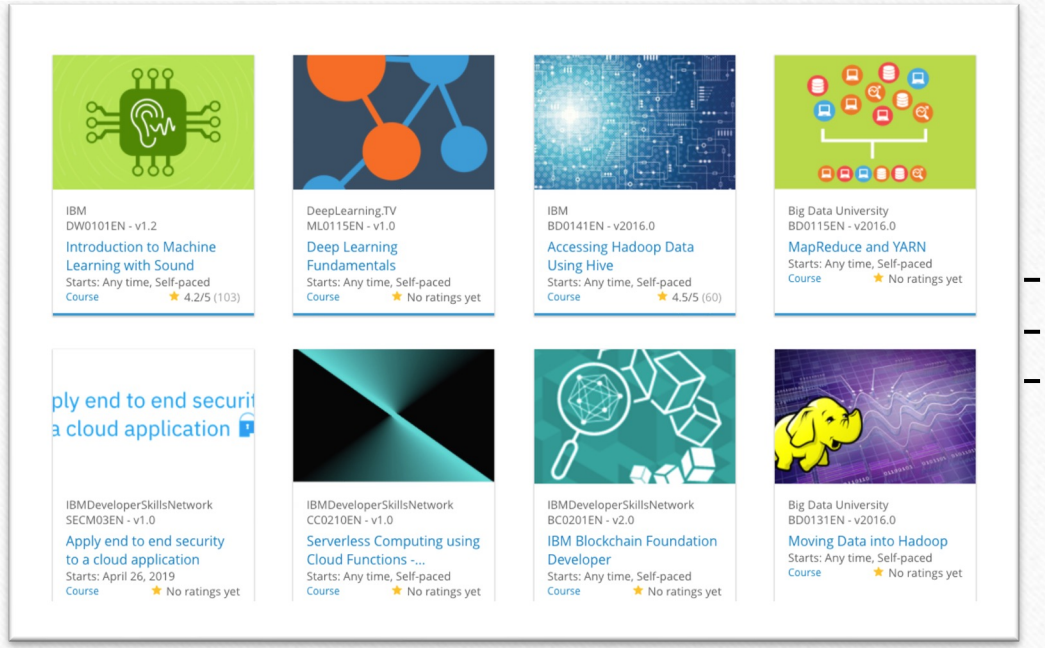
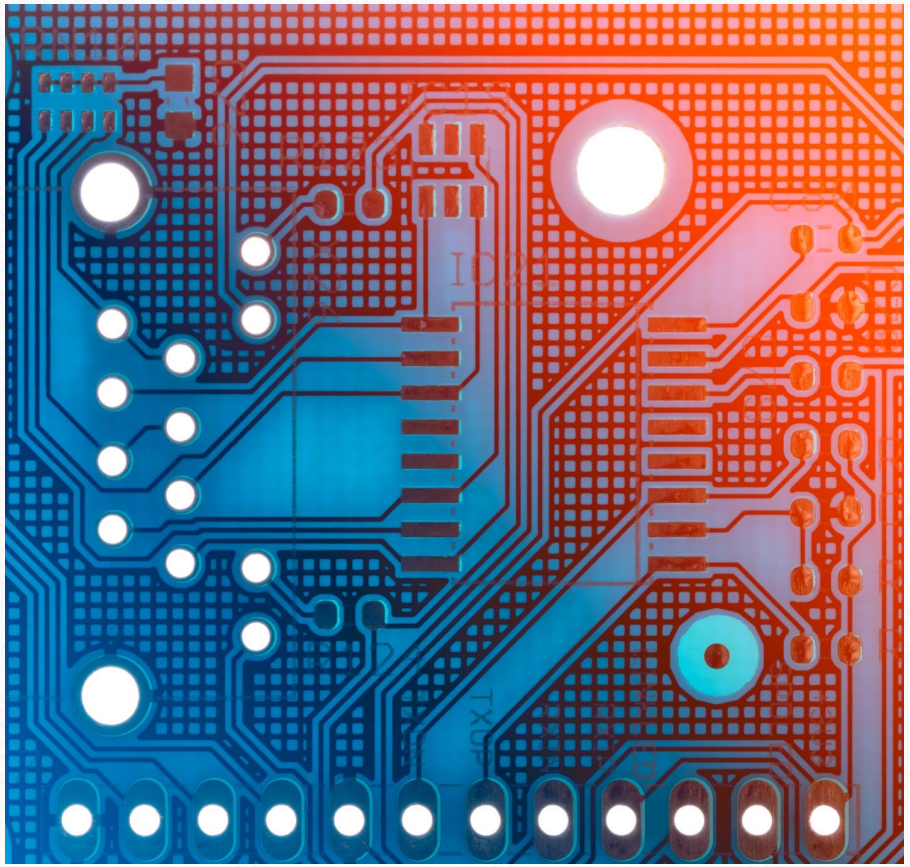
Build a Personalized Online Course Recommender System with Machine Learning

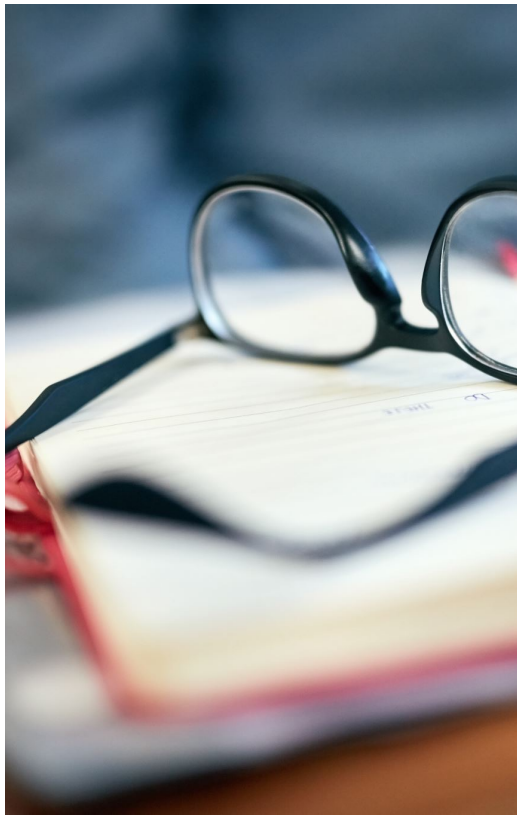
Dianne Liu
May-01, 2023



Outline



- Introduction and Background
- Exploratory Data Analysis
- Content-based Recommender System using Unsupervised Learning
- Collaborative-filtering based Recommender System using Supervised learning
- Conclusions
- Appendix

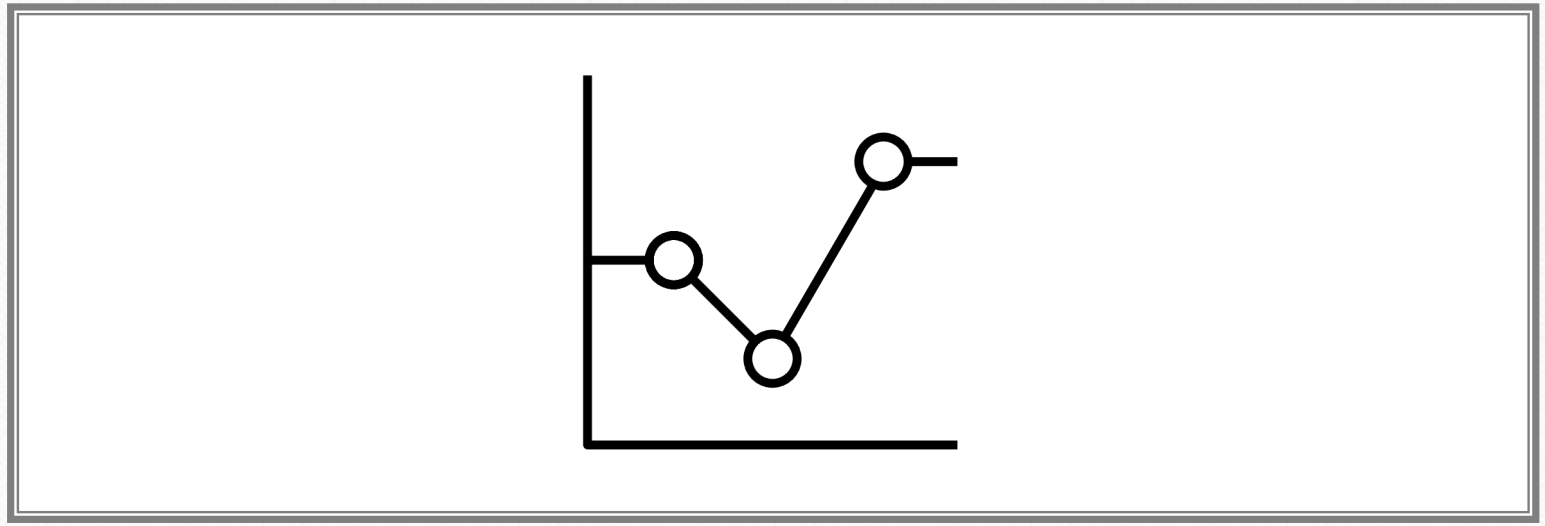


Introduction and Background

Improve learners' learning experience via helping them quickly find new interested courses and better paving their learning paths.

With more learners interacting with more courses via your recommender systems, your company's revenue may also be increased.

