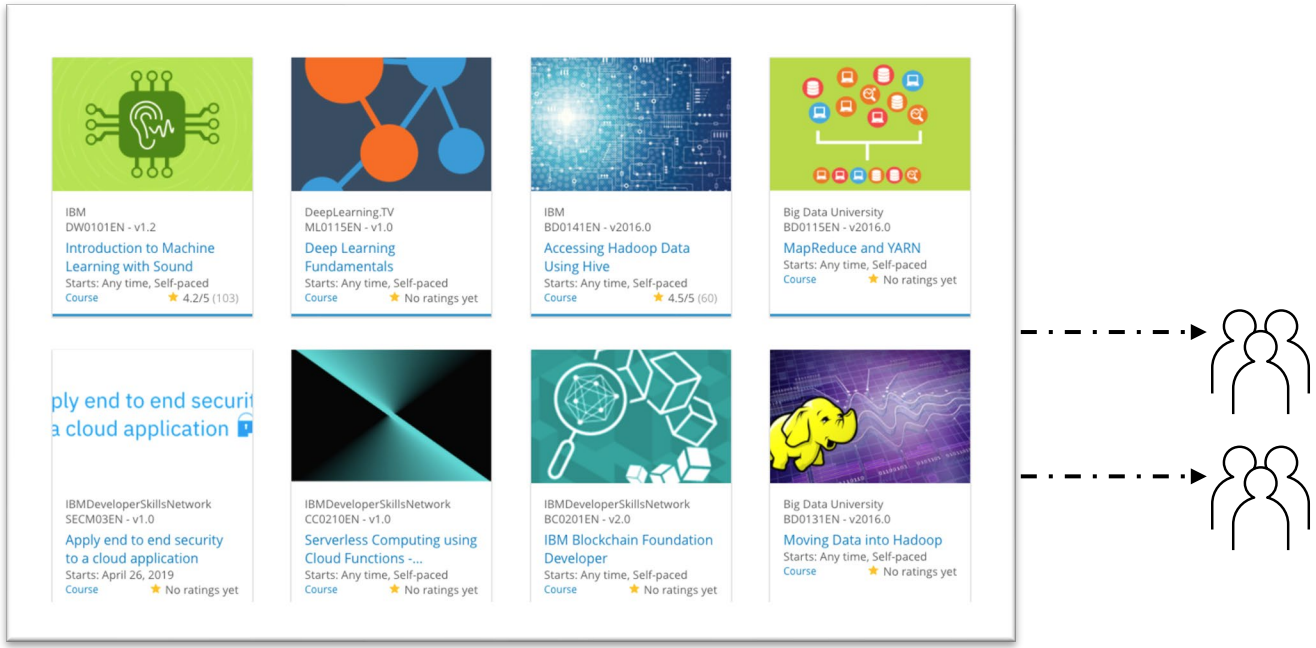
Build a Personalized Online Course Recommender System with Machine Learning

Swetha Konduru

17/02/2024



Outline

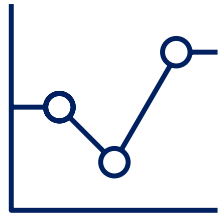
- Introduction and Background
- Exploratory Data Analysis
- Content-based Recommender System using Unsupervised Learning
- Collaborative-filtering based Recommender System using Supervised learning
- Conclusion
- Appendix

Introduction

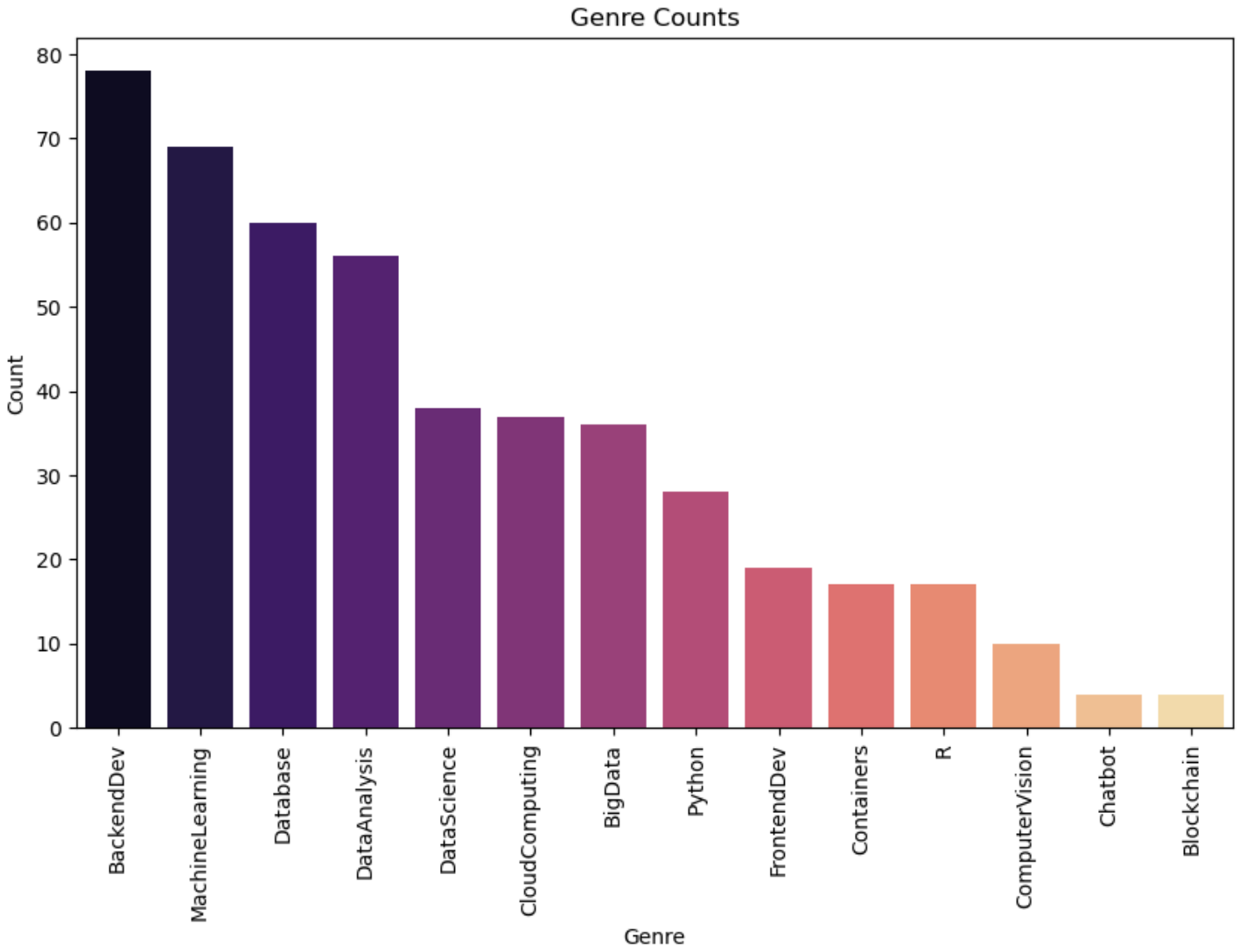
Personalized Online Course Recommender System with Machine Learning is a tool that uses machine learning to recommend online courses based on a user's behavior and preferences. It adapts to the user's learning progress, uses collaborative and content-based filtering techniques, and can combine these methods to improve recommendation accuracy.

- A course recommendation system will help in:
  - Finding better courses
  - Finding courses that well suits each person's interests
  - The aim is to find the best courses to recommend to users based on their interests and the courses they are enrolled in.
  - Here are some key features:
    - Adaptive Learning
    - Collaborative Filtering
    - Content-Based Filtering

Exploratory Data Analysis

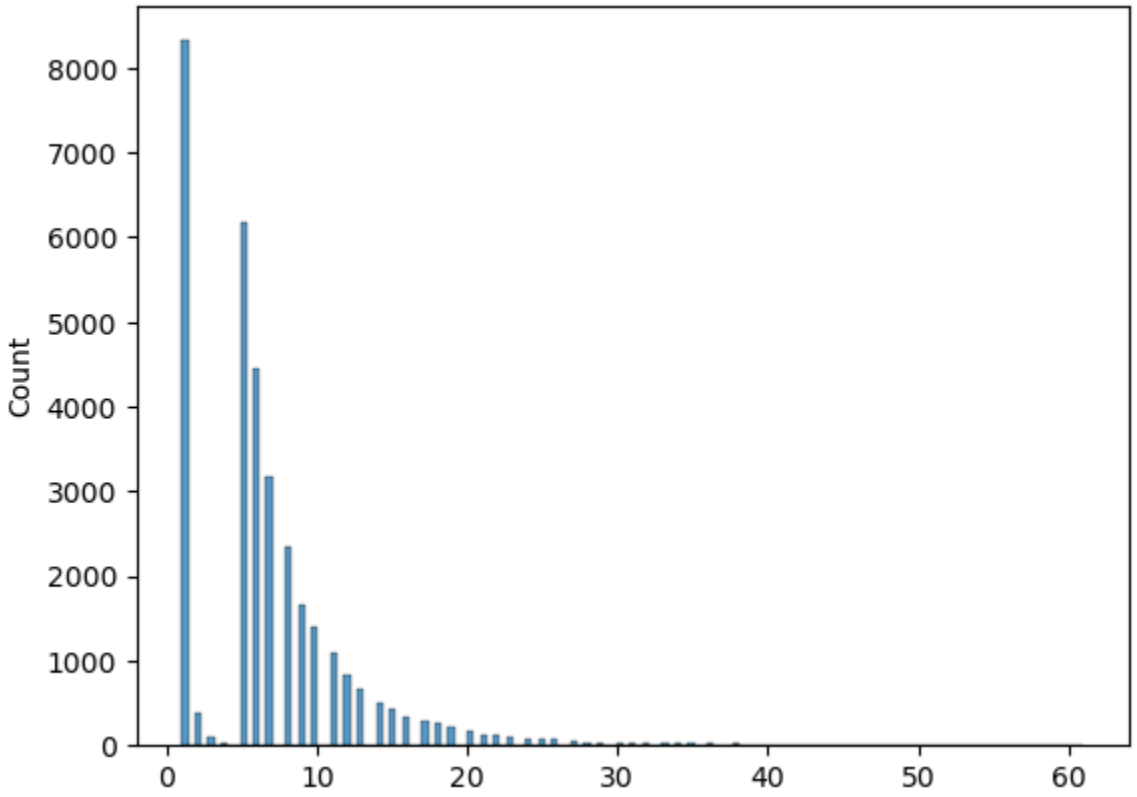


Course counts per genre



Each genre has a corresponding bar indicating its count. The tallest bar is for BackendDev, indicating it has the highest count, while the shortest bar is for Blockchain, indicating it has the lowest count. The graph provides a visual representation of the popularity or preference of these tech-related genres.