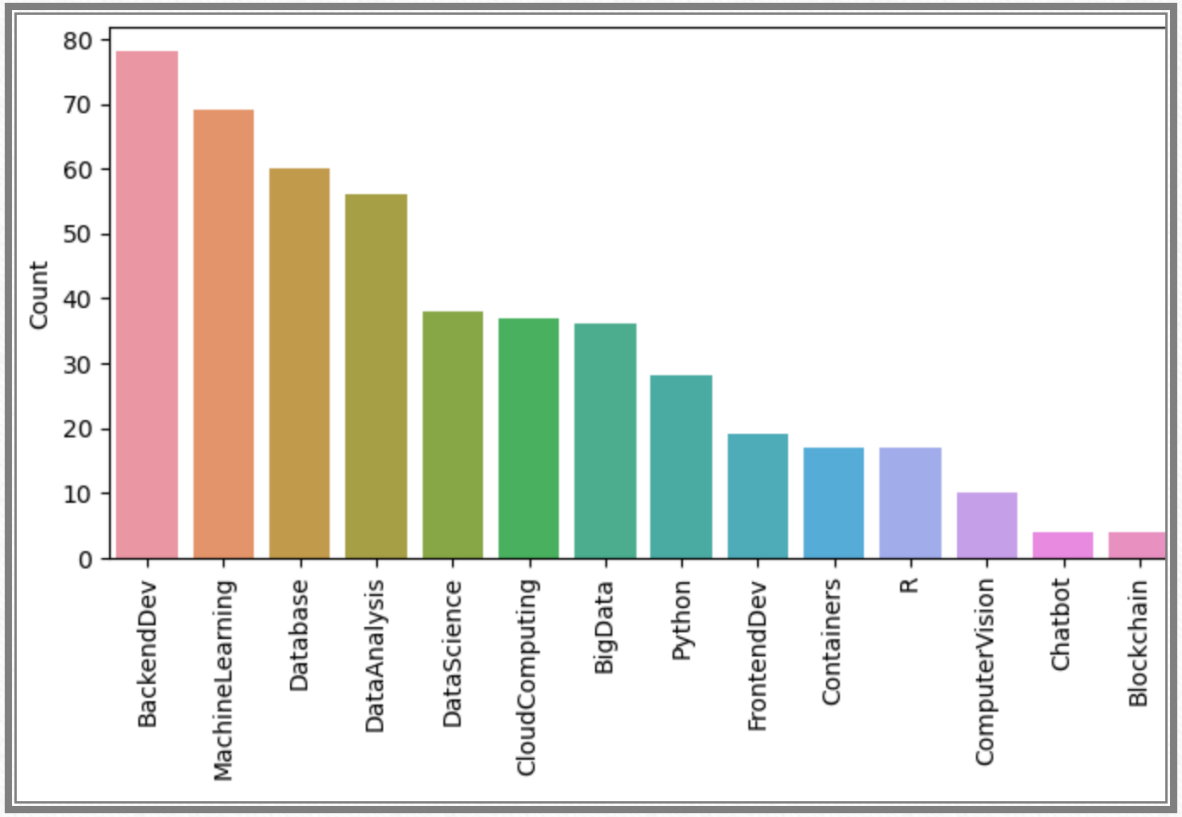
Mainly focus to explore and compare various machine learning models and find one with the best performance in off-line evaluations.



Exploratory Data Analysis

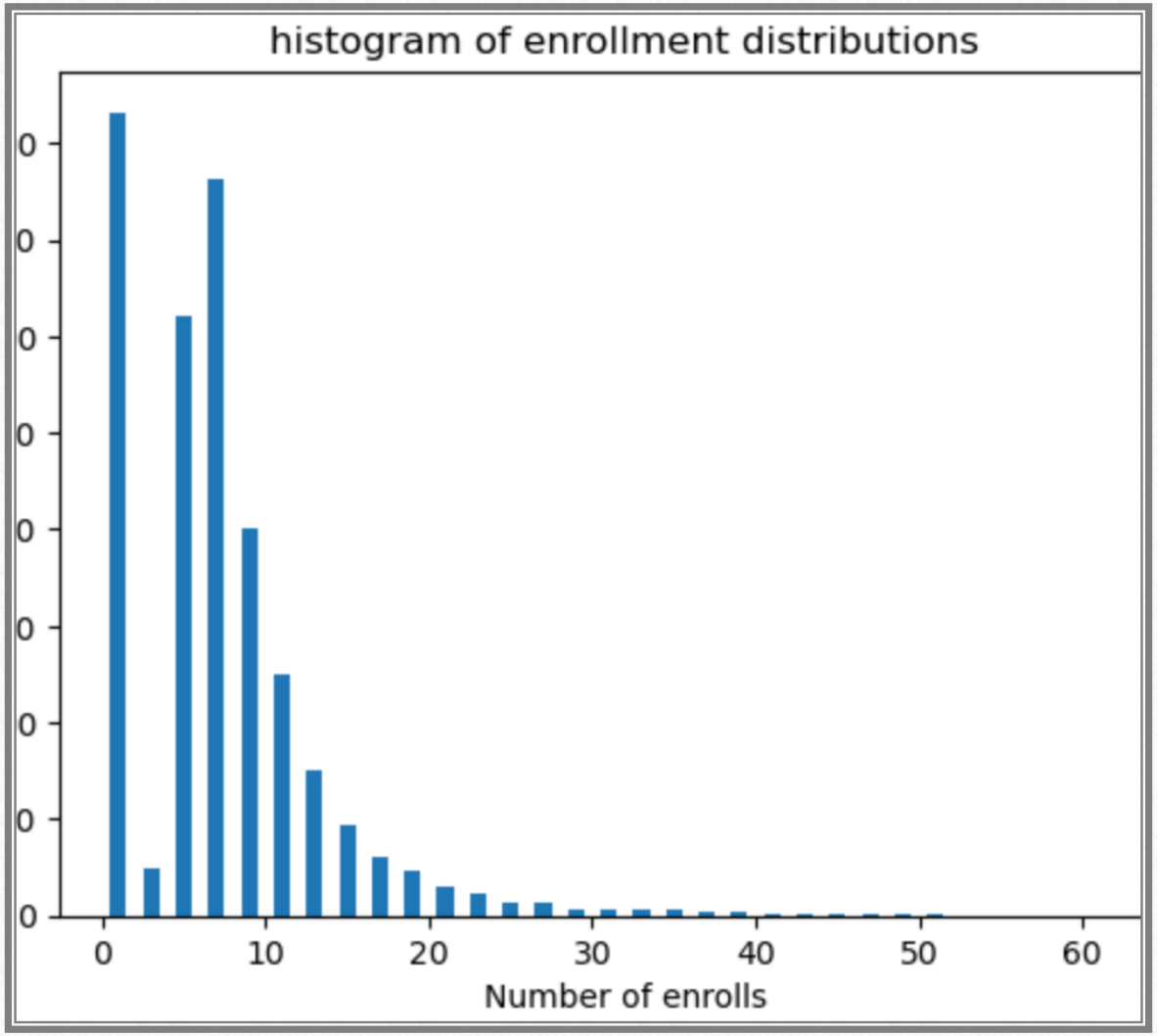
Course counts per genre

- This chart shows all courses genre and the course count of each genre.
- It is sorted by course count so that audience can easily find the most popular course genre



Course enrollment distribution

- This histogram chart shows the enrollment distributions, e.g., how many users rated just 1 item or how many rated 10 items, etc.



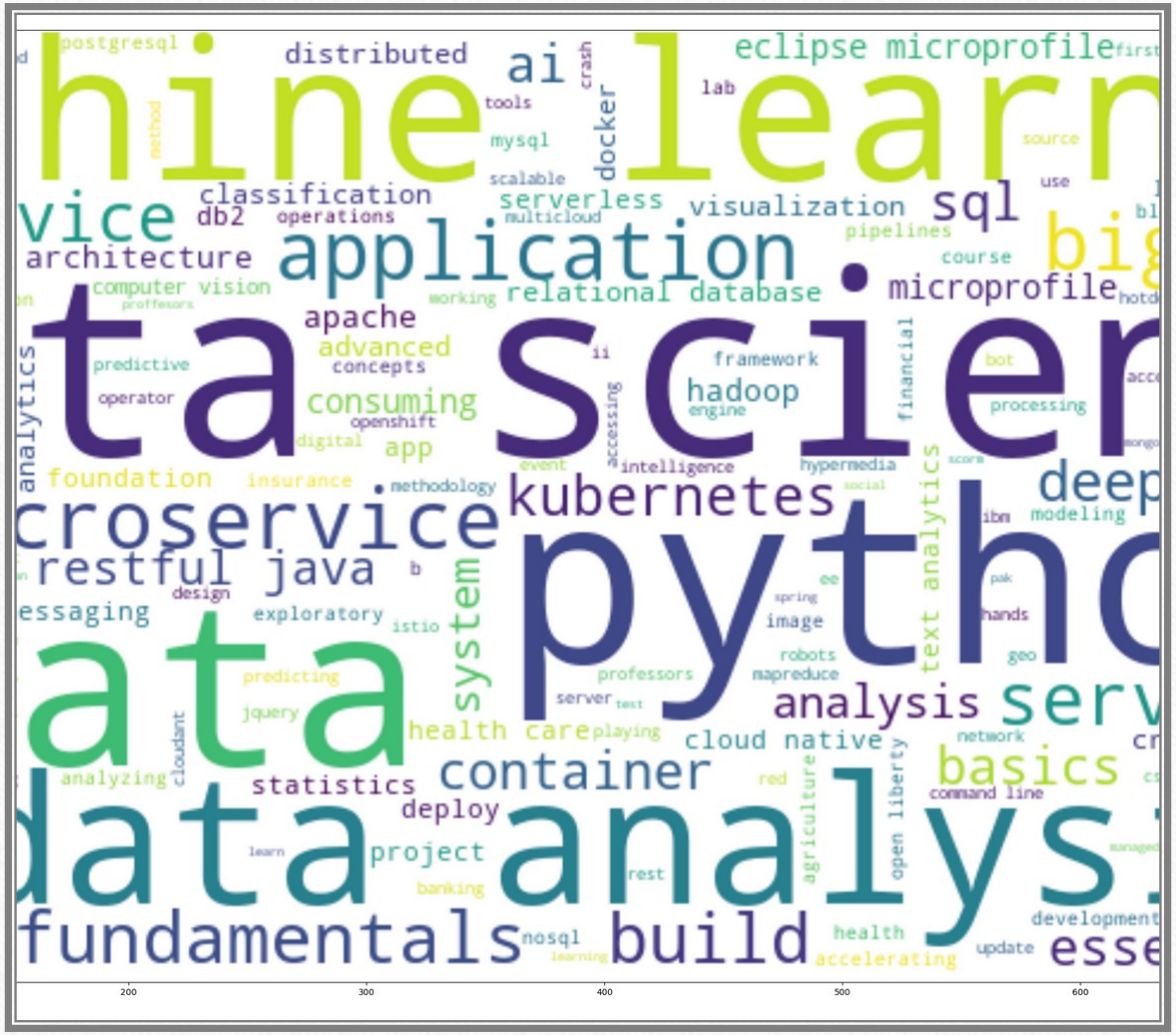
20 most popular courses

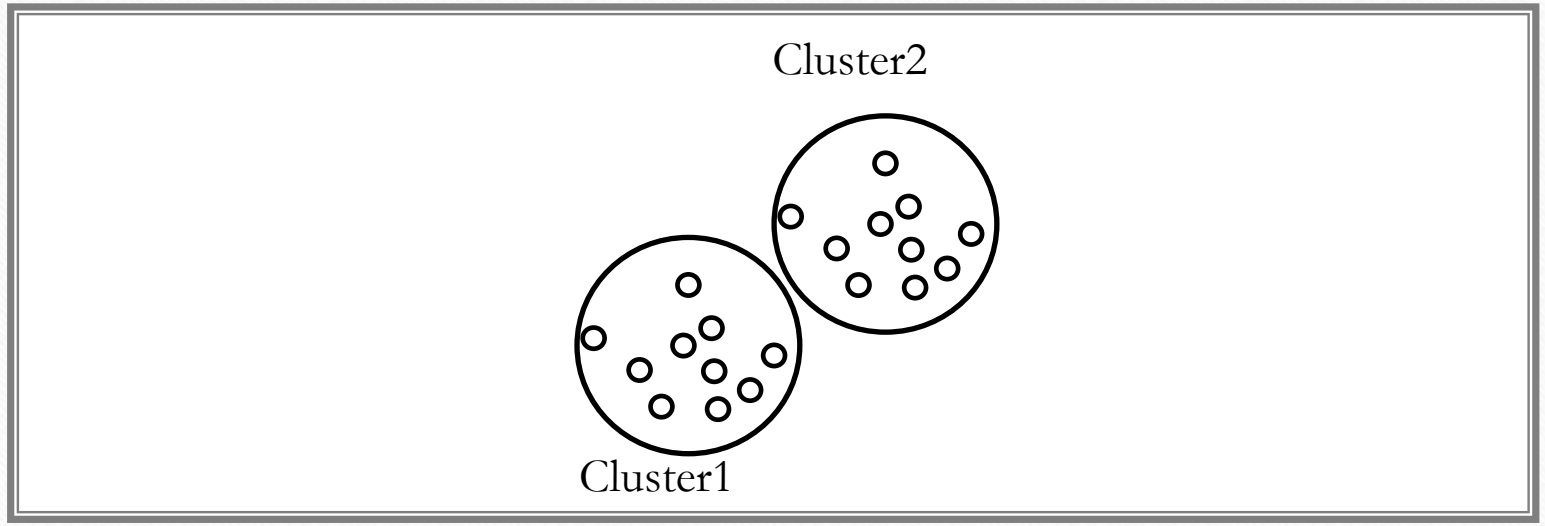
- The list shows most popular 20 courses, i.e., items with the most rating counts.

	COURSE_ID	Ratings
点击展开输出；双击隐藏输出		936
1	DS0101EN	14477
2	BD0101EN	13291
3	BD0111EN	10599
4	DA0101EN	8303
5	DS0103EN	7719
6	ML0101ENv3	7644
7	BD0211EN	7551
8	DS0105EN	7199
9	BC0101EN	6719
10	DV0101EN	6709
11	ML0115EN	6323
12	CB0103EN	5512
13	RP0101EN	5237
14	ST0101EN	5015
15	CC0101EN	4983
16	CO0101EN	4480
17	DB0101EN	3697
18	BD0115EN	3670
19	DS0301EN	3624

Word cloud of course titles

- The word count are generated from the course titles
- From the wordcloud, we can see many popular IT related keywords such as python, data science, machine learning, big data, ai, tensorflow, container, cloud, etc.
- By looking at these keywords, we should have a general understanding that the courses in the dataset are focused on demanding IT skills.





Content-based Recommender System using Unsupervised Learning

Overview for Content-based Recommender System

The content-based recommender system is highly based on the similarity calculation among items. The similarity or closeness of items is measured based on the similarity in the content or features of those items.

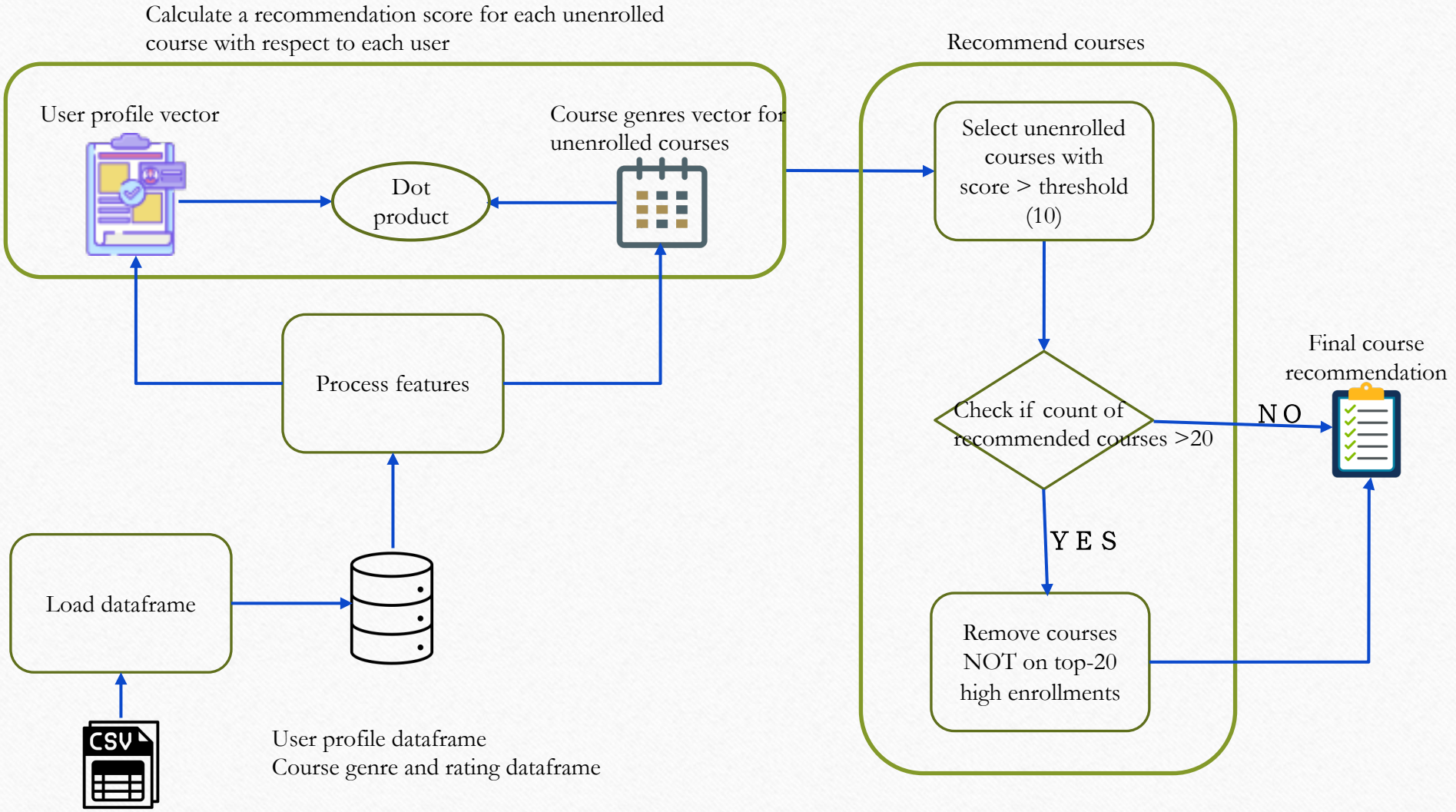
Next we are going to introduce using the unsupervised machine learning method to build a content-based course recommender system. The features we will use to compute the similarity will be:

- Course genres
- User interest
- BoW value (numeric feature vector transformed from course textual content)

Content-based recommender system - using user profile and course genres

- 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.
- Next we are going to:
  - Generate user profile based on course genre and rating.
  - Transform the user profile to be numeric feature vector
  - Compute the interest score that also represents the recommendation score by computing dot product between a specific course matrix and a single user's user profile vector
  - Recommend the new courses with high recommendation score to a user

Flowchart of content-based recommender system using user profile and course genres



Pipeline of content-based recommender system using user profile and course genres

Load

- **Load Dataframe**
  - Load course genre dataframe: contains course id, course title, course attributes
  - Load user profile dataframe: contains course interest for each user
  - Load test dataframe: contains users to whom we want to make recommendation

Process

- **Process features**
  - Create user profile vector based on course genre and rating
  - Create list of unenrolled courses for each user
  - With respective of each user, create course genre vector for each unenrolled course

Compute

- **Compute recommendation score**
  - Calculate recommendation score by using dot product between user profile vector for a single user and the course genre vector for a specific enrolled course

Recommend

- **Recommend courses**
  - Find the course with recommendation score greater than threshold (10) and add the course to the recommendation
  - Count recommended courses for each user. If the count is greater than 20, remove those courses with low count of enrollments, i.e. the enrollment count is NOT on top-20.

Evaluation results of user profile-based recommender system

Threshold for recommendation score = 10

Maxima count of recommended courses per user = 20

```
recommendations_size=res_df3.groupby('USER').size()
users_count=len(recommendations_size)
recommendations_avg=recommendations_size.iloc[0:users_count].values.sum()/len(recommendations_size)

print("Average Number of Recommended Courses per User:", "{:.2f}".format(recommendations_avg))
```

```
Average Number of Recommended Courses per User: 18.46
```

Question: On average, how many new/unseen courses have been recommended per user (in the test user dataset)?

Answer: 18.46

```
# The top 10 most recommended courses
recommendations_course_counts=res_df3.groupby('COURSE_ID').size().sort_values(ascending=False)
recommendations_course_counts[0:10]
```

Question: What are the most frequently recommended courses?