Course enrollment distribution



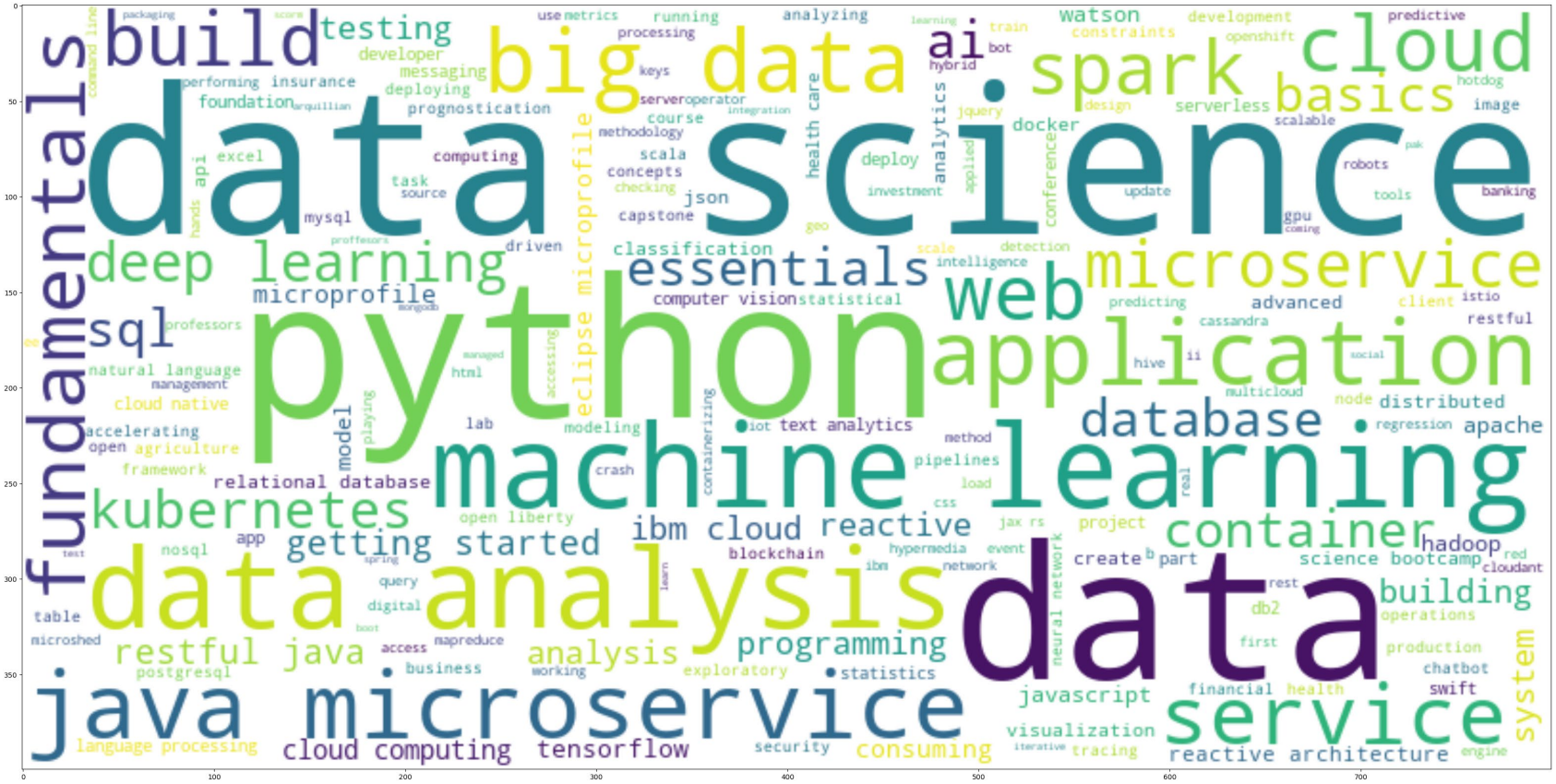
The histogram shows a significant concentration of user ratings distribution between approximately 0 and 15 on the x-axis, with the highest frequency reaching close to 8000 at around 0. As we move towards higher values on the x-axis, the frequency of data points decreases sharply. This distribution indicates that the majority of the ratings falls within the lower range.

20 most popular courses

	TITLE	Ratings
0	python for data science	14936
1	introduction to data science	14477
2	big data 101	13291
3	hadoop 101	10599
4	data analysis with python	8303
5	data science methodology	7719
6	machine learning with python	7644
7	spark fundamentals i	7551
8	data science hands on with open source tools	7199
9	blockchain essentials	6719
10	data visualization with python	6709
11	deep learning 101	6323
12	build your own chatbot	5512
13	r for data science	5237
14	statistics 101	5015
15	introduction to cloud	4983
16	docker essentials a developer introduction	4480
17	sql and relational databases 101	3697
18	mapreduce and yarn	3670
19	data privacy fundamentals	3624

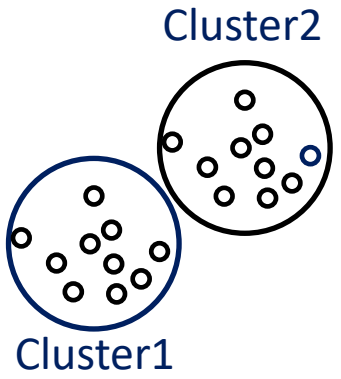
“Python for Data Science” has the highest rating of 14936, while “Data Privacy Fundamentals” has the lowest rating of 3624. This table appears to be an informational resource for individuals interested in data science, providing an overview of various courses/topics and their respective popularity or quality based on the given ratings.