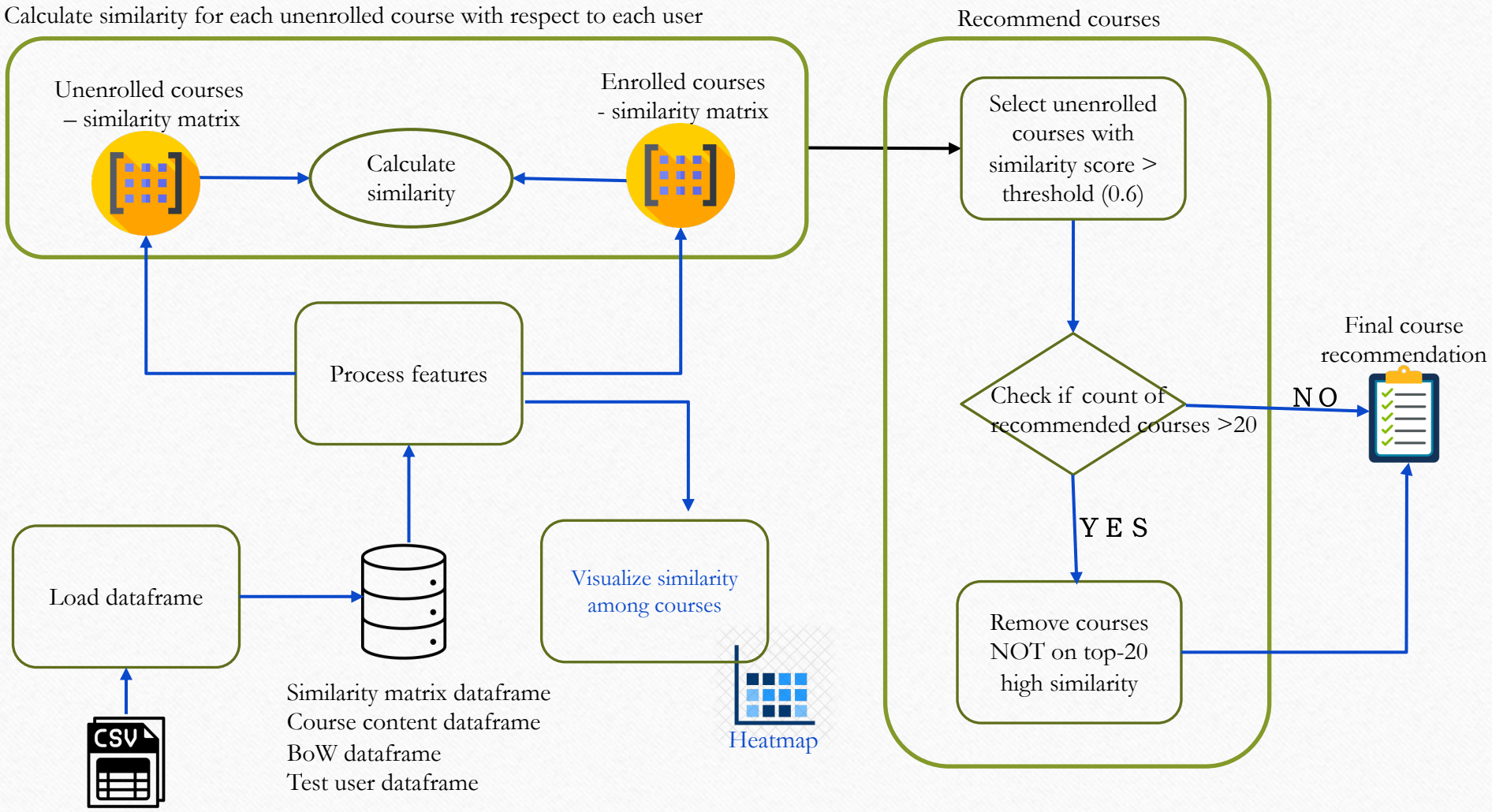
Return the top-10 commonly recommended courses across all users


```
COURSE_ID
RP0105EN      441
TMP0105EN     430
ML0122EN      426
SC0103EN      395
GPXX0IBEN     395
excourse21    380
excourse22    377
TA0106EN      376
BD0212EN      373
ML0101EN      359
```

Content-based recommender system - using course similarity

- The content-based recommender system is highly based on the similarity calculation among items. The similarity or closeness of items is measured based on the similarity in the content or features of those items. The course genres and users' profile are important features.
- In addition to course genre and users' profile, BoW value is another important type of feature in numeric form to represent course textual content.
- Next, we are going to apply the course similarities metric to:
  - Obtain the similarity between courses from a course similarity matrix
  - Use the course similarity matrix to find and recommend new courses which are similar to enrolled courses

Flowchart of content-based recommender system using course similarity



Pipeline of content-based recommender system using course similarity

Load

- **Load Dataframe**
  - Load the dataframe of pre-made course similarity matrix
  - Load the test dataframe: contains users to whom we will make recommendation of new courses
  - Load the dataframe of course content, contains user id and the user's enrolled courses' id, title, description
  - Load the dataframe of course BoW, contains course index, course id, tokens of each course, and the bow value of each token

Process

- **Process features**
  - Visualize the similarity matrix of the dataframe by heatmap. The heatmap shows many hot spots which indicate many courses are similar each other. Hence, it is possible to build the recommender system based on course similarity
  - For each user, create the list of enrolled courses and the list of unenrolled courses
  - Create mapping from course index to course id and from course id to course index, for convenience of query on similarity matrix

Compute

- **Compute similarity score**
  - For each enrolled and unenrolled course, find its 2 indices (row and column) based on course id and then based on its indices to query its similarity matrix
  - For each user, iterate each unenrolled course and compute its similarity to each enrolled course based on the course's similarity matrix.

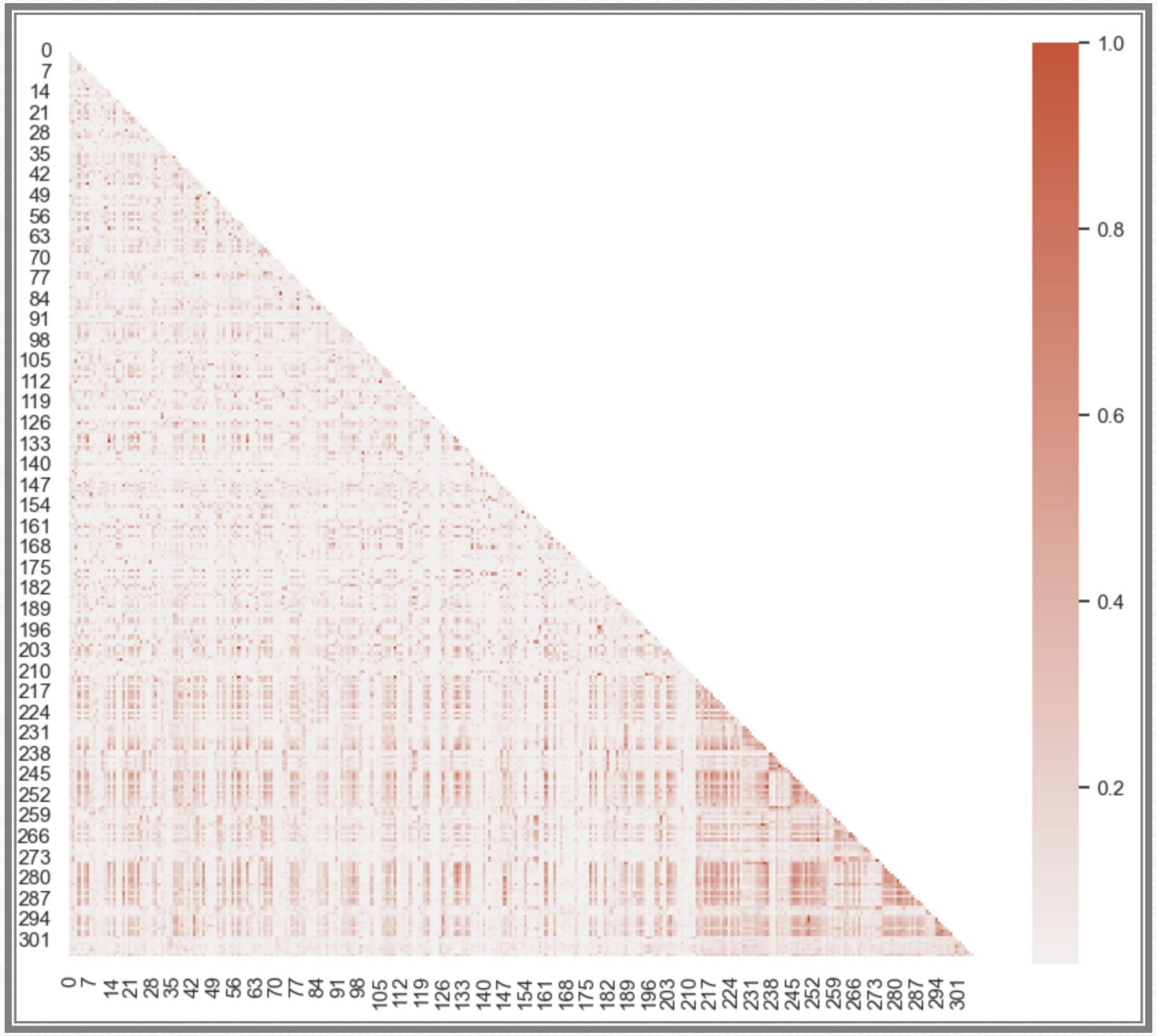
Recommend

- **Recommend courses**
  - Find the course with similarity score greater than threshold (0.6) and add the course to the recommendation
  - Count recommended courses for each user. If the count is greater than 20, remove courses with low similarity score, i.e. the similarity score is NOT on top-20.

Heatmap for metrics of course similarity

- We could use seaborn to visualize the similarity metric, and since it is symmetric, we can just show the triangular matrix (lower left).

- As we can see from the heatmap; there are many hot spots, which means many courses are similar to each other. Such patterns suggest that it is possible to build a recommender system based on course similarities.



Evaluation results of course similarity based recommender system

Threshold for similarity score for recommendation = 0.6

Maxima count of recommended courses per user = 20

```
recommendations_size=res_df2.groupby('USER').size()
users_count=len(recommendations_size)
recommendations_avg=recommendations_size.iloc[0:users_count].values.sum()/len(recommendations_size)
print("Average Number of Recommended Courses per User:", "{:.2f}".format(recommendations_avg))
```

```
Average Number of Recommended Courses per User: 11.19
```

Question: On average, how many new/unseen courses have been recommended per user (in the test user dataset)?

Answer: 11.19

```
# The top 10 most recommended courses
recommendations_course_counts=res_df2.groupby('COURSE_ID').size().sort_values(ascending=False)
recommendations_course_counts[0:10]
```