Word cloud of course titles

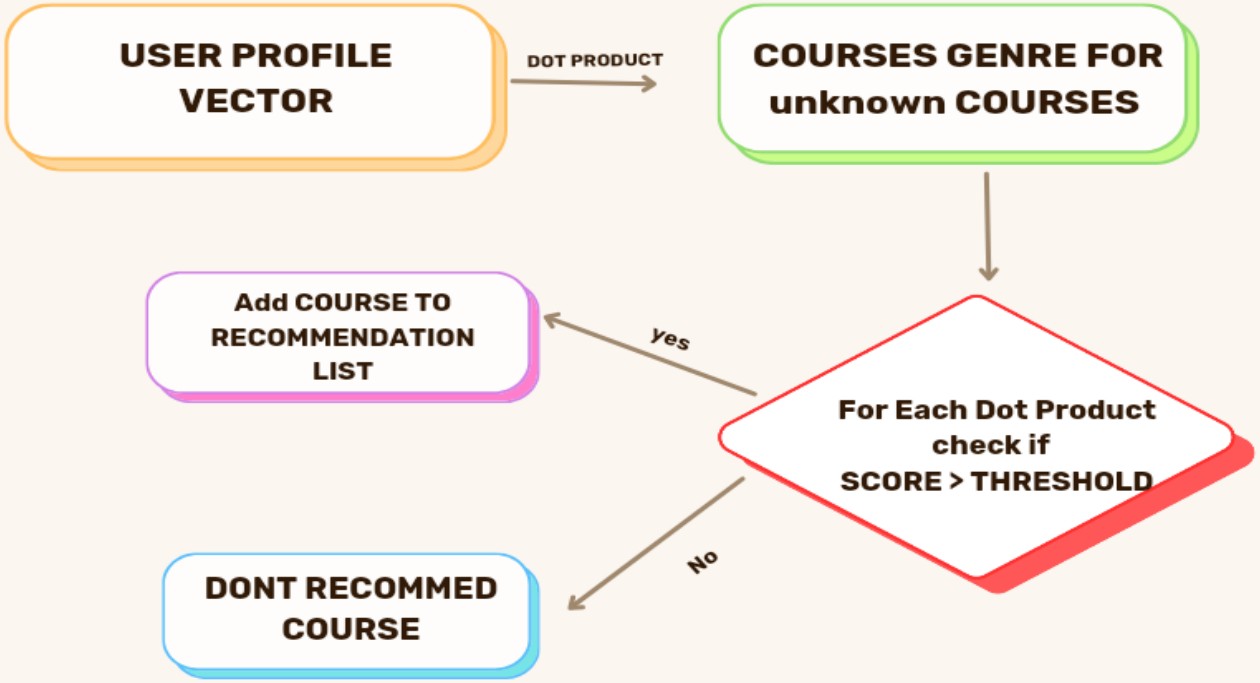


As we can see from the wordcloud, there are many popular IT related keywords such as python, data science, machine learning, big data, ai, tensorflow, container, cloud, etc. By looking at these keywords, we should have a general understanding that the courses in the dataset are focused on demanding IT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

Content-based Recommender System using Unsupervised Learning



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



It starts with a user's profile and unknown courses' genres. A score is calculated using the dot product of these two. If the score exceeds a threshold, the course is added to the recommendation list. If not, the course isn't recommended.

Evaluation results of user profile-based recommender system

score_threshold = 10.0

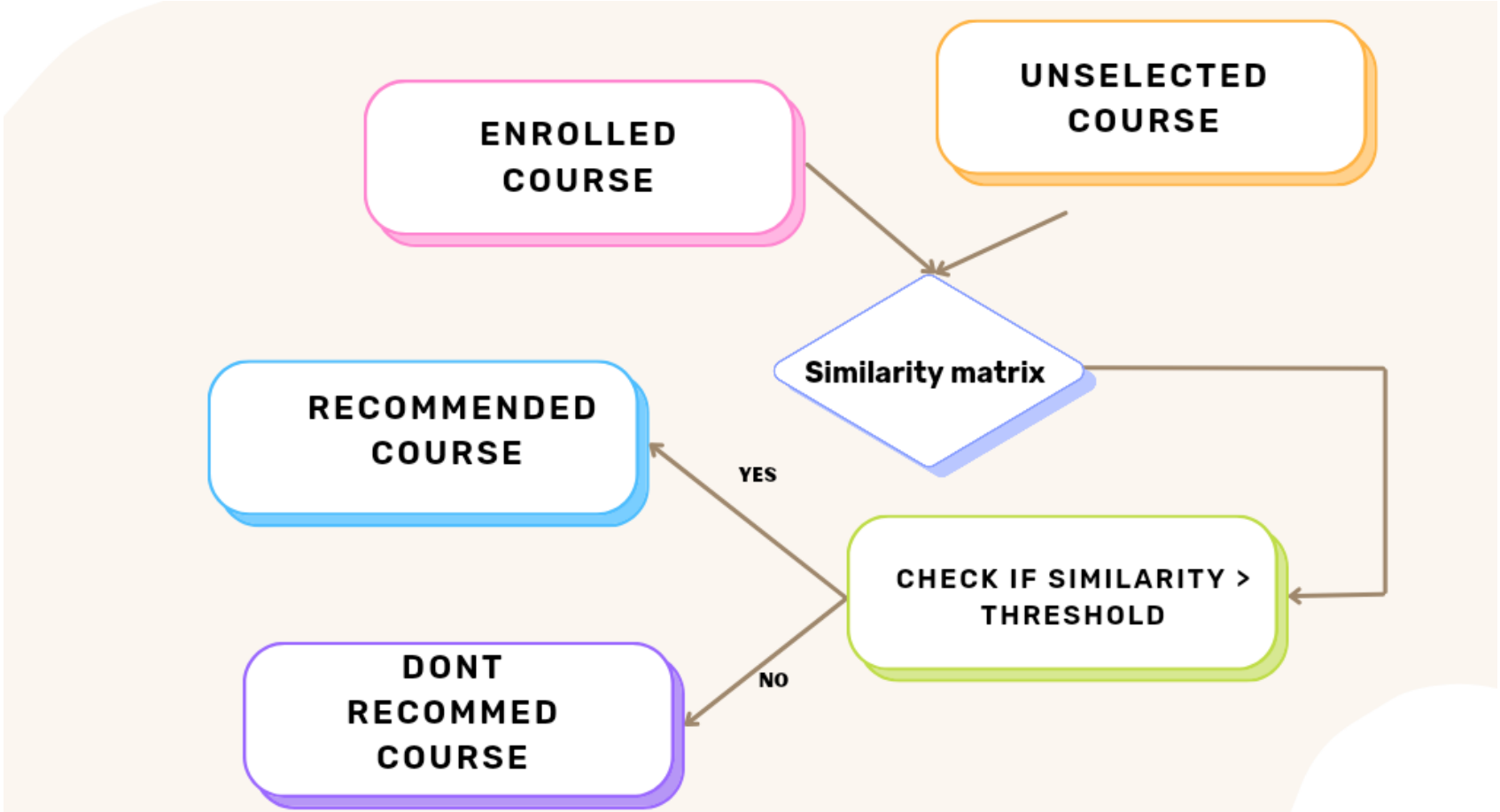
The output value of approximately 18.63 suggests that this is the average relevance score of the recommended courses for a particular user or set of users.
This average score can be useful in evaluating the performance of the recommender system

```
1  res_df['SCORE'].mean()
✓ 0.0s
18.62679972290352
```