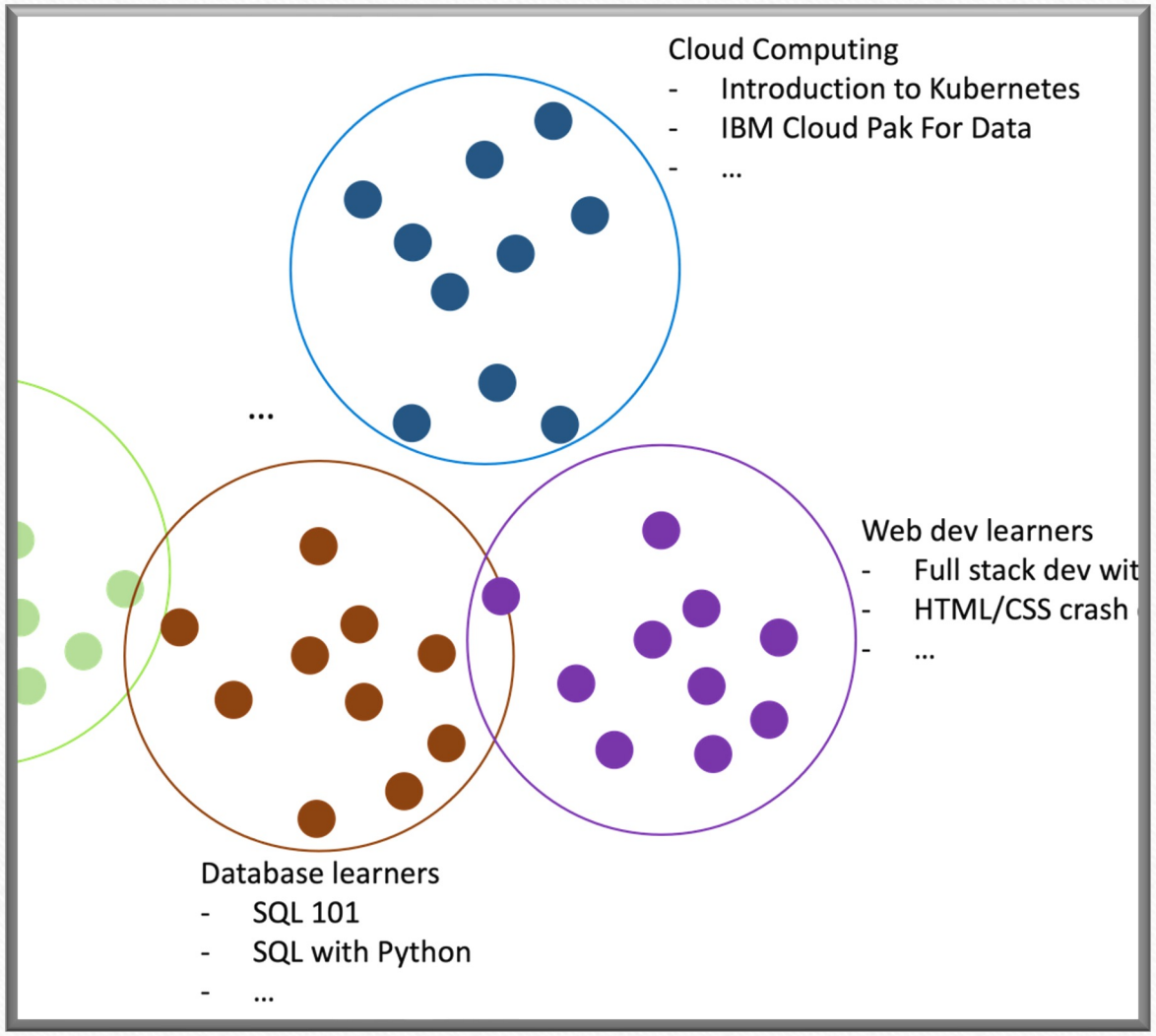
Question: What are the most frequently recommended courses?

Return the top-10 commonly recommended courses across all users →

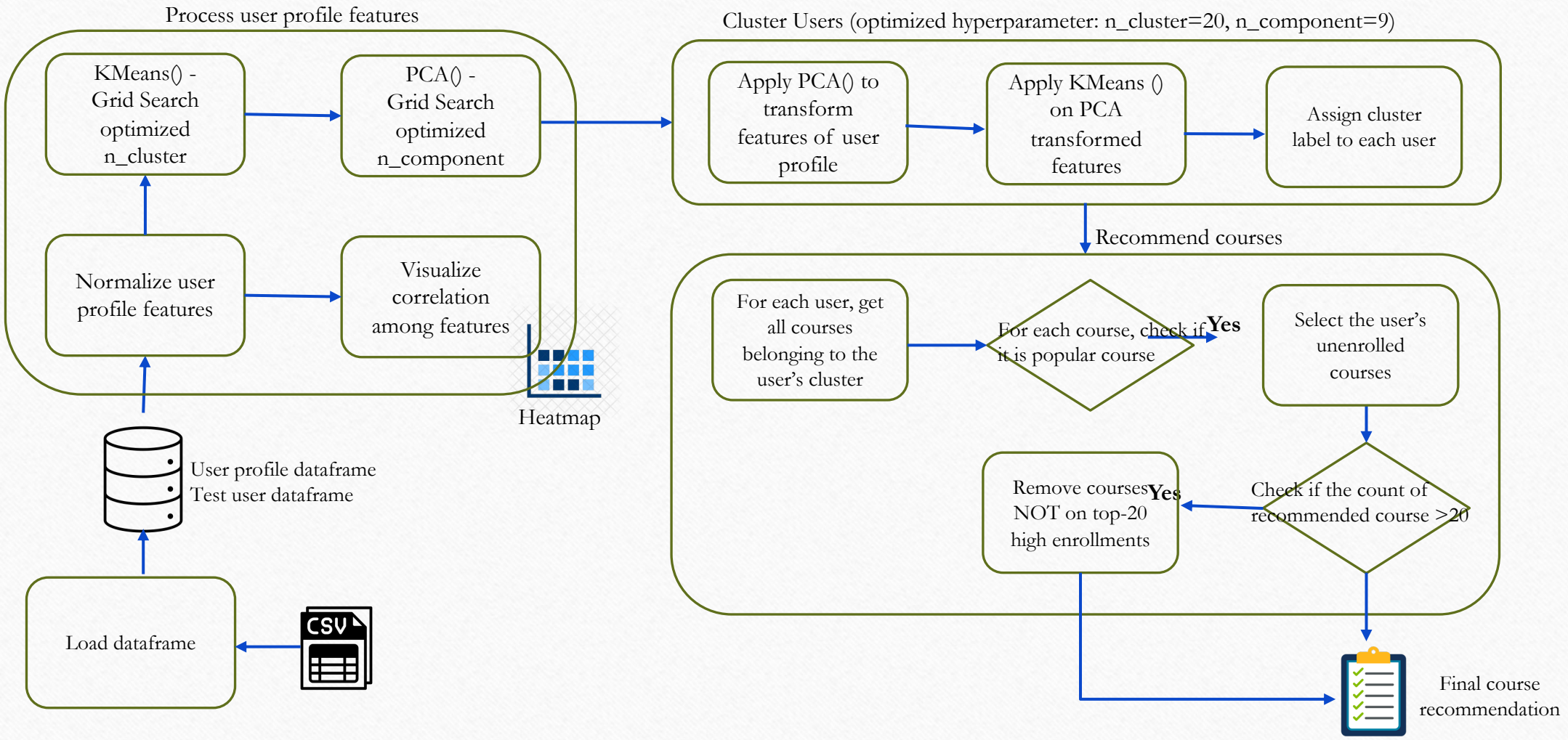
```
COURSE_ID
excourse62    575
excourse22    574
DS0110EN      561
excourse63    555
excourse72    551
excourse65    548
excourse74    539
excourse67    539
excourse68    502
BD0145EN      470
```

Clustering-based recommender system

- In clustering-based recommender system, we apply clustering algorithm such as K-means or DBSCAN to group users with similar learning interests.
- For each user group, we can count the most frequently enrolled courses, which are very likely to be most interested by the users in this group
- If we know a user belongs to this group, we may recommend these most enrolled courses to the user



Flowchart of clustering-based recommender system



Pipeline of clustering-based recommender system

Load

- **Load Dataframe**
  - Load the dataframe of user profile: contains each user's interested areas and ratings
  - Load the test dataframe: contains users to whom we will make recommendation of new courses

Process

- Process features
  - Nomalize the user profile features using StandardScalere()
  - Visualize the correlation among user profile features
  - Apply GridSearch to get the optimized hyper parameters: n_cluster for Kmeans() and n_component for PCA()

Cluster

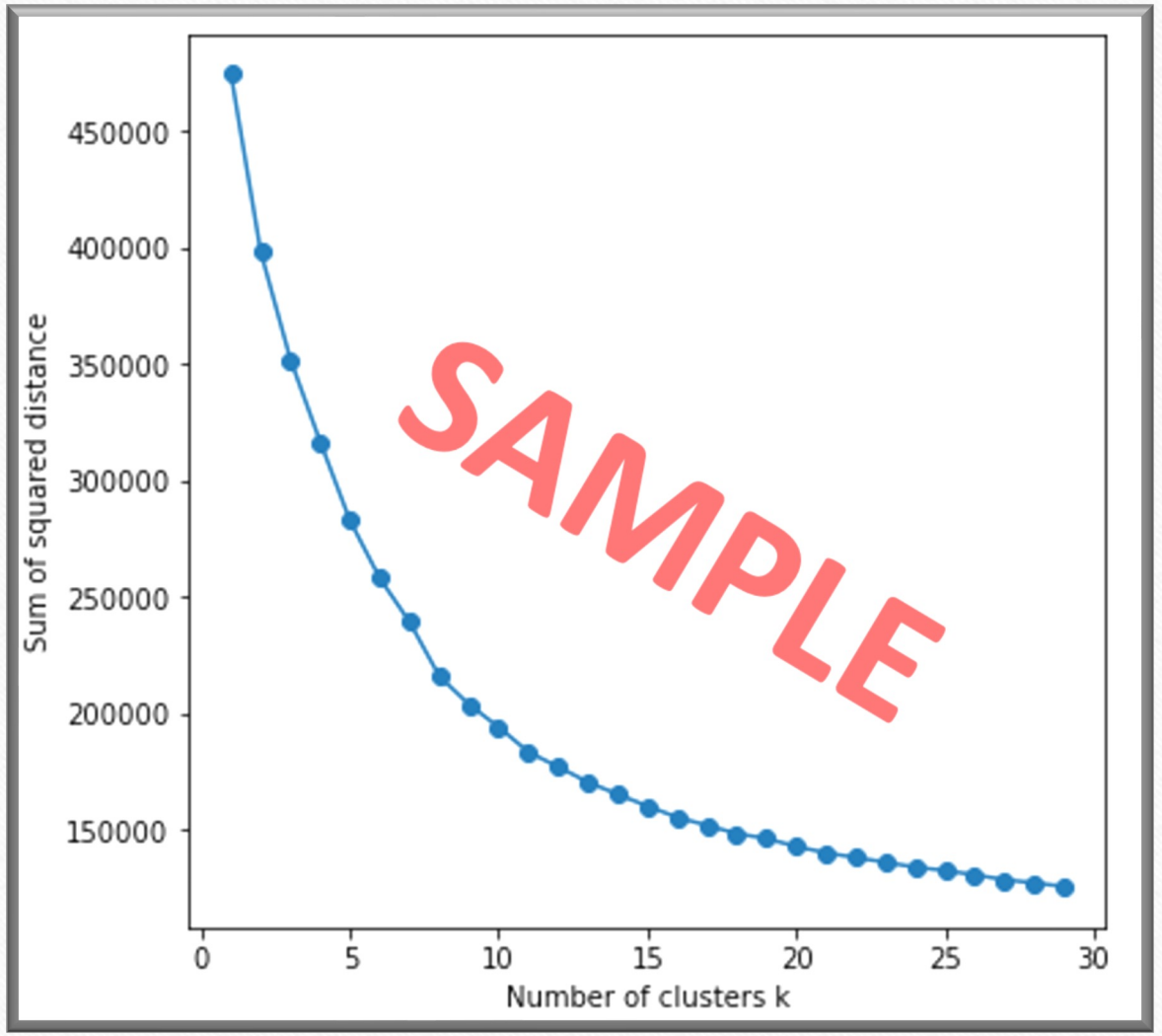
- **Cluster users**
  - Apply PCA(n_component=9) algorithm to transformed user profile features to reduce original 14 features to 9 features
  - Apply Kmeans(n_cluster=20) on PCA transformed features to group users into 20 clusters
  - Assign each user with a cluster label.

Recommend

- **Recommend courses**
  - For each user, get all courses within the user's cluster. Compare all the courses with the user's enrolled courses to get the user's unenrolled courses
  - Check if the user's enrolled courses are popular courses (count of enrollments >=150). If yes, add the courses to the recommendation.
  - Check if the count of recommendation is greater than 20. If yes, removed the courses NOT on top-20 high enrollments from recommendation.

Find the optimized hyperparameter for Cluster()

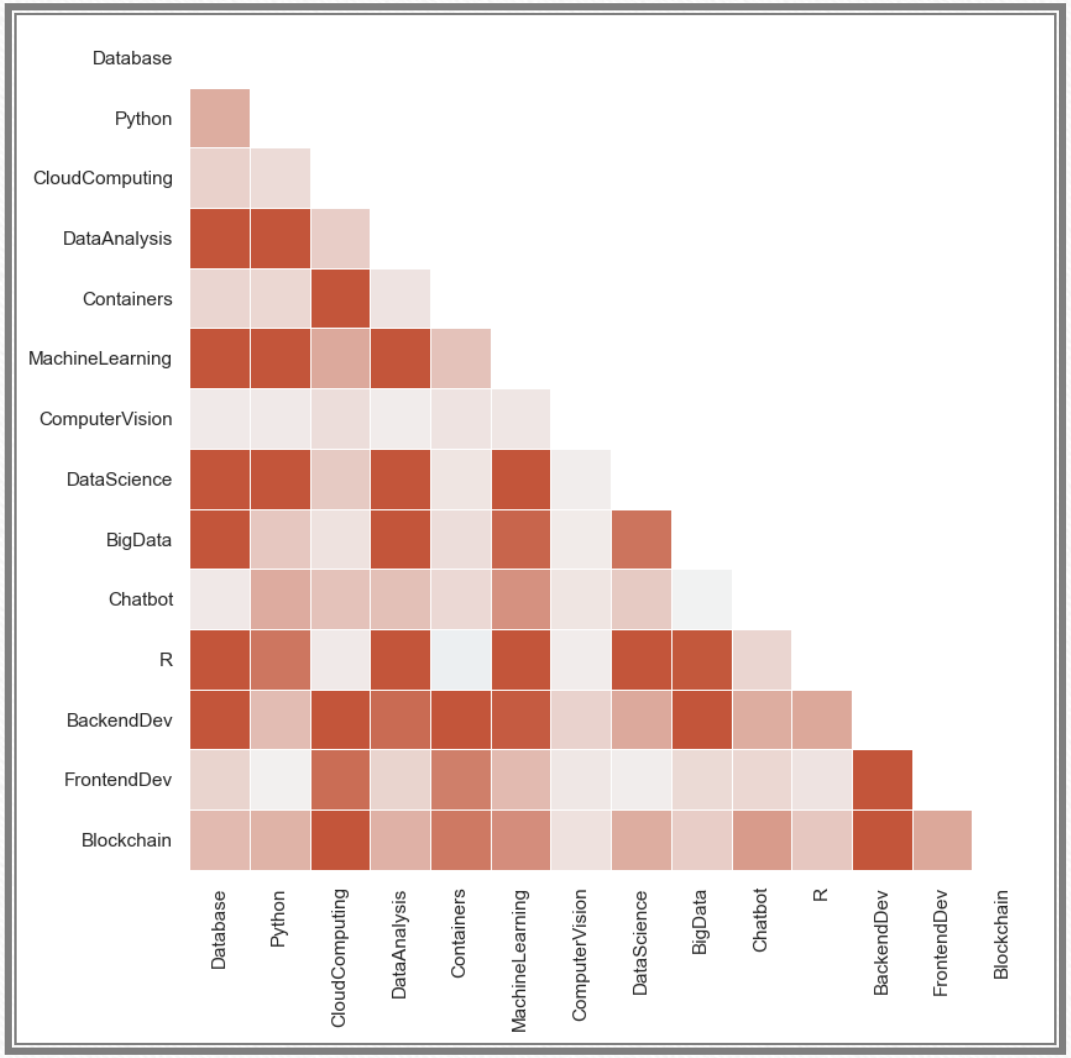
- We use KMeans() algorithm to perform clustering on the user profile feature vectors.
- Before we perform KMeans(), we can apply GridSearch to find its optimized hyperparameter, i.e., number of cluster, n_cluster.
- Grid search a list of candidates and find the one with the best or optimized clustering evaluation metrics such as minimal sum of squared distance.
- From the elbow plot, we can see the point 20 is where the metric starting to be flatten, which indicates the optimized number of clusters.



Heatmap for covariance matrix of the user profile feature vectors

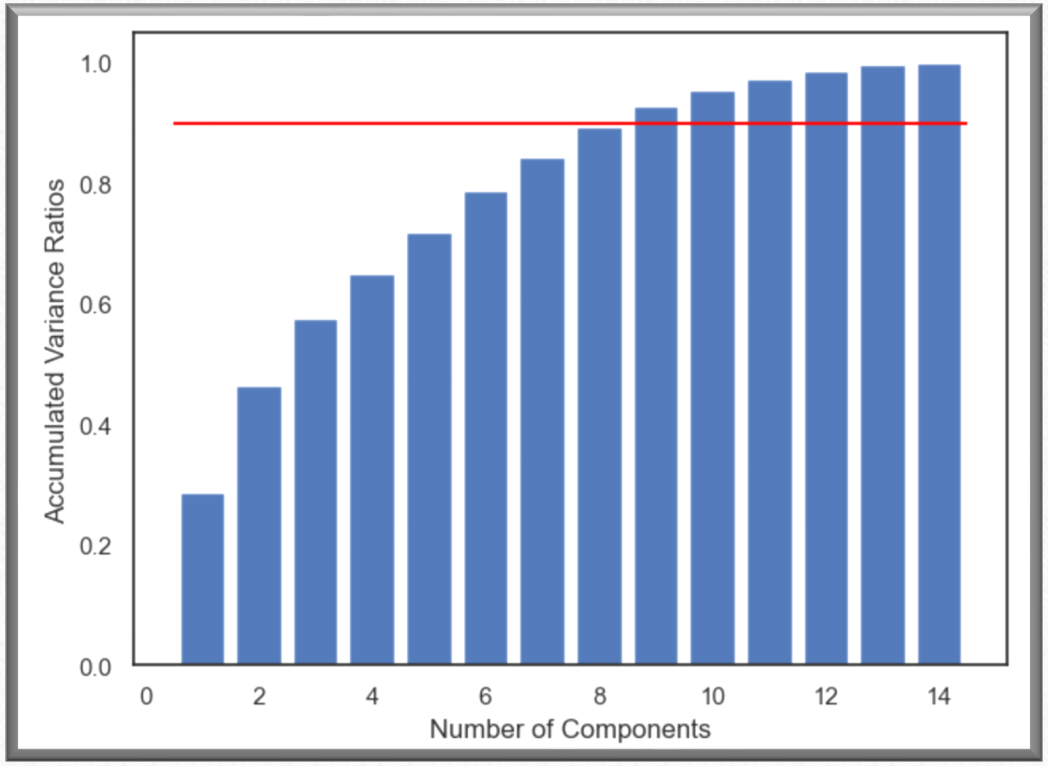
- We plot a covariance matrix of the user profile feature vectors with 14 features. we can observe that some features are actually correlated.

- Such covariances among features may indicate that we can apply PCA to find its main components. If we only keep the independent main components, then we can reduce the dimensions of our user profile feature vectors.



Find the optimized hyperparameter for PCA()

- Apply PCA() to find the main components in user profile feature vectors and see if we can reduce its dimensions by only keeping the main components.
- When calling the PCA() class, there is an import argument n_component which indicates how many components we want to keep in the PCA result.
- We apply Grid search on a list of argument candidates in range(1,15) and then calculate the ratio of the accumulated variance for each candidate.
- If the accumulated variances ratio of a argument candidate (i.e. n_components) is larger than threshold 0.9, then we can say the transformed n_components could explain about 90% of variances of the original data variance. Hence is can be considered as an optimized components size.
- We select n_component = 9, due to the minimal ratio > 0.9



Evaluation results of clustering-based recommender system

For Kmeans(), n_cluster=20
For PCA(), n_component=9
Maximum count of recommendation=20

```
# Average count of recommendation per user
res['count'] = [1] * len(res)
user_avg_count=res.groupby(['user']).agg(total_count = ('count','sum')).reset_index()
print("Average count of recommendation per user:",sum(user_avg_count['total_count'])/len(test_user_ids))
```

```
Average count of recommendation per user: 11.672
```

Question: On average, how many new/unseen courses have been recommended per user (in the test user dataset)