The course with ID “TA0106EN” was recommended 608 times, “GPXX01BEN” was recommended 548 times, and so on. This data could be useful for understanding which courses are most frequently recommended by the system, indicating their relevance to a large number of users.

```
COURSE_ID
TA0106EN      608
GPXX0IBEN     548
excourse21    547
excourse22    547
ML0122EN      544
GPXX0TY1EN    533
excourse04    533
excourse06    533
excourse31    524
excourse72    516
dtype: int64
```

Flowchart of content-based recommender system using course similarity



The flowchart shows a recommender system. It compares an enrolled course with unselected ones. If they're similar enough, the unselected course is recommended

Evaluation results of course similarity based recommender system

threshold = 0.6

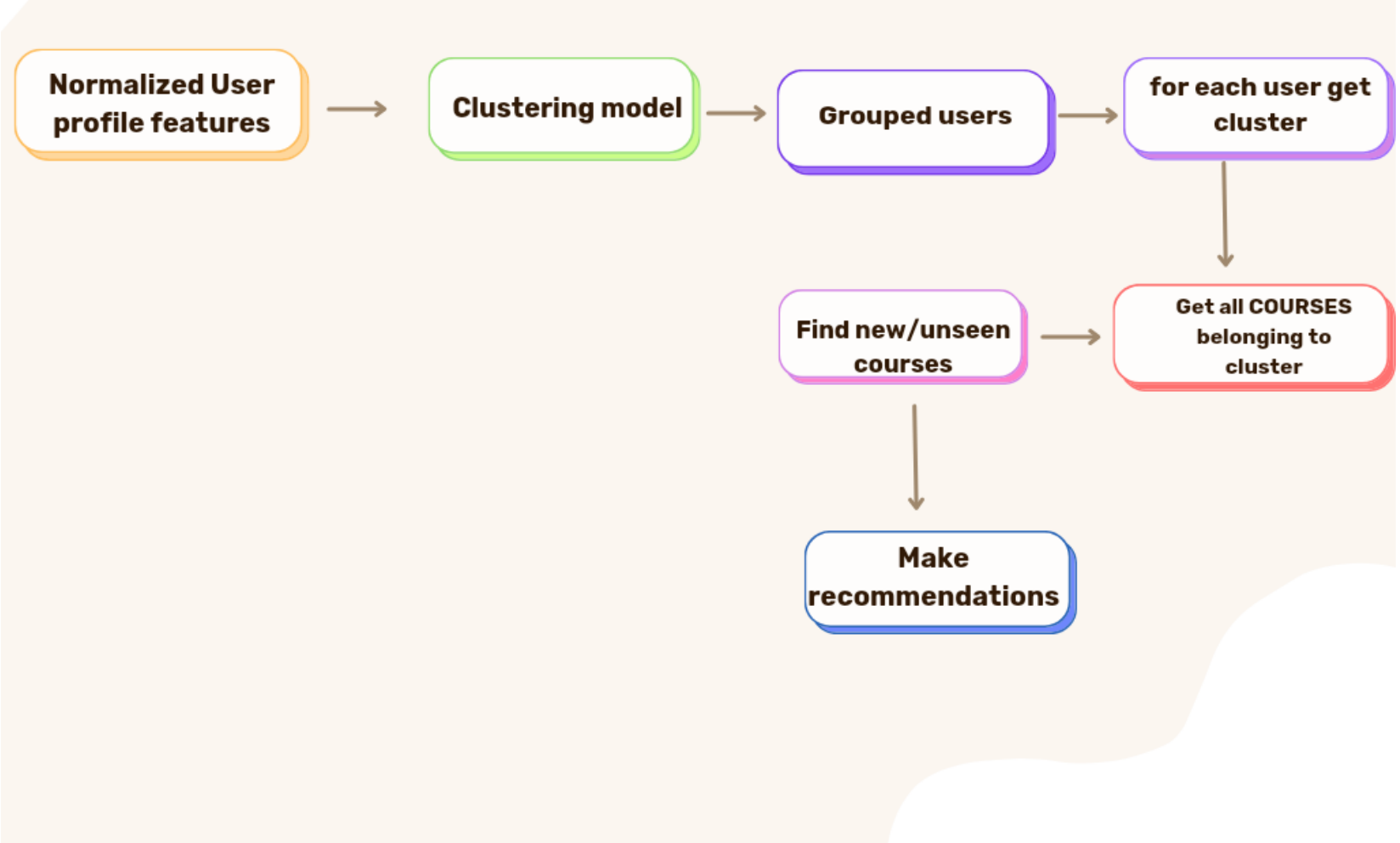
```
s = 0
for i in range(len(res_df['COURSE_ID'])):
    s += len(res_df['COURSE_ID'].iloc[i])
avg = s/len(res_df['COURSE_ID'])
```

	avg
0	11.377

The courses and their scores are listed in descending order, which suggests that the higher the score, the more similar the course is to a given course.

```
excourse62    579
excourse22    579
DS0110EN      562
excourse63    555
excourse65    555
excourse72    551
excourse68    550
excourse67    539
excourse74    539
BD0145EN      506
dtype: int64
```

Flowchart of clustering-based recommender system



Evaluation results of clustering-based recommender system

(n_clusters=20)

```
s = 0
for r in user_recommendations.values:
    s+=r[1:].sum()
    avg = s/len(user_recommendations)

```

```
avg
```

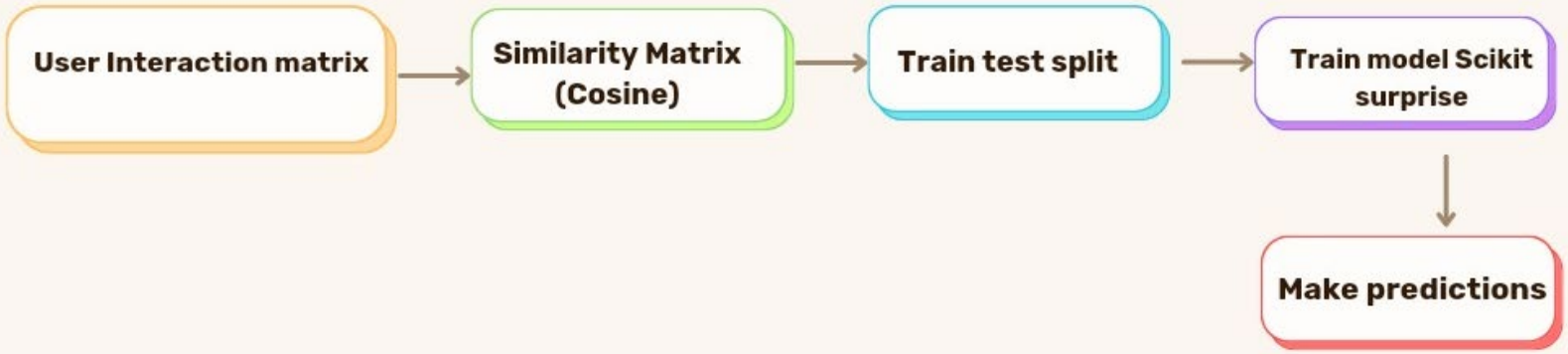
```
4.949
```

The higher the score, the more similar the course is to a given course.

```
BD0101EN    432
BD0111EN    412
PY0101EN    403
DS0101EN    368
DS0103EN    320
ST0101EN    266
ML0115EN    241
CC0101EN    206
BD0211EN    196
DS0105EN    193
dtype: int64
```

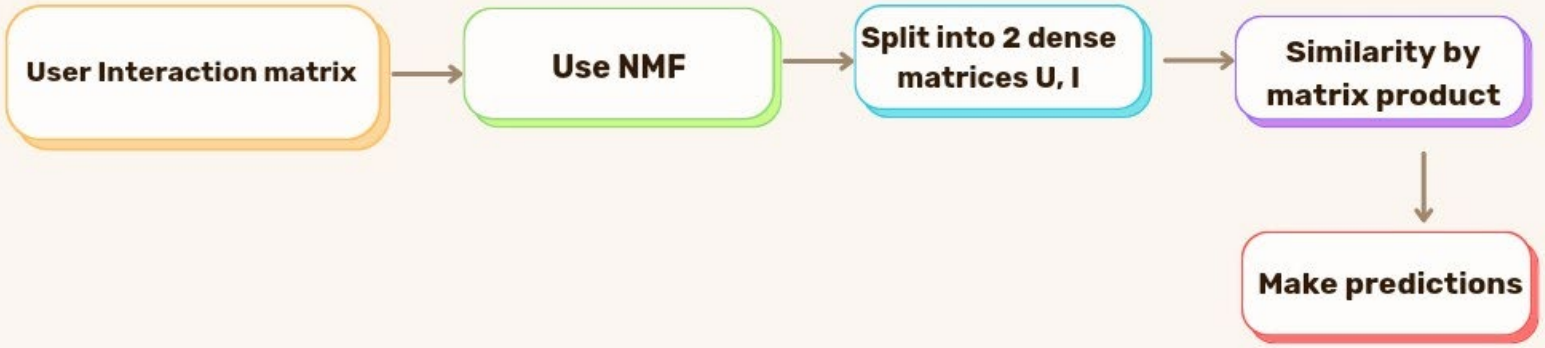
Collaborative-filtering Recommender System using Supervised Learning

Flowchart of KNN based recommender system



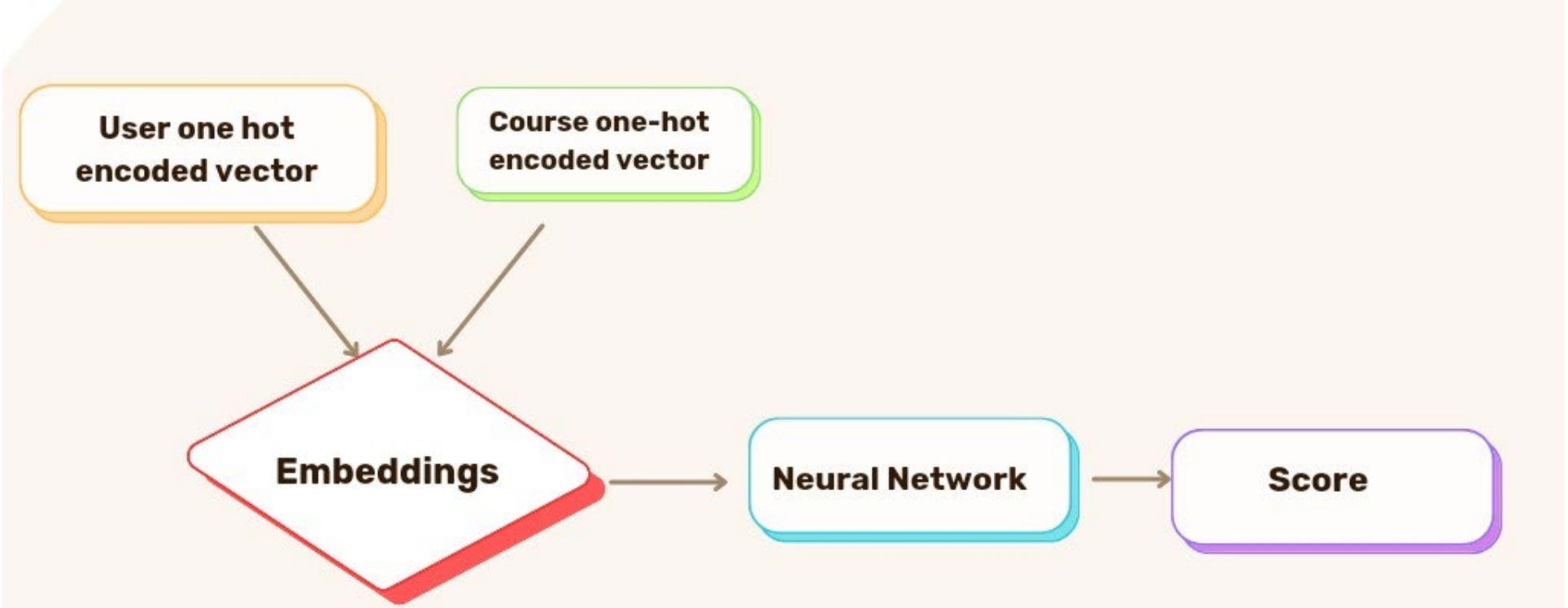
This flowchart provides overview of how a KNN based recommender system works. Each step in the flowchart represents a critical stage in the process of building and using the recommender system.

Flowchart of NMF based recommender system



This flowchart provides a visual representation of how an NMF-based recommender system works, from initial data collection to making final predictions. It's a useful tool for understanding the underlying processes involved in this type of recommender system.

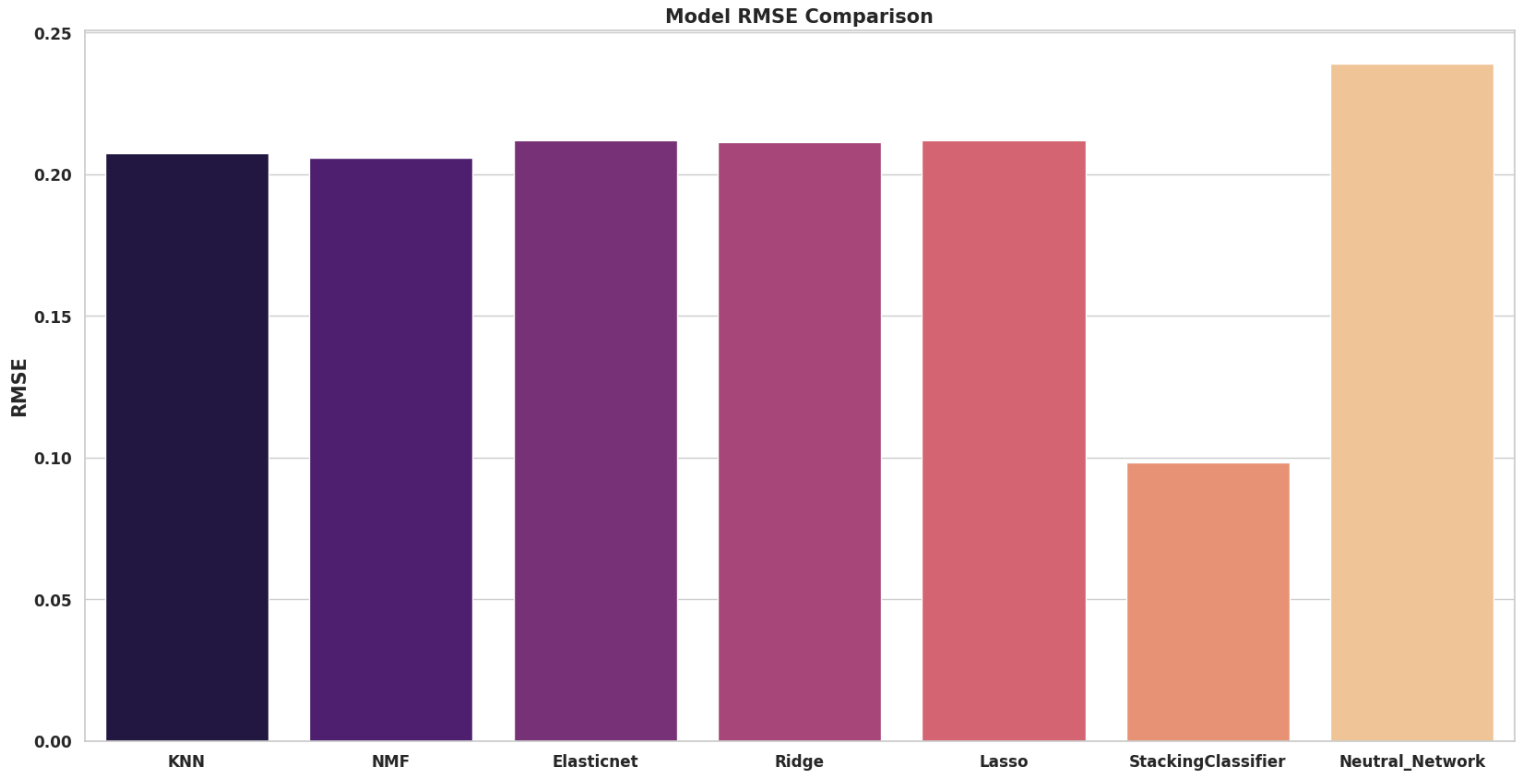
Flowchart of Neural Network Embedding based recommender system



The flowchart shows a Neural Network Embedding recommender system. It starts with one-hot encoded vectors for users and courses. These vectors are processed in an embedding step, then passed through a neural network. The final output is a score, indicating the match between a user and a course.

Compare the performance of collaborative-filtering models

This graph is useful for comparing the performance of different machine learning models. The lower the RMSE, the better the model's performance. In this case, the Neural Network model performs the best.



Conclusions

- The **Neural Network** model has the highest performance metric of **0.2389**, suggesting it may be the most effective model for this course recommender system. However, the **Stacking Classifier** model has a significantly lower performance metric of **0.0982**, indicating it might not be suitable for this task.
- The **KNN**, **NMF**, **Elasticnet**, **Ridge**, and **Lasso** models have similar performance metrics ranging from **0.2058** to **0.2120**, suggesting they could provide comparable results.

In conclusion, the Neural Network model appears to be the best choice for this course recommender system based on the provided performance metrics. However, further testing and validation would be necessary to confirm this. It's also important to consider other factors such as computational efficiency and the interpretability of the model.

Appendix

Different regression models such as Ridge, Lasso, ElasticNet and tune their hyperparameters

Lasso

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso,Ridge,ElasticNet

# Lasso
las = Lasso()
params = {'alpha':np.geomspace(0.1,1,20)}
grid_las = GridSearchCV(estimator= las, param_grid= params,cv= 3)
grid_las.fit(x_train,y_train)
print(grid_las.best_params_)
```

```
{'alpha': 0.1}
```

```
pred_las = grid_las.predict(x_test)
print('RMSE',np.sqrt(mean_squared_error(pred_las,y_test)))
```

```
RMSE 0.21200026329107982
```

Ridge

```
# ridge
r = Ridge()
params = {'alpha':np.geomspace(0.1,1,20)}
grid_r = GridSearchCV(estimator= r, param_grid= params,cv= 3)
grid_r.fit(x_train,y_train)
print(grid_r.best_params_)
```

```
{'alpha': 1.0}
```

```
pred_r = grid_r.predict(x_test)
print('RMSE',np.sqrt(mean_squared_error(pred_r,y_test)))
```

```
RMSE 0.21132863543056202
```

ElasticNet

```
# Elasticnet
elas= ElasticNet()
params = {'alpha':np.geomspace(0.1,1,5),'l1_ratio':np.geomspace(0.1,1,5)}
grid_elas= GridSearchCV(estimator= elas, param_grid= params,cv= 3)
grid_elas.fit(x_train,y_train)
print(grid_elas.best_params_)
```

```
{'alpha': 0.1, 'l1_ratio': 0.1}
```

```
pred_elas = grid_elas.predict(x_test)
print('RMSE',np.sqrt(mean_squared_error(pred_elas,y_test)))
```

```
RMSE 0.21200026329107982
```

Appendix

Classification models such as Logistic Regression, Tree models, SVM, Bagging, and Boosting models

Logistic Regression

```python
### You may need to tune the hyperparameters of the models
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
class_weight = {}

class_weight[0]= 0.96
class_weight[1]= 0.04
model_lr = LogisticRegression(penalty='l2',random_state=rs,max_iter=1000,class_weight=class_weight)
threshold = 0.5
model_lr.fit(X_train, y_train)
predicted_proba = model_lr.predict_proba(X_test)
yp_lr = (predicted_proba [:,1]>= threshold).astype('int')
print(classification_report(yp_lr, y_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.07      0.13     26484
           1       0.45      0.98      0.61     20178

    accuracy                           0.46     46662
   macro avg       0.64      0.53      0.37     46662
weighted avg       0.67      0.46      0.34     46662
```

Classification models

```python
### The main evaluation metrics could be accuracy, recall, precision, F score, and AUC.
from sklearn.ensemble import StackingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

estimators = [('SVM', SVC(random_state=rs)),('KNN',KNeighborsClassifier()),('dt',DecisionTreeClassifier())]
clf = StackingClassifier(estimators= estimators,final_estimator= LogisticRegression())
clf.fit(X_train, y_train)
preds_clf = clf.predict(X_test)
print(classification_report(preds_clf,y_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      1949
           1       1.00      0.99      0.99     44713

    accuracy                           0.99     46662
   macro avg       0.92      0.97      0.94     46662
weighted avg       0.99      0.99      0.99     46662
```