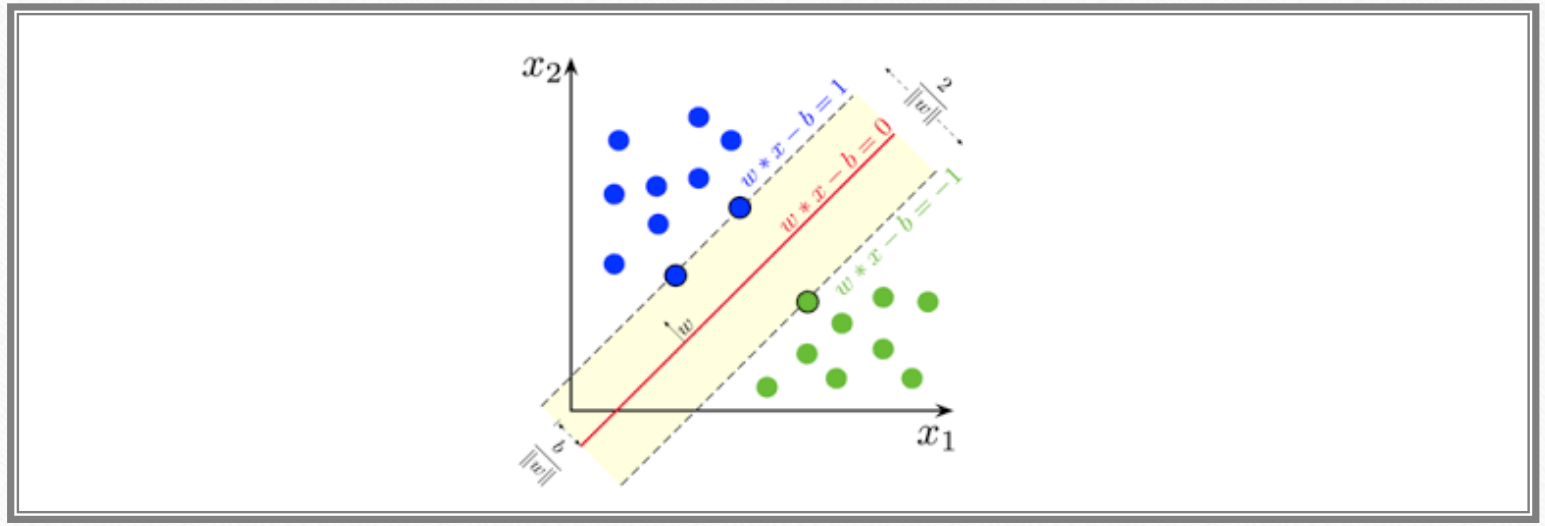
Answer: 11.67

```
# Get the top-10 most recommended courses
recommendation_count=res.groupby(['recommendation']).agg(total_count = ('count','sum')).reset_index()
recommendation_count.sort_values(by=['total_count'],ascending=False, inplace=True, ignore_index=True)
print("The top-10 recommended courses:")
print(recommendation_count[:10])
```

Question: What are the most frequently recommended courses?

Return the top-10 commonly recommended courses →

	recommendation	total_count
0	ST0101EN	800
1	RP0101EN	773
2	CC0101EN	741
3	DV0101EN	740
4	ML0115EN	732
5	BD0115EN	722
6	CO0101EN	719
7	DS0105EN	719
8	ML0101ENv3	701
9	BD0211EN	697

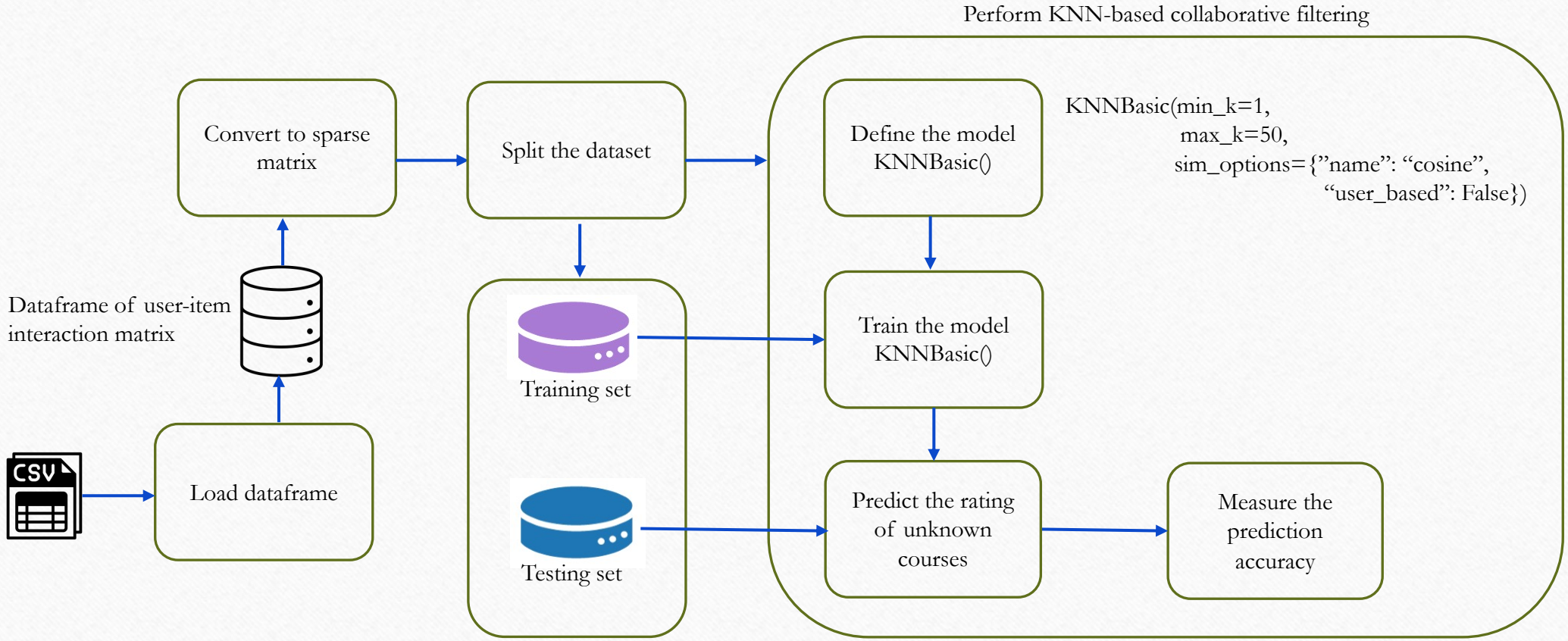


Collaborative-filtering Recommender System using Supervised Learning

Overview for collaborative-filtering recommender system

- Collaborative filtering is probably the most commonly used recommendation algorithm, the following are two main types of methods which work similarly:
  - **User-based** collaborative filtering is based on the user similarity or neighborhood
  - **Item-based** collaborative filtering is based on similarity among items
- For most collaborative filtering-based recommender systems, the main dataset format is a 2-D matrix called the **user-item interaction matrix.** In the matrix, each row vector represents the rating history of a user and each column vector represents the users who rated the item.
  - For user-based collaborative filtering, we determine if two users are similar, we can simply calculate the similarities between their row vectors in the interaction matrix. Then based on the similarity measurements, we can find the k nearest neighbor as the similar users.
  - For item-based collaborative filtering, we look at the user-item matrix vertically and determine if the items (courses) are similar. If two courses are enrolled by two groups of similar users, then we could consider the two items are similar and use the known ratings from the other users to predict the unknown ratings by which we may determine if a course can be recommended to a specific user.

Flowchart of KNN-based recommender system



Pipeline of KNN-based recommender system

Load

- **Load Dataframe**
  - Load the dataframe of a user-item (learn-course) interaction matrix, contains 3 columns: user id (learner), item id (course), and rating (enrollment mode)
  - Convert the original form of matrix to a sparse matrix using **pivot**

Split

- **Split the data**
  - Split the data into training set and test set. Size of test set = 0.3. Training set will be used for training algorithm. Testing set will be used for predict rating.

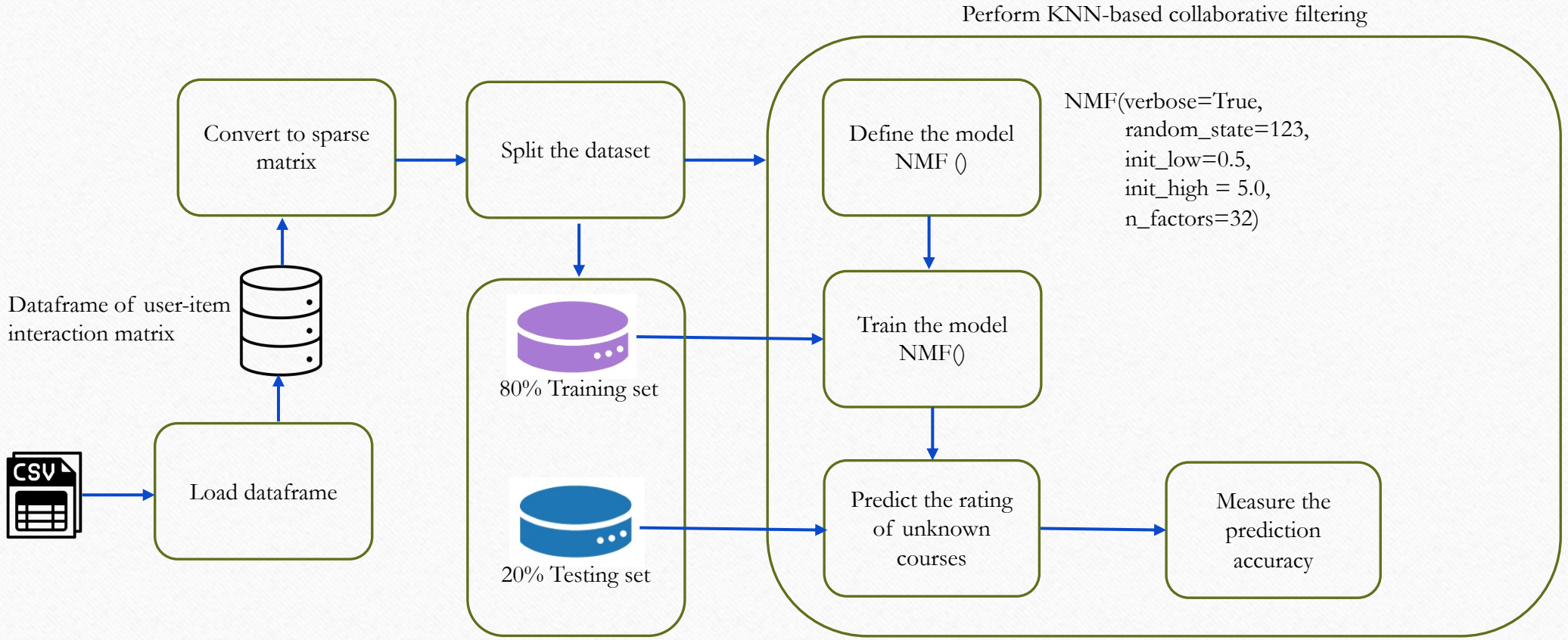
Train

- **Train the model KNNBasic() provided by scikit-surprise library, which is defined as follows:**
  - Compute the cosine similarity between items (item-based)
  - Maximum number of neighbors = 50, minimum number of neighbors = 1

Evaluate

- **Predict the rating and evaluate result**
  - Predict rating of unknown items based on the aggregation of the user's K nearest neighbor's ratings
  - Measure prediction accuracy by computing RMSE (root mean square error). accuracy.rmse(predictions) = 0.1943

Flowchart of NMF-based recommender system



Pipeline of NMF-based recommender system

Load

- **Load Dataframe**
  - Load the dataframe of a user-item (learn-course) interaction matrix, contains 3 columns: user id (learner), item id (course), and rating (enrollment mode)
  - Convert the original form of matrix to a sparse matrix using **pivot**

Split

- **Split the data**
  - Split the data into training set and test set. Size of test set = 0.3. Training set will be used for training algorithm. Testing set will be used for predict rating.

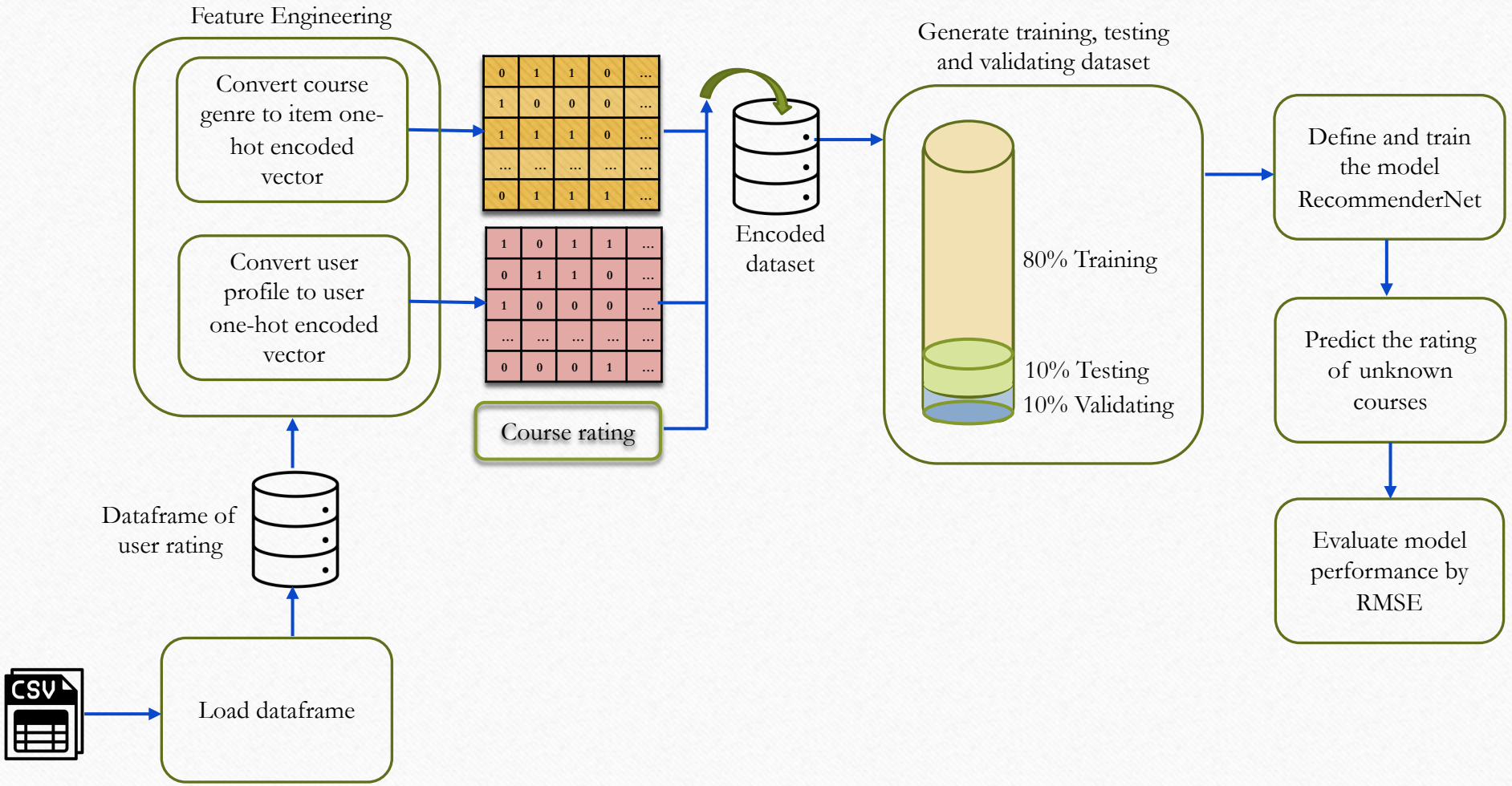
Train

- **Train the model NMF() provided by scikit-surprise library, which is defined as follows:**
  NMF(verbose=True, random_state=123, init_low=0.5, init_high = 5.0, n_factors=32)

Evaluate

- **Predict the rating and evaluate result**
  - Predict rating
  - Measure prediction accuracy by computing RMSE (root mean square error). accuracy.rmse(predictions) = 0.1924

Flowchart of Neural Network Embedding based recommender system



Pipeline of Neural Network Embedding based recommender system

Load

- **Load and process dataframe**
  - Load the rating dataset, contains 3 columns: user id (learner), item id (course), and rating (enrollment mode)

Process

- Process features
  - Convert the actual user ids and item ids into integer indices for tensorflow to creating 2 one-hot encoding vectors, one for users, and another for items.

Split

- **Split the data**
  - Split the data into 3 sets for training on 80% of the data, validating on 10% o the data, and testing on remaining 10% of the data

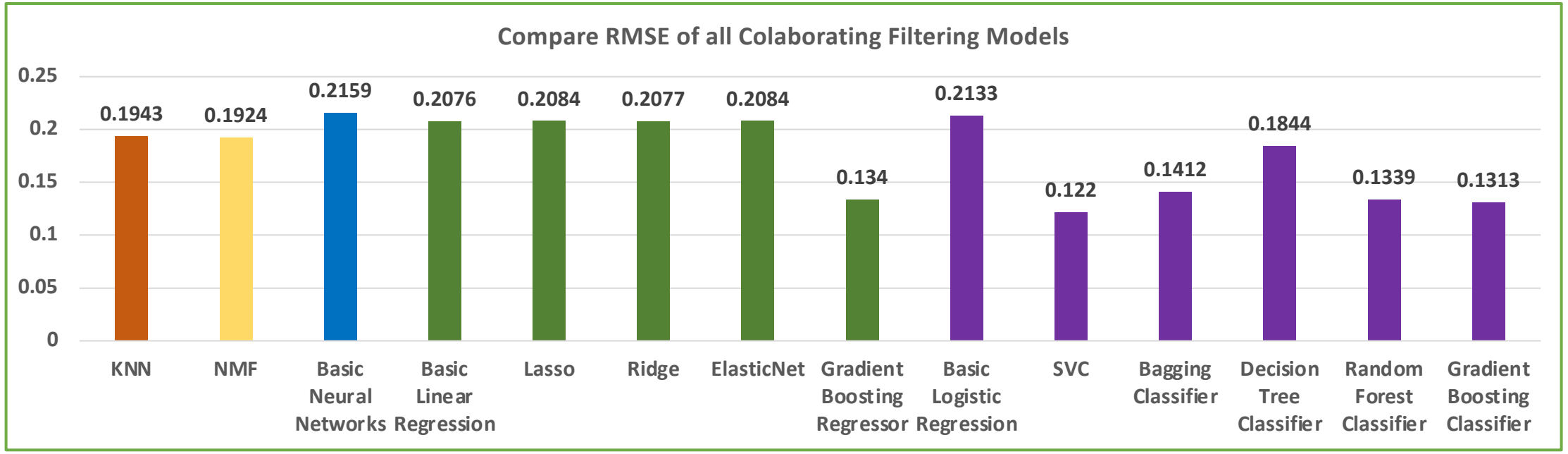
Train

- **Train the model**
  - Define the mode RecommenderNet() as a subclass inheriting the tensorflow.keras.Model super class. The input of the model will be the users and items one-hot encoding vectors. The output will be the predicted course rating.
  - Call model.compile() method to set up the loss and optimizer and metrics for the model training as follows:
    - optimizer =SGD(learning_rate=.003), loss = "MeanSquaredError", metrics=["RootMeanSquaredError"]
  - Call model.fit() method to train the model.

Evaluate

- **Evaluate the prediction accuracy of the model**
  - Call model.evaluate() method to evaluate the model: RMSE = 0.2159

Compare the performance of collaborative-filtering models



- The bar chart above shows the performance measured by RMSE (Root Mean Squared Error) for various collaborative filtering models built to predict the course rating in this analysis.
- The colors of the bars in the chart above represent different types of models. For example, green bars present performance of regression-based models, and purple bars present the performance of classification-based models
- The closer RMSE is to 0, the more accurate the model is. Hence, **the SVC classification model is considered to the best model in this analysis.**

Conclusions

Content-based recommender using unsupervised algorithms

- In this analysis, I have built 3 types of content-based recommender systems using various unsupervised algorithms.
- The 3 types of content-based recommender systems results in different set of top-10 recommended courses.
- I would consider next to investigate further and assess which type could work better than the others for recommendation

Course-similarity based course recommender

COURSE_ID	
excourse62	575
excourse22	574
DS0110EN	561
excourse63	555
excourse72	551
excourse65	548
excourse74	539
excourse67	539
excourse68	502
BD0145EN	470

User-profile based course recommender

COURSE_ID	
RP0105EN	441
TMP0105EN	430
ML0122EN	426
SC0103EN	395
GPXX0IBEN	395
excourse21	380
excourse22	377
TA0106EN	376
BD0212EN	373
ML0101EN	359

Clustering based course recommender

	recommendation	total_count
0	ST0101EN	800
1	RP0101EN	773
2	CC0101EN	741
3	DV0101EN	740
4	ML0115EN	732
5	BD0115EN	722
6	CO0101EN	719
7	DS0105EN	719
8	ML0101ENv3	701
9	BD0211EN	697

Collaborative-filtering recommender system

- Various types of collaborative filtering-based recommender systems were built using different types of algorithms in this analysis as below:
  - KNN (K Nearest Neighbor)
  - NMF (**Non-negative Matrix Factorization)**
  - Neural Networks
  - Regression with embedding features
  - Classification with embedding features
- Evaluating the performance of the models by RMSE metrics, our findings are as follows:
  - Among total 14 models created in this analysis, the classification algorithm SVC works the best on prediction of course rating.
  - In general, the classification algorithms works better than the other types of algorithms (i.e. regressiom, neural networks, NMF, KNN)
  - Among the 5 regression models created in this analysis, after tuning the hyperparameters using GridSearchCV method, the algorithm Gradient Boosting Regressor works significantly better than the other regression models in prediction of course rating

Next Step

My interest for next step is to build a movie recommender system and pass through each required step same as described in this analysis to find out the best movie recommender model

The features I would like to collect will include not only the audience's interest and rating on movies, but also the audience's features such as gender, age, education, profession, country …etc. I think these features would also affect the audience's movie selection and hence the recommender models should include them as part of input for training and testing.

Appendix

ALL MY NOTEBOOK FOR THIS CAPSTONE PROJECT IN GITHUB: LINK

CURRENT REPORT IN GITHUB: