IBM ML Professional Certificate Capstone

Build a Personalized Online Course Recommender System with Machine Learning

Name: Shashank
Date: 31/03/2024

Outline:

Introduction

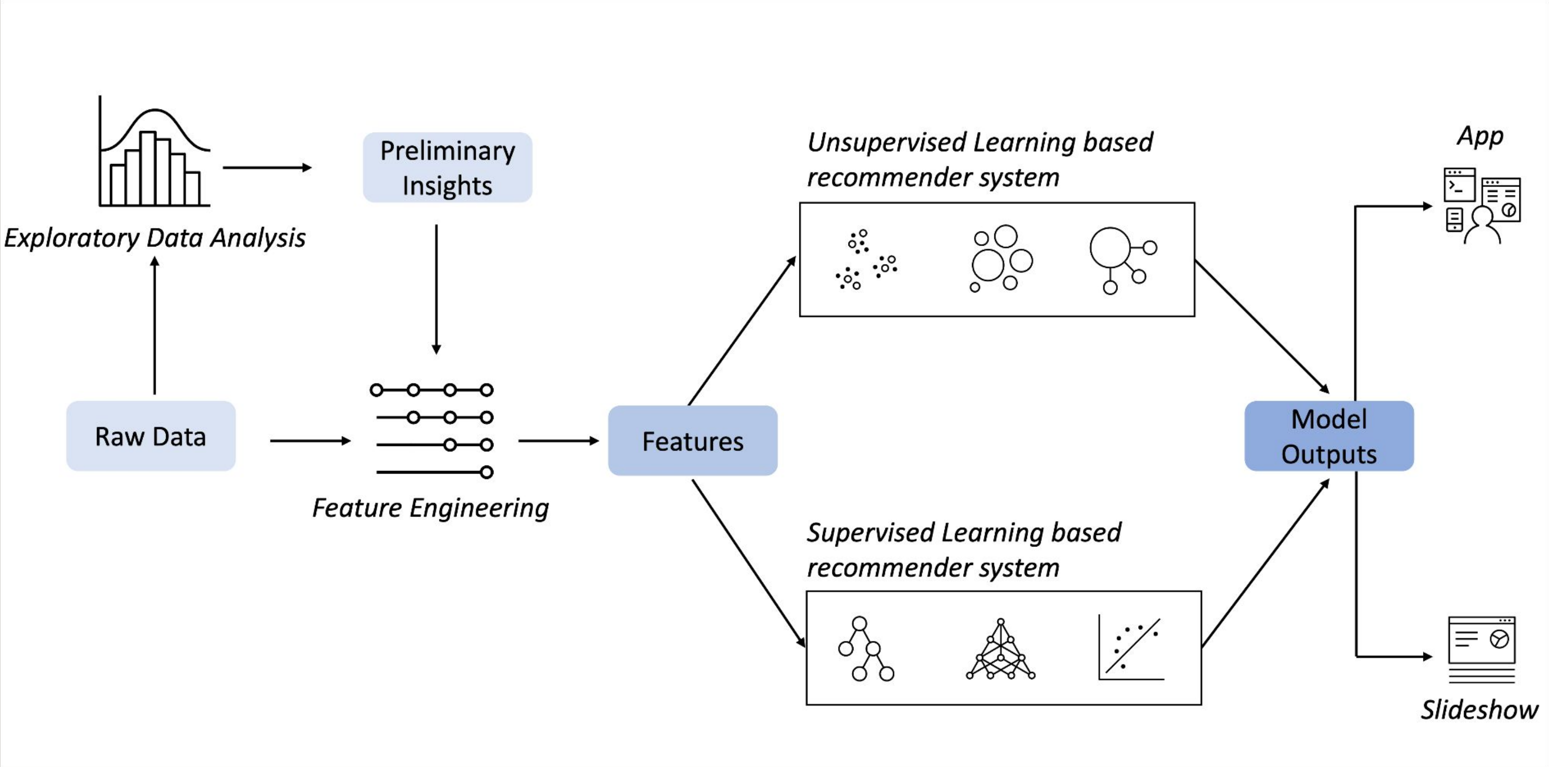
The primary objective of this project is to enhance the learning journey for students by facilitating the discovery of new courses aligned with their interests, thereby optimizing their learning paths.

Concurrently, by leveraging recommender systems to facilitate increased learner engagement with various courses, there's potential to boost the company's revenue.

Currently, the project is in the Proof of Concept (PoC) phase, prioritizing the exploration and comparison of different machine learning models to identify the one that exhibits optimal performance in offline evaluations.

Machine Learning Pipeline

1. Collecting and understanding data
2. Performing exploratory data analysis on online course enrollments datasets
3. Extracting Bag of Words (BoW) features from course textual content
4. Calculating course similarity using BoW features
5. Building content-based recommender systems using various unsupervised learning algorithms, such as: Distance/Similarity measurements, K-means, Principal Component Analysis (PCA), etc.
6. Building collaborative-filtering recommender systems using various supervised learning algorithms K Nearest Neighbors, Non-negative Matrix Factorization (NMF), Neural Networks, Linear Regression, Logistic Regression, RandomForest, etc.
7. Deploying and demonstrate modeling via a web app built with `streamlit`. `Streamlit` is an open-source app framework for Machine Learning and Data Science to quickly demonstration.
8. Reporting in paper.


Exploratory Data Analysis
Preliminary Insights
Raw Data
Feature Engineering
Features
Unsupervised Learning based recommender system
Supervised Learning based recommender system
Model Outputs
App
Slideshow

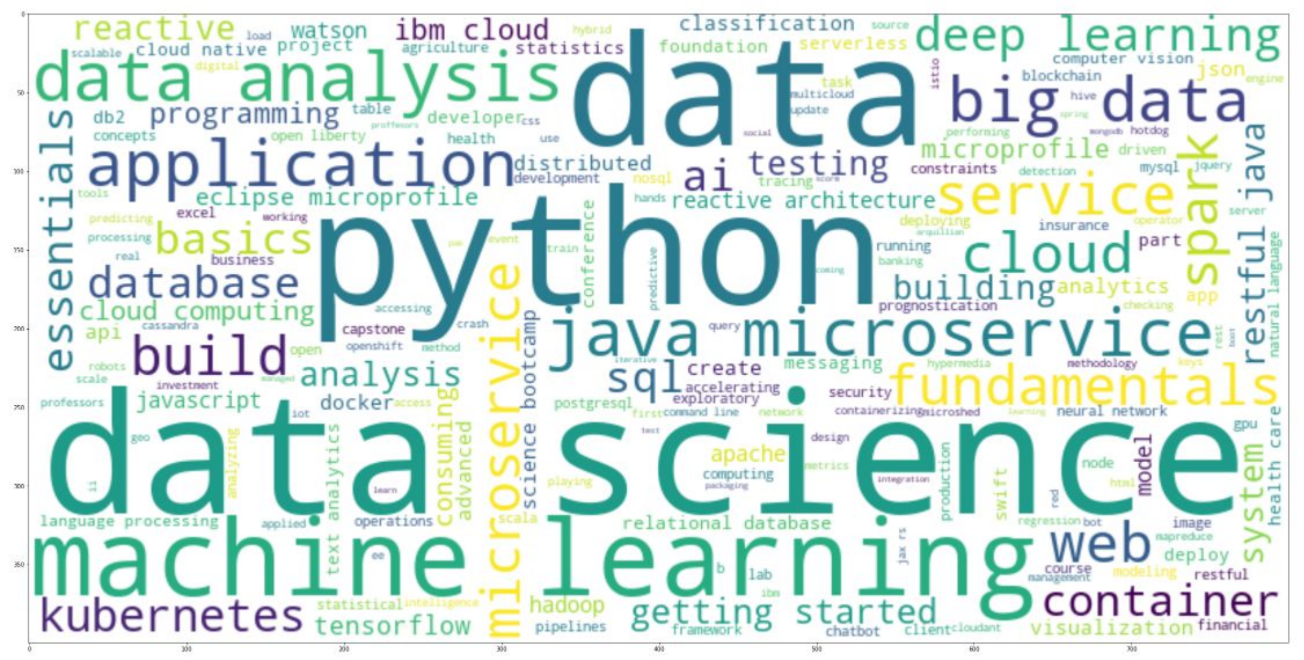
Exploratory Data Analysis

1. Describe the statistic of data columns
2. Identify keywords in course titles using a Word Cloud
3. Determine popular course genres
4. Calculate the summary statistics and create visualizations of the online course enrollment dataset

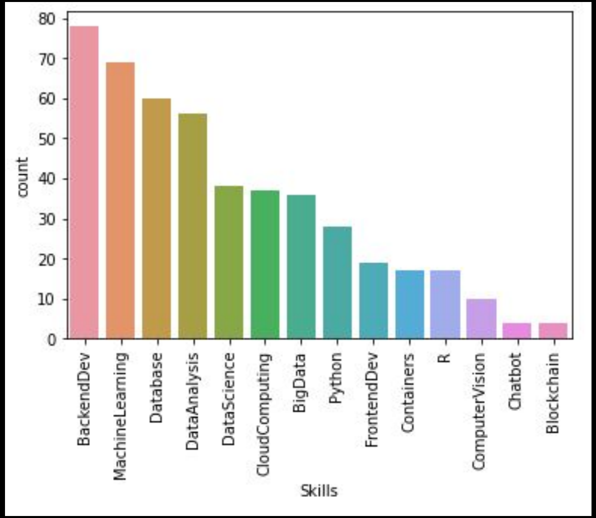
```
COURSE_ID           object
TITLE               object
Database             int64
Python               int64
CloudComputing       int64
DataAnalysis         int64
Containers           int64
MachineLearning      int64
ComputerVision       int64
DataScience          int64
BigData              int64
Chatbot              int64
R                    int64
BackendDev           int64
FrontendDev          int64
Blockchain           int64
dtype: object
```

- There are 307 courses at total.
- Each course is a vector of genres 1x16.
- **0** means this course is not related to this genre.
- **1** means this course is related to this genre.

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

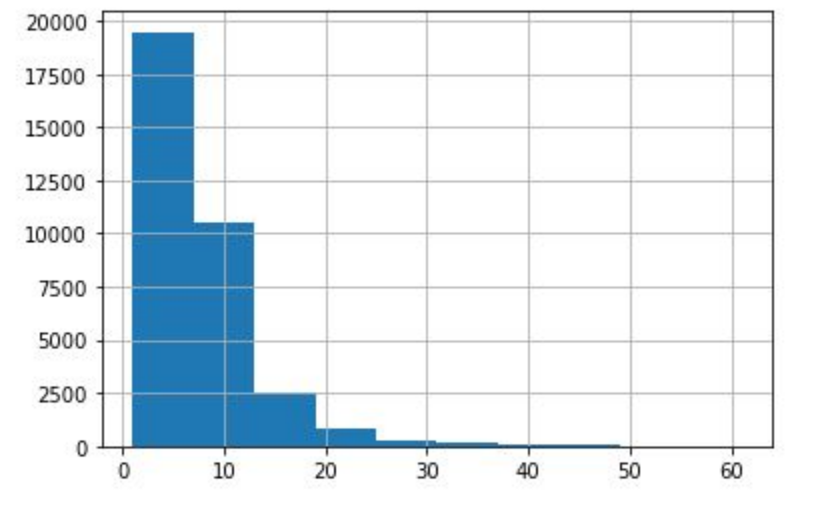

data
python
data science
machine learning
data analysis
application
big data
deep learning
java microservice
fundamentals
service
cloud
spark
container
kubernetes
build
database
basics
microservice
essentials
web
getting started
restful
java
testing
ai
reactive
ibm cloud
watson
classification
programming
sql
analysis
cloud computing
eclipse microprofile
reactive architecture
microprofile
building
analytics
system
tensorflow
hadoop
apache
javascript
docker
consuming
advanced
science bootcamp
text analytics
relational database
visualization
distributed
statistics
cloud native
project
foundation
serverless
computer vision
blockchain
json
developer
health care
natural language
language processing
messaging
security
create
model
chatbot
pipelines
framework
financial
deploy
neural network
postgresql
exploratory
accelerating
capstone
openshift
operations
api
cassandra
db2
concepts
open liberty
health
development
nosql
excel
business
processing
real
tools
scalable
agriculture
hybrid
source
multicloud
task
update
hive
hotdog
driven
mysql
jquery
constraints
detection
tracing
insurance
deploying
running
part
banking
prognostication
checking
app
hypermedia
methodology
keys
containerizing
microshed
gpu
design
node
image
mapreduce
course
modeling
restful
management
regression
bot
swift
production
computing
metrics
integration
packaging
playing
scala
applied
ee
statistical
intelligence
lab
client
cloudant
professors
investment
scale
robots
iot
geo
analyzing
access
crash
method
query
accessing
predicting
working
train
event
conference
predictive
hands
load
digital
table
css
use
istio
engine
operator
server
first
command line
network

	Skills	count
11	BackendDev	78
5	MachineLearning	69
0	Database	60
3	DataAnalysis	56
7	DataScience	38
2	CloudComputing	37
8	BigData	36
1	Python	28
12	FrontendDev	19
4	Containers	17
10	R	17
6	ComputerVision	10
9	Chatbot	4
13	Blockchain	4



- Backend Dev, Machine Learning, Database are the utmost popular genres.
- While Blockchain, Chatbot, Computer Vision are the most less common ones.

	COURSE_ID	count	TITLE
0	DS0301EN	3624	data privacy fundamentals
1	BD0115EN	3670	mapreduce and yarn
2	DB0101EN	3697	sql and relational databases 101
3	CO0101EN	4480	docker essentials a developer introduction
4	CC0101EN	4983	introduction to cloud
5	ST0101EN	5015	statistics 101
6	RP0101EN	5237	r for data science
7	CB0103EN	5512	build your own chatbot
8	ML0115EN	6323	deep learning 101
9	DV0101EN	6709	data visualization with python
10	BC0101EN	6719	blockchain essentials
11	DS0105EN	7199	data science hands on with open source tools
12	BD0211EN	7551	spark fundamentals i
13	ML0101ENv3	7644	machine learning with python
14	DS0103EN	7719	data science methodology
15	DA0101EN	8303	data analysis with python
16	BD0111EN	10599	hadoop 101
17	BD0101EN	13291	big data 101
18	DS0101EN	14477	introduction to data science
19	PY0101EN	14936	python for data science

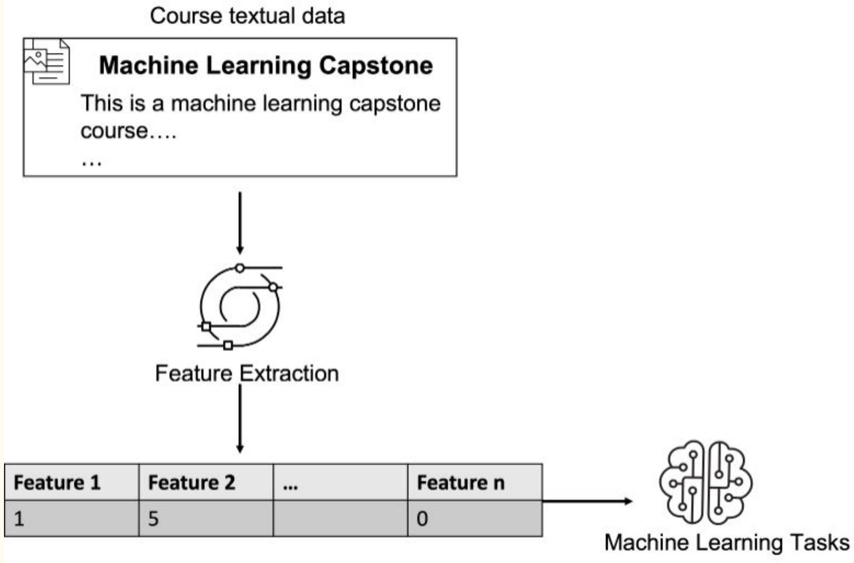


- The table shows top 20 widespread courses.
- 9 courses out of the top 10 are belong to data topic.
- The histogram illustrates the amount of user rating counts.
- Most of the users rarely rated any courses.
- A few exceptional students rated above 40 courses.

Feature Engineering

1. Extract Bag of Words (BoW) Features from Course Textual Content
   1.1. Bag of Words (BoW) features
   1.2. BoW dimensionality reduction
   1.3. Extract BoW features for course textual content and build a dataset
2. Calculate Course Similarity using BoW Features
   2.1. Calculate the cosine similarity between two example courses
   2.2. Find similar courses to the specific course

Feature Engineering



The main goal of recommender systems is to help users find items they potentially interested in. Depending on the recommendation tasks, an item can be a movie, a restaurant, or, in our case, an online course.

Machine learning algorithms cannot work on an item directly so we first need to extract features and represent the items mathematically, i.e., with a feature vector.

Many items are often described by text so they are associated with textual data, such as the titles and descriptions of a movie or course. Since machine learning algorithms can not process textual data directly, we need to transform the raw text into numeric feature vectors.

Bag of Words (BoW) features

BoW features are essentially the counts or frequencies of each word that appears in a text (string). Let's illustrate it with some simple examples.

Suppose we have two course descriptions as follows:

```
course1 = "this is an introduction data science course which introduces data science to beginners"
```

```
course2 = "machine learning for beginners"
```

```
courses = [course1, course2]
courses
```

```
['this is an introduction data science course which introduces data science to beginners',
 'machine learning for beginners']
```

The first step is to split the two strings into words (tokens). A token in the text processing context means the smallest unit of text such as a word, a symbol/punctuation, or a phrase, etc. The process to transform a string into a collection of tokens is called `tokenization`.

One common way to do `tokenization` is to use the Python built-in `split()` method of the `str` class. However, in this lab, we want to leverage the `nltk` (Natural Language Toolkit) package, which is probably the most commonly used package to process text or natural language.

More specifically, we will use the `word_tokenize()` method on the content of course (string):

```
# Tokenize the two courses
tokenized_courses = [word_tokenize(course) for course in courses]
```

```
Bag of words for course 0:
--Token: 'an', Count:1
--Token: 'beginners', Count:1
--Token: 'course', Count:1
--Token: 'data', Count:2
--Token: 'introduces', Count:1
--Token: 'introduction', Count:1
--Token: 'is', Count:1
--Token: 'science', Count:2
--Token: 'this', Count:1
--Token: 'to', Count:1
--Token: 'which', Count:1
Bag of words for course 1:
--Token: 'beginners', Count:1
--Token: 'for', Count:1
--Token: 'learning', Count:1
--Token: 'machine', Count:1
```

If we turn to the long list into a horizontal feature vectors, we can see the two courses become two numerical feature vectors:

	an	beginners	course	data	science	...
course1	1	1	1	2	2	
course2	0	1	0	0	0	

BoW dimensionality reduction

- A document often comprises a large number of words, resulting in a substantial dimensionality of the Bag of Words (BoW) feature vector.
- To mitigate this, a common approach involves reducing dimensionality by filtering out less meaningful tokens, such as stop words.
- Additionally, position and adjective words are sometimes incorporated for refinement.

Then we can filter those English stop words from the tokens in course1:

```
# Tokens in course 1
tokenized_courses[0]
```

```
['this',
 'is',
 'an',
 'introduction',
 'data',
 'science',
 'course',
 'which',
 'introduces',
 'data',
 'science',
 'to',
 'beginners']
```

```
processed_tokens = [w for w in tokenized_courses[0] if not w.lower() in stop_words]
```

```
processed_tokens
```

```
['introduction',
 'data',
 'science',
 'course',
 'introduces',
 'data',
 'science',
 'beginners']
```

You can see the number of tokens for `course1` has been reduced.

Another common way is to only keep nouns in the text. We can use the `nltk.pos_tag()` method to analyze the part of speech (POS) and annotate each word.

```
tags = nltk.pos_tag(tokenized_courses[0])
tags
```

```
[('this', 'DT'),
 ('is', 'VBZ'),
 ('an', 'DT'),
 ('introduction', 'NN'),
 ('data', 'NNS'),
 ('science', 'NN'),
 ('course', 'NN'),
 ('which', 'WDT'),
 ('introduces', 'VBZ'),
 ('data', 'NNS'),
 ('science', 'NN'),
 ('to', 'TO'),
 ('beginners', 'NNS')]
```

As we can see [`introduction`, `data`, `science`, `course`, `beginners`] are all of the nouns and we may keep them in the BoW feature vector.

Extract BoW features for course textual content and build a dataset

Then we need to create a token dictionary `tokens_dict`

TODO: Use gensim.corpora.Dictionary(tokenized_courses) to create a token dictionary.

```
# WRITE YOUR CODE HERE
tokens_dict = gensim.corpora.Dictionary(tokenized_courses)
print(tokens_dict.token2id)
```


```
{'ai': 0, 'apps': 1, 'build': 2, 'cloud': 3, 'coming': 4, 'create': 5, 'data': 6, 'developer': 7, 'found': 8, 'fun': 9, 'iot': 10, 'irobot': 11, 'learn': 12, 'node': 13, 'objects': 14, 'p
i': 15, 'pictures': 16, 'place': 17, 'program': 18, 'raspberry': 19, 'raspcam': 20, 'read': 21, 'recognize': 22, 'red': 23, 'robot': 24, 'robots': 25, 'services': 26, 'swift': 27, 'take':
28, 'temperature': 29, 'use': 30, 'want': 31, 'watson': 32, 'way': 33, 'accelerate': 34, 'accelerated': 35, 'accelerating': 36, 'analyze': 37, 'based': 38, 'benefit': 39, 'caffe': 40, 'ca
se': 41, 'chips': 42, 'classification': 43, 'comfortable': 44, 'complex': 45, 'computations': 46, 'convolutional': 47, 'course': 48, 'datasets': 49, 'deep': 50, 'dependencies': 51, 'deplo
y': 52, 'designed': 53, 'feel': 54, 'google': 55, 'gpu': 56, 'hardware': 57, 'house': 58, 'ibm': 59, 'images': 60, 'including': 61, 'inference': 62, 'large': 63, 'learning': 64, 'librarie
s': 65, 'machine': 66, 'models': 67, 'need': 68, 'needs': 69, 'network': 70, 'networks': 71, 'neural': 72, 'nvidia': 73, 'one': 74, 'overcome': 75, 'platform': 76, 'popular': 77, 'power':
78, 'preferring': 79, 'premise': 80, 'problem': 81, 'problems': 82, 'processing': 83, 'public': 84, 'reduce': 85, 'scalability': 86, 'scaling': 87, 'sensitiveand': 88, 'several': 89, 'sol
ution': 90, 'support': 91, 'supports': 92, 'system': 93, 'systems': 94, 'takes': 95, 'tensor': 96, 'tensorflow': 97, 'theano': 98, 'time': 99, 'torch': 100, 'tpu': 101, 'trained': 102, 't
raining': 103, 'understand': 104, 'unit': 105, 'uploading': 106, 'videos': 107, 'client': 108, 'consuming': 109, 'http': 110, 'invoke': 111, 'jax': 112, 'microservices': 113, 'reactive':
114, 'restful': 115, 'rs': 116, 'using': 117, 'analysis': 118, 'analyzing': 119, 'apache': 120, 'api': 121, 'big': 122, 'cluster': 123, 'computing': 124, 'distributed': 125, 'enables': 12
6, 'familiar': 127, 'frame': 128, 'framework': 129, 'performing': 130, 'provides': 131, 'r': 132, 'scale': 133, 'spark': 134, 'sparkr': 135, 'structured': 136, 'syntax': 137, 'used': 138,
'users': 139, 'application': 140, 'boot': 141, 'containerize': 142, 'containerizing': 143, 'liberty': 144, 'modification': 145, 'open': 146, 'package': 147, 'packaging': 148, 'run': 149,
'running': 150, 'server': 151, 'spring': 152, 'conference': 153, 'introduction': 154, 'native': 155, 'security': 156, 'bootcamp': 157, 'day': 158, 'intensive': 159, 'multi': 160, 'offere
d': 161, 'person': 162, 'proffesors': 163, 'science': 164, 'university': 165, 'containers': 166, 'development': 167, 'docker': 168, 'iterative': 169, 'scorm': 170, 'scron': 171, 'test': 1
72, 'basic': 173, 'collections': 174, 'creating': 175, 'database': 176, 'document': 177, 'first': 178, 'get': 179, 'guided': 180, 'management': 181, 'mongodb': 182, 'project': 183, 'start
ed': 184, 'working': 185, 'arquillian': 186, 'container': 187, 'develop': 188, 'managed': 189, 'testing': 190, 'tests': 191, 'aiops': 192, 'attending': 193, 'comprehensive': 194, 'demonst
```

	doc_index	doc_id	token	bow
0	0	ML0201EN	ai	2
1	0	ML0201EN	apps	2
2	0	ML0201EN	build	2
3	0	ML0201EN	cloud	1
4	0	ML0201EN	coming	1
...	...	...	...	...
10358	306	excourse93	modifying	1
10359	306	excourse93	objectives	1
10360	306	excourse93	pieces	1
10361	306	excourse93	plugins	1
10362	306	excourse93	populate	1

10363 rows × 4 columns

Create a new course_bow dataframe based on the extracted BoW features. he new dataframe needs to include the following columns:

- 'doc_index': the course index starting from 0
- 'doc_id': the actual course id such as ML0201EN
- 'token': the tokens for each course
- 'bow': the bow value for each token

Course Similarity using BoW Features

The measurement of similarity between items serves as the cornerstone for numerous recommendation algorithms, particularly those centered around content-based recommendations. For instance, if a new course bears resemblance to courses already enrolled by a user, it can be suggested to that user. Similarly, if User A shares similarities with User B, courses from User B's portfolio can be recommended to User A, assuming they have comparable interests.

Various similarity metrics such as cosine similarity, Jaccard index, or Euclidean distance can be employed for this purpose, operating on pairs of vectors, sets, or sometimes even matrices or tensors.

In the previous section, Bag of Words (BoW) features were extracted from the textual content of courses. With these BoW feature vectors at hand, it becomes straightforward to apply similarity measurement techniques to compute the similarity between courses, as illustrated in the subsequent figure.

Course 1: “Machine Learning for Everyone”

	machine	learning	for	everyone	beginners
course1	1	1	1	1	0

Course 2: “Machine Learning for Beginners”

	machine	learning	for	everyone	beginners
course2	1	1	1	0	1

Similarity Calculation:
Cosine, Euclidean, Jaccard index, …

75%

Unsupervised Learning based Recommendation System

1. Content-based Course Recommender System using User Profile and Course Genres
2. Content-based Course Recommender System using Course Similarities
3. Clustering based Course Recommender System

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

The predominant form of content-based recommendation systems involves suggesting items to users according to their profiles.

The user's profile is centered around their preferences and tastes, molded by factors such as user ratings, including clicks on various items or expressing liking for them.

Recommendation is driven by the similarity between items, gauged through the likeness in their content. Content here encompasses attributes like category, tags, genre, and other features pertaining to an item. In essence, it refers to the characteristics that describe an item.

In online course recommender systems, once features are extracted from courses (such as genres or Bag of Words features), the next step involves constructing user profiles based on course genres and users' ratings, especially for users whose profiles are unknown.

A user profile can be viewed as a mathematical representation of a user's learning interests, depicted as a user feature vector.

With both user profile feature vectors and course genre feature vectors in place, various computational methods can be employed. For instance, a straightforward approach like the dot product can be utilized to compute or predict an interest score for each course. Subsequently, courses with high interest scores can be recommended to users.

Raw data

Course genres dataframe: course_id, title, [genre_x, genre_y,...]

User dataframe: user_id, [genre_interest_x, genre_interest_y,...]

Feature engineering

Analysis data

Drop the exceptions

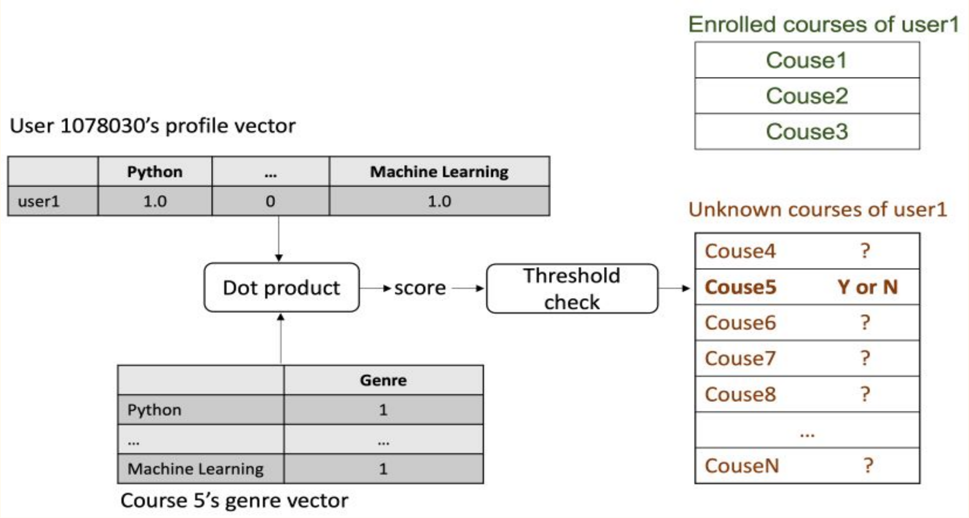
Normalise data

Get recommend score

Use dot product between single user vector and specific course to get recommend score

Recommendation

Make prediction by comparing score with determine threshold



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

$\otimes$

	user	Database	Python	CloudComputing	DataAnalysis	Containers	MachineLearning
0	2	52.0	14.0	6.0	43.0	3.0	33.0
1	4	40.0	2.0	4.0	28.0	0.0	14.0
2	5	24.0	8.0	18.0	24.0	0.0	30.0
3	7	2.0	0.0	0.0	2.0	0.0	0.0
4	8	6.0	0.0	0.0	4.0	0.0	0.0

=

	USER	COURSE_ID	SCORE
0	37465	RP0105EN	27.0
1	37465	GPXX06RFEN	12.0
2	37465	CC0271EN	15.0
3	37465	BD0145EN	24.0
4	37465	DE0205EN	15.0
...	...	...	...
53406	2087663	excourse88	15.0
53407	2087663	excourse89	15.0
53408	2087663	excourse90	15.0
53409	2087663	excourse92	15.0
53410	2087663	excourse93	15.0

Content-based Course Recommender System using Course Similarities

As previously mentioned, content-based recommender systems heavily rely on calculating the similarity between items. This similarity is assessed based on the likeness in the content or features of those items.

Course genres serve as crucial features, and alongside them, the Bag of Words (BoW) values provide another essential type of feature to represent the textual content of courses.

In this lab, we will utilize course similarity metrics to recommend new courses that are akin to a user's currently enrolled courses. By leveraging the similarities between courses, we aim to suggest additional courses that align closely with the ones the user is already engaged with.

Raw data	Features	Score	Prediction

1. Course similarity matrix

2. Course dataframe:
course_id, title, description

Visualise the similarly metric

Handle data text in title and description

- Remove stop words
- word2vec

Turn a course description to a vector

Check similarity by compare 2 vector

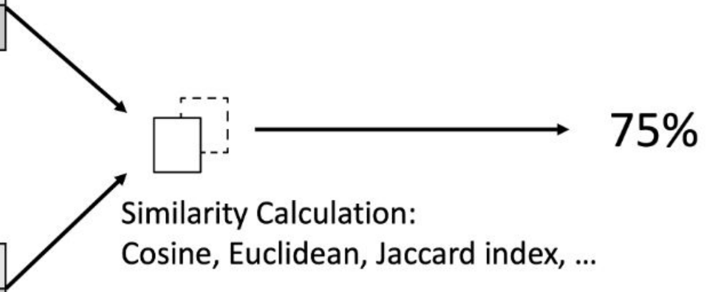
Determine rely on similar score

Course 1: “Machine Learning for Everyone”

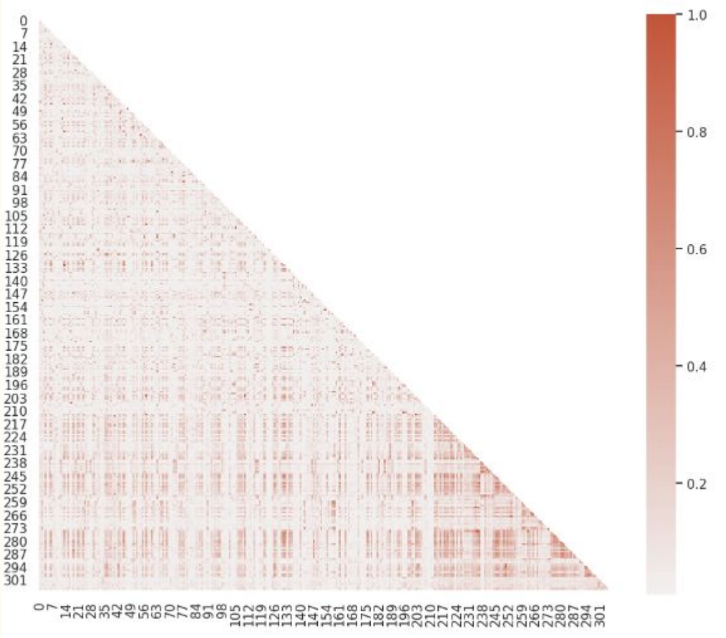
	machine	learning	for	everyone	beginners
course1	1	1	1	1	0

Course 2: “Machine Learning for Beginners”

	machine	learning	for	everyone	beginners
course2	1	1	1	0	1



	0	1	2	3	4
0	1.000000	0.088889	0.088475	0.065556	0.048810
1	0.088889	1.000000	0.055202	0.057264	0.012182
2	0.088475	0.055202	1.000000	0.026463	0.039406
3	0.065556	0.057264	0.026463	1.000000	0.000000
4	0.048810	0.012182	0.039406	0.000000	1.000000
...	...	...	...	...	...



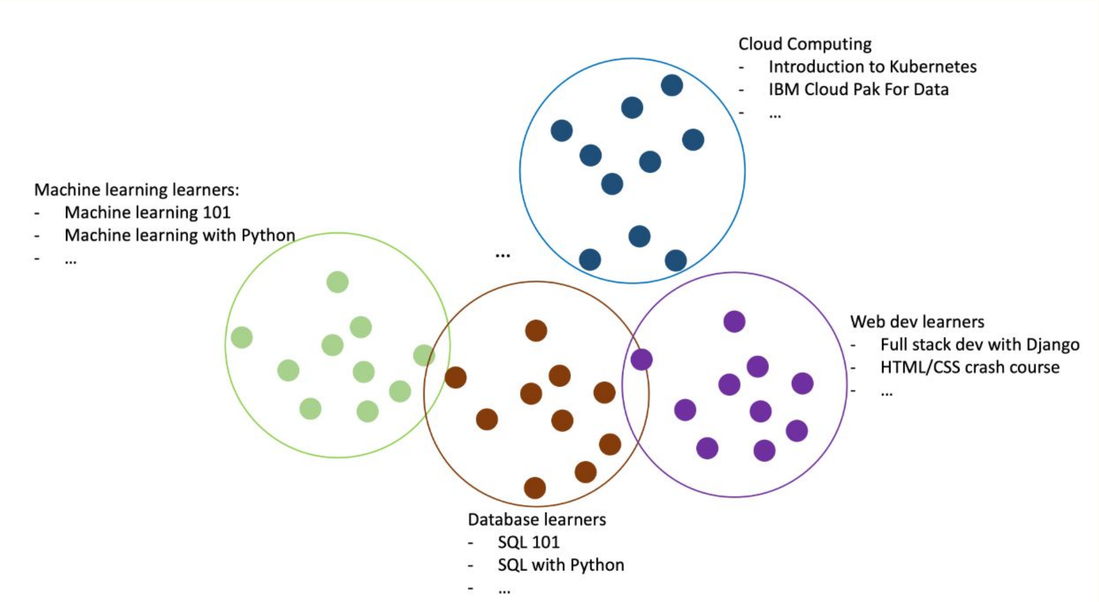
	USER	COURSE_ID	SCORE
0	37465	excourse67	0.708214
1	37465	excourse72	0.652535
2	37465	excourse74	0.650071
3	37465	BD0145EN	0.623544
4	37465	excourse68	0.616759

Clustering based Course Recommender System

Clustering algorithms like K-means or DBSCAN can be employed to group users with similar learning interests.

For instance, below are examples of user clusters where each cluster comprises users who have taken courses related to machine learning, cloud computing, databases, web development, and so on.

These clusters enable us to identify cohorts of users with comparable learning preferences, facilitating more targeted recommendations and personalized learning experiences.

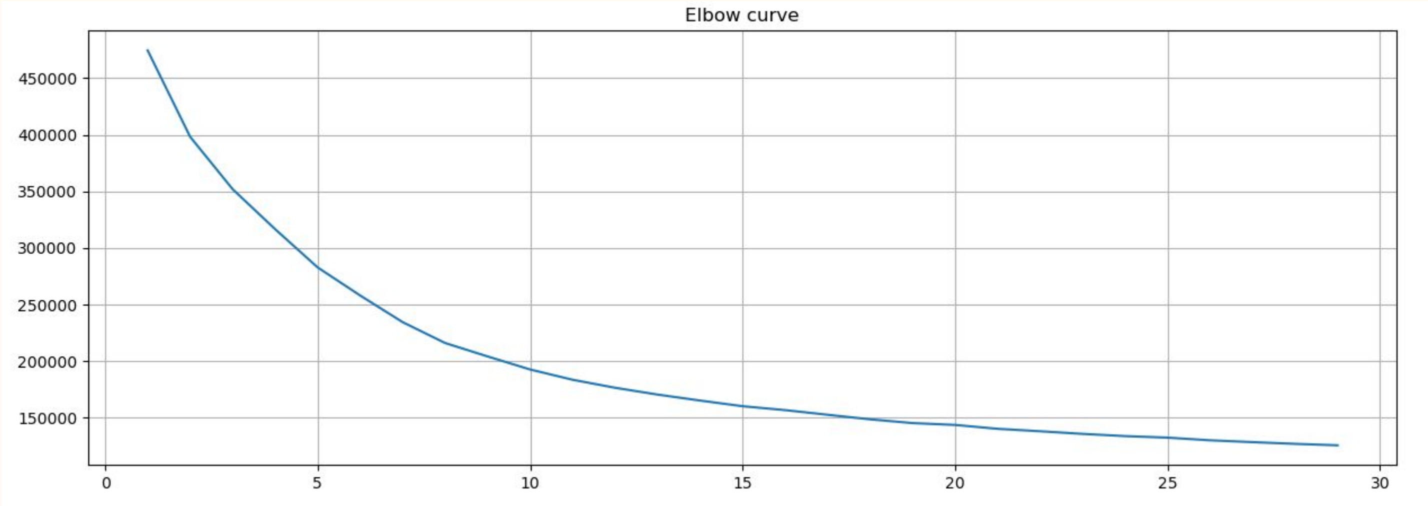


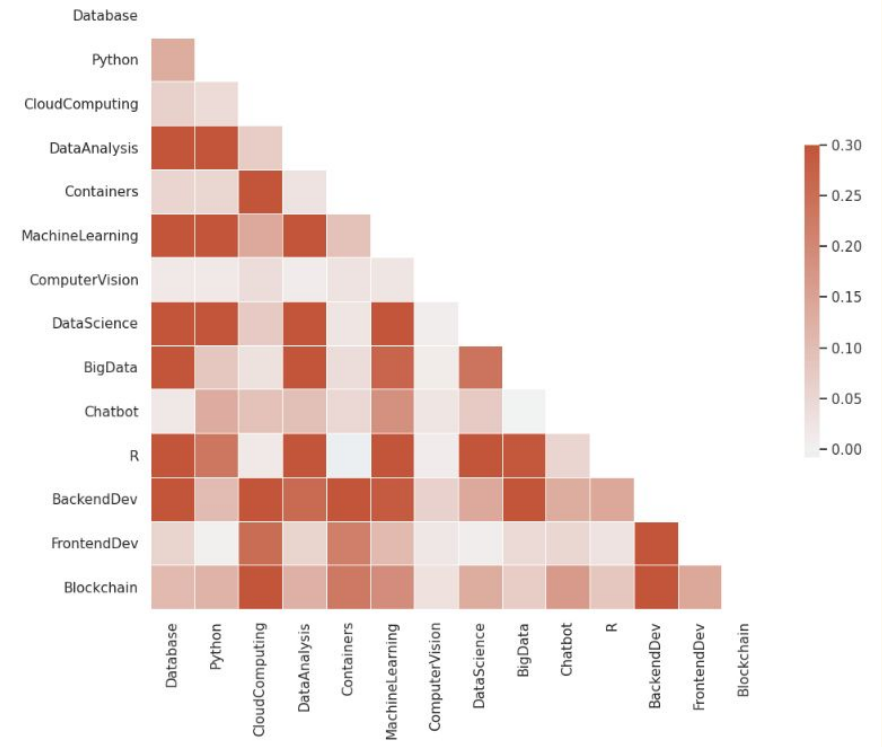
Raw data → Features → Clusters → Prediction

1. Raw data:
   - User profile dataframe: user_id, [genre_x, genre_y,...]
2. Features:
   - Normalise user profile features
   - Apply PCA to keep only important features
3. Apply Clustering algorithms to group similar courses
4. Make recommendation by taking courses in user's interest group

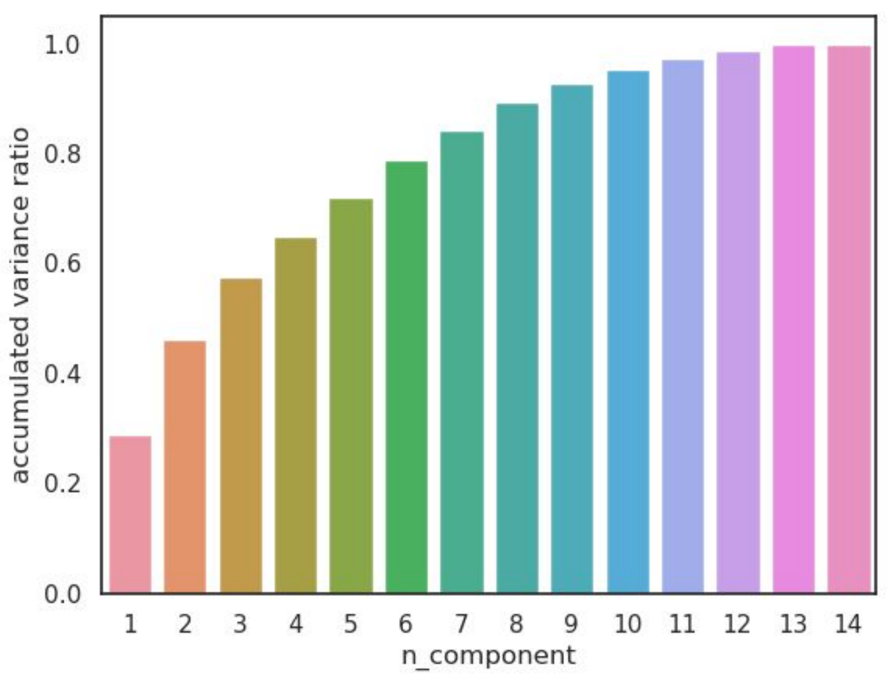
In the K-Means algorithm, a crucial hyperparameter is the number of clusters, denoted as n_clusters. To determine the optimal value for n_clusters, one effective approach is to perform a grid search across a range of candidate values and select the value that yields the best clustering evaluation metrics, such as the minimal sum of squared distances.

This grid search involves iterating through a list of possible values for n_clusters and evaluating the resulting clusters using a chosen metric, such as the within-cluster sum of squared distances. The value of n_clusters that minimizes this metric typically indicates the most suitable number of clusters for the given dataset.





Plot a covariance matrix of the user profile feature vectors with 14 features, we can observe that some features are actually correlated



- Utilizing the PCA (Principal Component Analysis) function from scikit-learn, we aim to identify the primary components within user profile feature vectors and explore the possibility of reducing its dimensionality by retaining only these key components.
- By examining the cumulative variance ratio, such as 90%, for a given number of components (n_components), we can determine if the transformed components adequately explain the variance in the original data.
- If the accumulated variance ratio exceeds the specified threshold, say 90%, for a certain n_components value, such as 8, then it suggests that these components capture around 90% of the original data's variance and can be considered as an optimized size for the components.
- Therefore, we select n_component = 8 as it meets the criterion of having a cumulative variance ratio greater than 0.9.

Apply PCA to features:

	user	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
0	2	17.772494	0.200681	1.730609	2.567359	-3.825814	2.707154	0.681042	2.312613	0.868272
1	4	7.145199	-2.847481	2.358636	-0.576654	0.398803	-0.134533	0.549769	0.469033	0.033405
2	5	11.363270	1.873619	-1.522077	1.076144	-1.711688	0.883212	1.677582	2.937669	2.097639
3	7	-1.834033	-0.277462	0.564905	0.053470	-0.064440	0.165757	0.030956	0.039519	0.210887
4	8	-1.049125	-0.684767	1.072765	0.006371	-0.005695	0.118686	0.118559	0.559292	0.186379
...	...	...	...	...	...	...	...	...	...	...
33896	2102054	0.633824	0.108815	-0.388871	-0.122665	-0.098364	0.358333	1.752049	1.486542	-0.523600
33897	2102356	-2.095339	0.135058	0.244727	-0.088185	0.025081	0.183641	0.046413	0.191709	0.260437
33898	2102680	0.625943	-0.547167	-1.692824	-0.630589	0.166632	0.676244	-0.055100	0.582091	1.703193
33899	2102983	-2.036832	-0.153534	0.162852	0.082651	-0.126419	0.255109	0.072496	0.113750	0.622900
33900	2103039	-2.036832	-0.153534	0.162852	0.082651	-0.126419	0.255109	0.072496	0.113750	0.622900

33901 rows × 10 columns

Apply K-Means on transformed features:

	user	cluster
0	2	9
1	4	23
2	5	9
3	7	15
4	8	8
...	...	...
33896	2102054	21
33897	2102356	15
33898	2102680	17
33899	2102983	15
33900	2103039	15

33901 rows × 2 columns

```
user in cluster 0 will be sugessted 3 courses as ['BC0101EN' 'BD0101EN' 'DS0101EN']
user in cluster 1 will be sugessted 3 courses as ['CO0101EN' 'CC0101EN' 'CO0201EN']
user in cluster 2 will be sugessted 3 courses as ['PY0101EN' 'CB0103EN' 'DA0101EN']
user in cluster 3 will be sugessted 3 courses as ['CB0103EN' 'BC0101EN' 'PY0101EN']
user in cluster 4 will be sugessted 3 courses as []
user in cluster 5 will be sugessted 3 courses as ['PY0101EN' 'DS0101EN' 'DA0101EN']
user in cluster 6 will be sugessted 3 courses as ['CC0101EN' 'PY0101EN' 'CC0103EN']
user in cluster 7 will be sugessted 3 courses as ['BC0101EN' 'BC0201EN' 'PY0101EN']
user in cluster 8 will be sugessted 3 courses as ['BD0101EN' 'BD0111EN' 'DS0101EN']
user in cluster 9 will be sugessted 3 courses as ['BD0101EN' 'BD0111EN' 'SW0101EN']
user in cluster 10 will be sugessted 3 courses as ['DS0101EN' 'RP0101EN' 'PY0101EN']
user in cluster 11 will be sugessted 3 courses as ['CO0101EN' 'LB0101ENv1' 'CO0401EN']
user in cluster 12 will be sugessted 3 courses as ['BD0111EN' 'BD0115EN' 'BD0141EN']
user in cluster 13 will be sugessted 3 courses as ['CO0101EN' 'CO0201EN' 'CO0301EN']
user in cluster 14 will be sugessted 3 courses as ['BC0101EN' 'PY0101EN' 'DA0101EN']
user in cluster 15 will be sugessted 3 courses as ['DS0101EN' 'BD0101EN' 'PY0101EN']
user in cluster 16 will be sugessted 3 courses as ['CB0103EN' 'PY0101EN' 'DA0101EN']
user in cluster 17 will be sugessted 3 courses as ['PY0101EN' 'ML0101ENv3' 'ML0115EN']
user in cluster 18 will be sugessted 3 courses as ['BD0111EN' 'BD0211EN' 'BD0101EN']
user in cluster 19 will be sugessted 3 courses as ['BD0211EN' 'BD0101EN' 'DS0101EN']
user in cluster 20 will be sugessted 3 courses as ['BD0111EN' 'BD0101EN' 'BD0211EN']
user in cluster 21 will be sugessted 3 courses as ['RP0101EN' 'DS0101EN' 'DS0103EN']
user in cluster 22 will be sugessted 3 courses as ['LB0101ENv1' 'LB0103ENv1' 'LB0105ENv1']
user in cluster 23 will be sugessted 3 courses as ['BD0111EN' 'PY0101EN' 'BD0211EN']
user in cluster 24 will be sugessted 3 courses as ['CB0103EN' 'DS0101EN' 'BD0101EN']
```

- Find popular courses in clusters and suggest to user in cluster
- Insights: On average, how many new/unseen courses have been recommended per user (in the test user dataset)
- What are the most frequently recommended courses?
- Return the top-10 commonly recommended courses

Supervised Learning based Recommendation System

1. CF using K Nearest Neighbor
2. CF using Non-negative Matrix Factorization
3. Course Rating Prediction using Neural Networks
4. Regression-Based Rating Score Prediction Using Embedding Features
5. Classification-based Rating Mode Prediction using Embedding Features

CF using K Nearest Neighbor

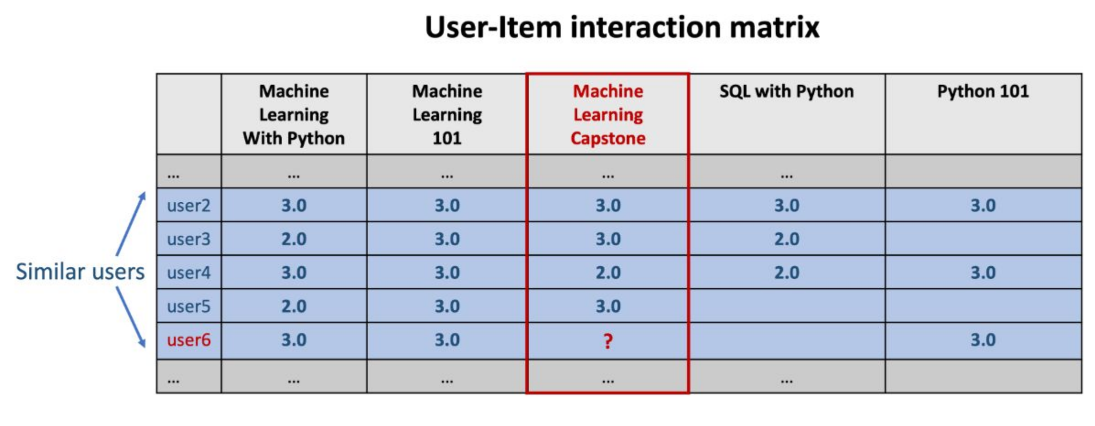
Collaborative filtering is probably the most commonly used recommendation algorithm, there are two main types of methods:

- **User-based** collaborative filtering is based on the user similarity or neighborhood
- **Item-based** collaborative filtering is based on similarity among items

User-based collaborative filtering identifies users with similar preferences by analyzing their interactions with items in a 2-D matrix format called the user-item interaction matrix.

Traditional methods use explicit user profiles for similarity calculation, but when these are unavailable, techniques like cosine similarity or Pearson correlation coefficient are applied directly to the interaction matrix to measure similarity.

This approach enables collaborative filtering systems to recommend items to users based on the patterns of their interactions with items.



	Machine Learning With Python	Machine Learning 101	Machine Learning Capstone	SQL with Python	Python 101
...	...	...	...	...	
user2	3.0	3.0	3.0	3.0	3.0
user3	2.0	3.0	3.0	2.0	
user4	3.0	3.0	2.0	2.0	3.0
user5	2.0	3.0	3.0		
user6	3.0	3.0	?		3.0
...	...	...	...	...	

Predict the rating of user *user6* to item *Machine Learning Capstone*

We used the library Surprise library to handle dataset and fit the data.

Distance metric: Only common users (or items) are taken into account. The cosine similarity is defined as:

For users u, v:

$$\text{cosine\_sim}(u, v) = \frac{\sum\limits_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum\limits_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} r_{vi}^2}}$$

For items i, j:

$$\text{cosine\_sim}(i, j) = \frac{\sum\limits_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum\limits_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} r_{uj}^2}}$$

Raw data → **Spare data** → **Model** → **Prediction**

Raw data

User - item - rating - dataframe: user_id, item, rating

Spare data

Use *pivot* method in pandas to turn data to features

Model

Fit data to KNN model based on *surprise* library

Use distance metric listed in previous page

Prediction

Make prediction by test data, use RMSE metric to evaluate model performance

	user	AI0111EN	BC0101EN	BC0201EN	BC0202EN	BD0101EN
0	2	0.0	3.0	0.0	0.0	3.0
1	4	0.0	0.0	0.0	0.0	2.0
2	5	2.0	2.0	2.0	0.0	2.0
3	7	0.0	0.0	0.0	0.0	0.0
4	8	0.0	0.0	0.0	0.0	0.0

5 rows × 127 columns

```
# Then compute RMSE
accuracy.rmse(predictions)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.1935
```

```
: 0.19350741218895207
```

CF using Non-negative Matrix Factorization

Non-negative matrix factorization (NMF) is an algorithm that addresses dimensionality reduction by breaking down a large, sparse matrix into two smaller, denser matrices.

Its core concept involves decomposing the extensive user-interaction dataset into two compact representations: one for transformed user characteristics and another for transformed item features.

An example is shown below, suppose we have a user-item interaction matrix A with 10000 users and 100 items (10000 x 100), and its element (j, k) represents the rating of item k from user j. Then we could decompose A into two smaller and dense matrices U (10000 x 16) and I (16 x 100). for user matrix U, each row vector is a transformed latent feature vector of a user, and for the item matrix I, each column is a transformed latent feature vector of an item.

Here the dimension 16 is a hyperparameter defines the size of the hidden user and item features, which means now the shape of transposed user feature vector and item feature vector is now 16 x 1.

The magic here is when we multiply the row j of U and column k of matrix I, we can get an estimation to the original rating $\hat{r}$ _jk .

For example, if we perform the dot product user ones row vector in U and item ones column vector in I, we can get the rating estimation of user one to item one, which is the element (1, 1) in the original interaction matrix I .

User-item interaction matrix: **A** 10000 x 100

	item1	**...**	**item100**
user1	...	...	
user2	3.0	3.0	3.0
user3	2.0	2.0	-
user4	3.0	2.0	3.0
user5	2.0	-	-
user6	3.0	-	3.0
...	...	...	

≈

User matrix: **U** 10000 x 16

	feature1	**...**	**feature16**
user1	...	...	...
user2	...	...	...
user3	...	...	...
user4	...	...	...
...	...	...	...
...	...	...	...
user6	...	...	...

X

Item matrix: **I** 16 x 100

	item1	**...**	**item100**
feature1	...	...	...
feature2	...	...	...
...	...	...	...
feature16	...	...	...

Raw data → **Decomposed matrix** → **Model** → **Prediction**

Raw data: User - item - rating - dataframe: user_id, item, rating

Decomposed matrix: Use *surprise* library to decompose full matrix to two smaller and denser ones: user matrix and item matrix

Model: Dot product each row in user matrix with each column in item matrix

Prediction: Make prediction by test data, use RMSE metric to evaluate model performance

User-item interaction matrix: **A** 10000 x 100

	item1	...	item100
user1	...	...	
user2	3.0	3.0	3.0
user3	2.0	2.0	-
user4	3.0	2.0	3.0
user5	2.0	-	-
user6	3.0	-	3.0
...	...	...	

≈

User matrix: **U** 10000 x 16

	feature1	...	feature16
user1	...	...	...
user2	...	...	...
user3	...	...	...
user4	...	...	...
...	...	...	...
...	...	...	...
user6	...	...	...

X

Item matrix: **I** 16 x 100

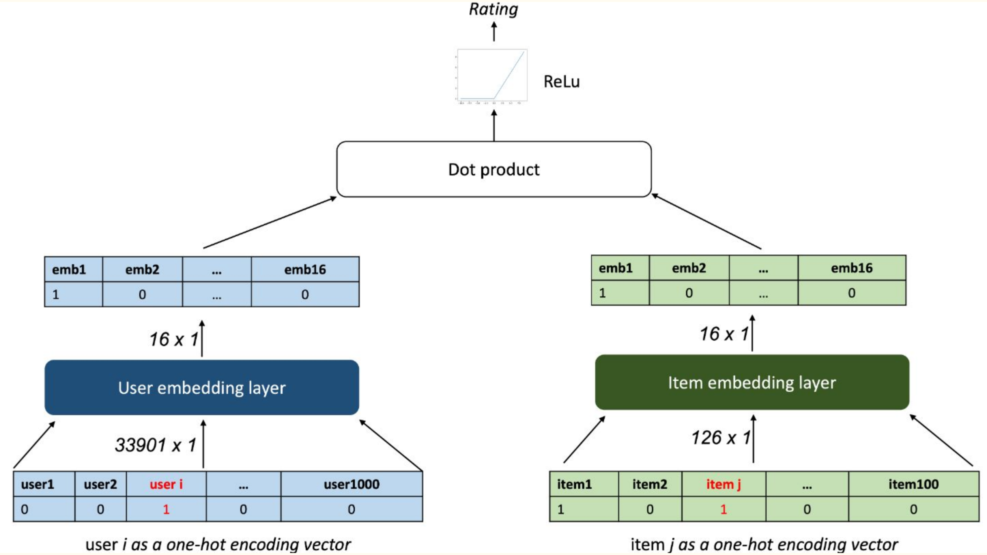
	item1	...	item100
feature1	...	...	...
feature2	...	...	...
...	...	...	...
feature16	...	...	...

```
Processing epoch 39
Processing epoch 40
Processing epoch 41
Processing epoch 42
Processing epoch 43
Processing epoch 44
Processing epoch 45
Processing epoch 46
Processing epoch 47
Processing epoch 48
Processing epoch 49
RMSE: 0.2078
: 0.20782347708297272
```

Course Rating Prediction using Neural Networks

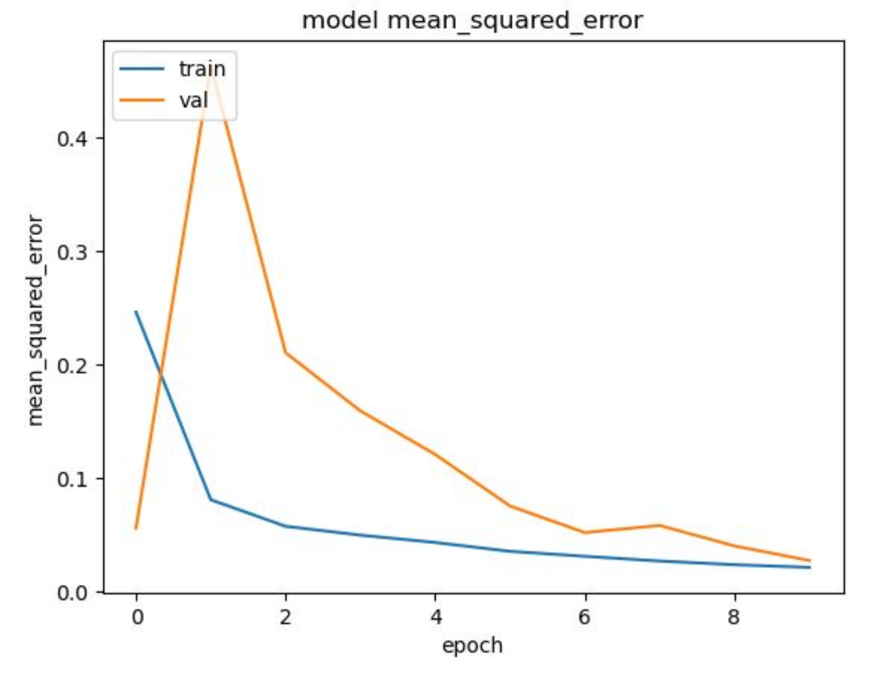
The goal is to create a neural network structure that can take the user and item one-hot vectors as inputs and outputs a rating estimation or the probability of interaction (such as the probability of completing a course).

While training and updating the weights in the neural network, its hidden layers should be able to capture the pattern or features for each user and item. Based on this idea, we can design a simple neural network architecture like the following:



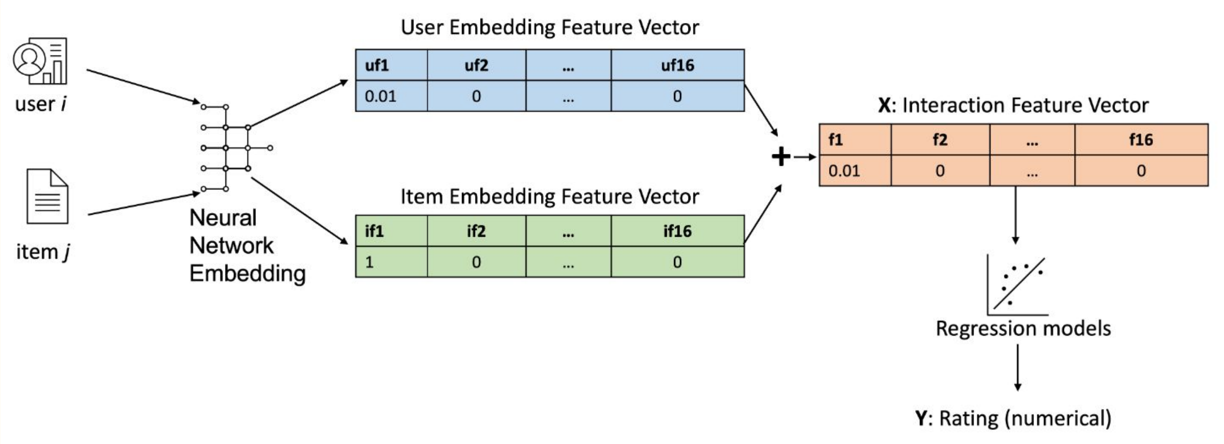
```
Model: "recommender_net"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 user_embedding_layer (Embed  multiple                 542416
 ding)

 user_bias (Embedding)       multiple                  33901

 item_embedding_layer (Embed  multiple                 2016
 ding)

 item_bias (Embedding)       multiple                  126

=================================================================
Total params: 578,459
Trainable params: 578,459
Non-trainable params: 0
```

- Optimizer: Adam
- Loss: Mean Square Error
- Metric: Mean Square Error
- Epoch 12
- Batch size: 512



- Mean squared error: 0.258
- Root mean squared error: 0.508

Regression-Based Rating Score Prediction Using Embedding Features



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.

```python
# Evaluation metrics
mae_lm = metrics.mean_absolute_error(y_test, lm_prediction)
mse_lm =  metrics.mean_squared_error(y_test, lm_prediction)
rmse_lm =  np.sqrt(mse_lm)

print('MAE:', mae_lm)
print('MSE:', mse_lm)
print('RMSE:', rmse_lm)
```

```
MAE: 0.41428838083033687
MSE: 0.9932500760760065
RMSE: 0.9966193235513781
```

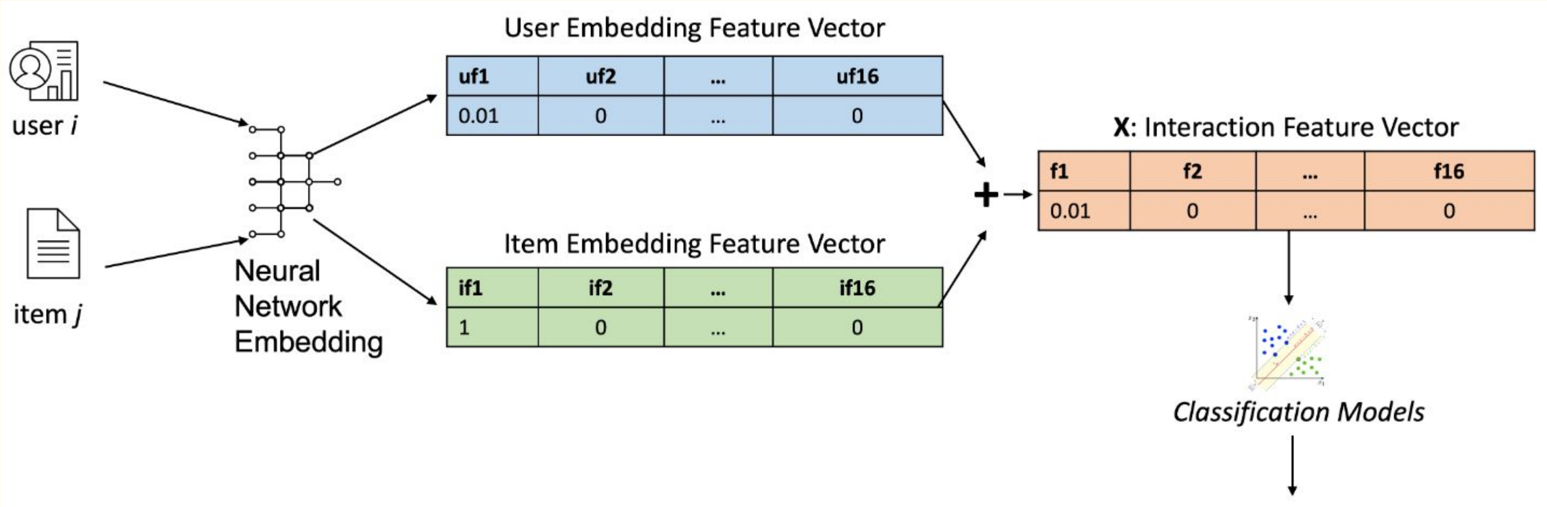
TODO: Try different regression models such as Ridge, Lasso, Elastic

```python
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
```

```python
### WRITE YOUR CODE HERE
rd = ElasticNet()
rd.fit(X_train,y_train)
rd_prediction = rd.predict(X_test)

mae_rd = metrics.mean_absolute_error(y_test, rd_prediction)
mse_rd =  metrics.mean_squared_error(y_test, rd_prediction)
rmse_rd =  np.sqrt(mse_rd)

print('MAE:', mae_rd)
print('MSE:', mse_rd)
print('RMSE:', rmse_rd)
```

```
MAE: 0.4167848022681181
MSE: 1.0000000000000002
RMSE: 1.0
```

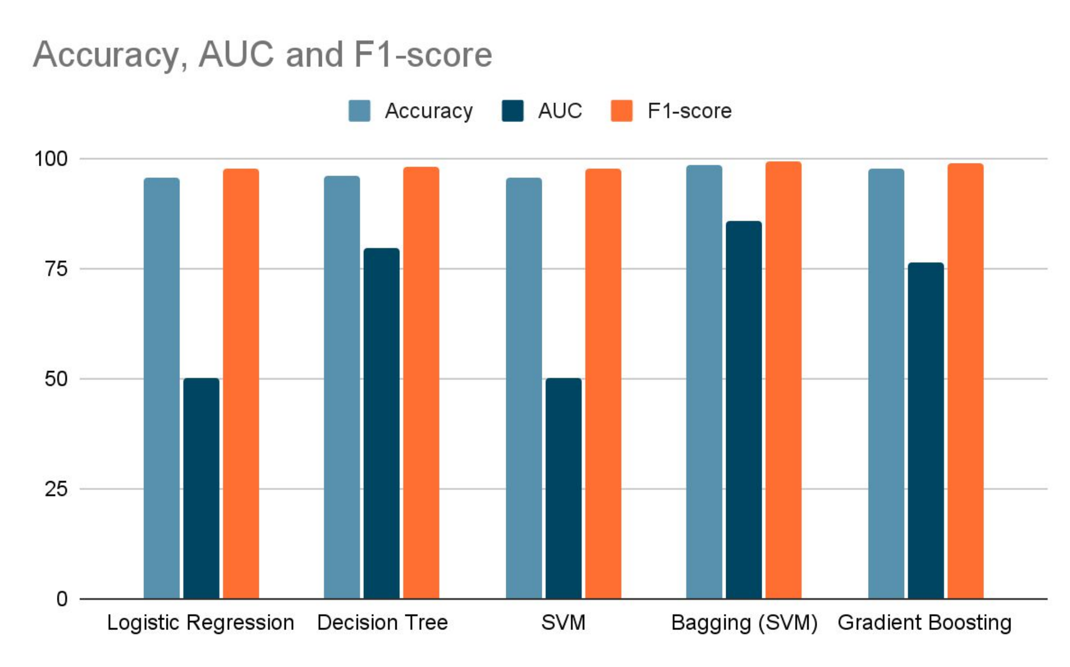
Classification-based Rating Mode Prediction Using Embedding Features

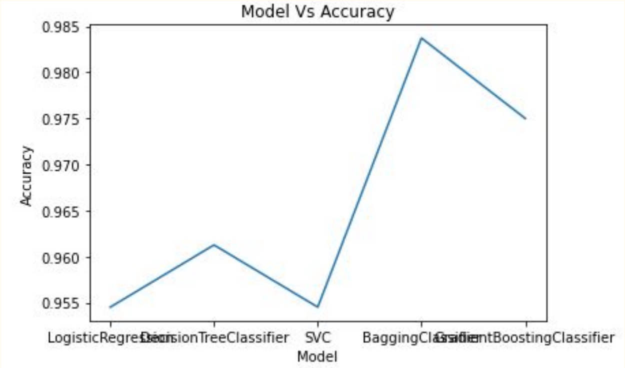


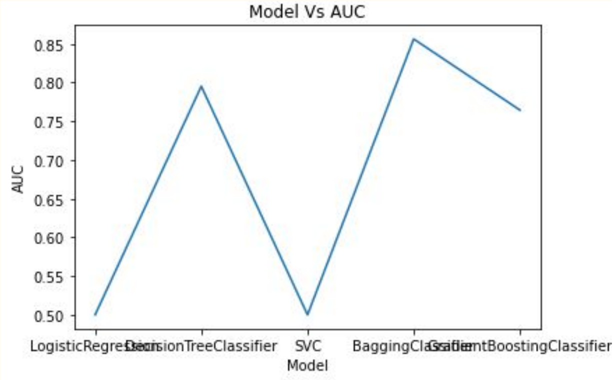
We first extract two embedding matrices out of the neural network, and aggregate them to be a single interaction feature vector as input data X.

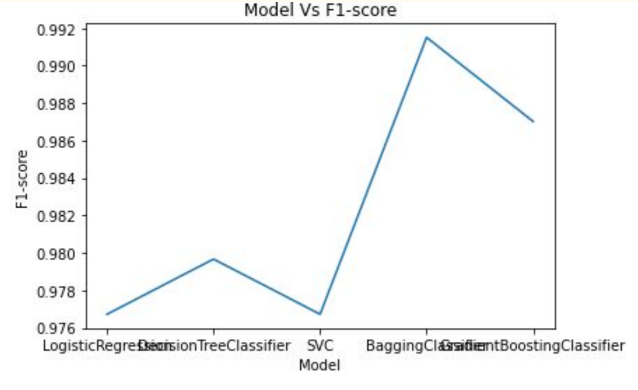
This time, with the interaction label Y as categorical rating mode, we can build classification models to approximate the mapping from X to Y, as shown in the above flowchart.

	Accuracy	AUC	F1-score
Logistic Regression	95.45	0.5	97.67
Decision Tree	96.12	79.51	97.97
SVM	95.45	0.5	97.67
Bagging (SVM)	98.37	85.62	99.15
Gradient Boosting	97.5	76.44	98.7

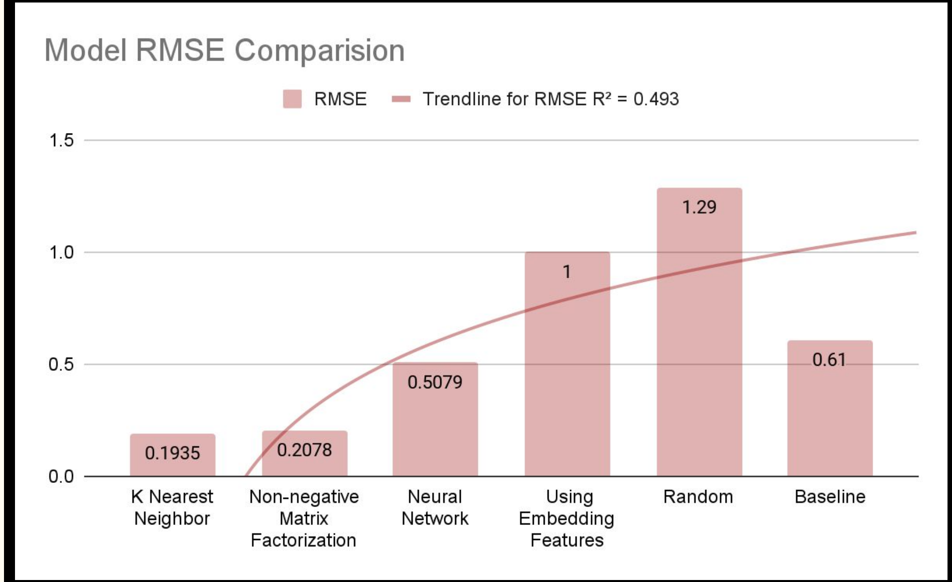



Model Vs Accuracy
Accuracy
0.985
0.980
0.975
0.970
0.965
0.960
0.955
LogisticRegression
DecisionTreeClassifier
SVC
BaggingClassifier
GradientBoostingClassifier
Model

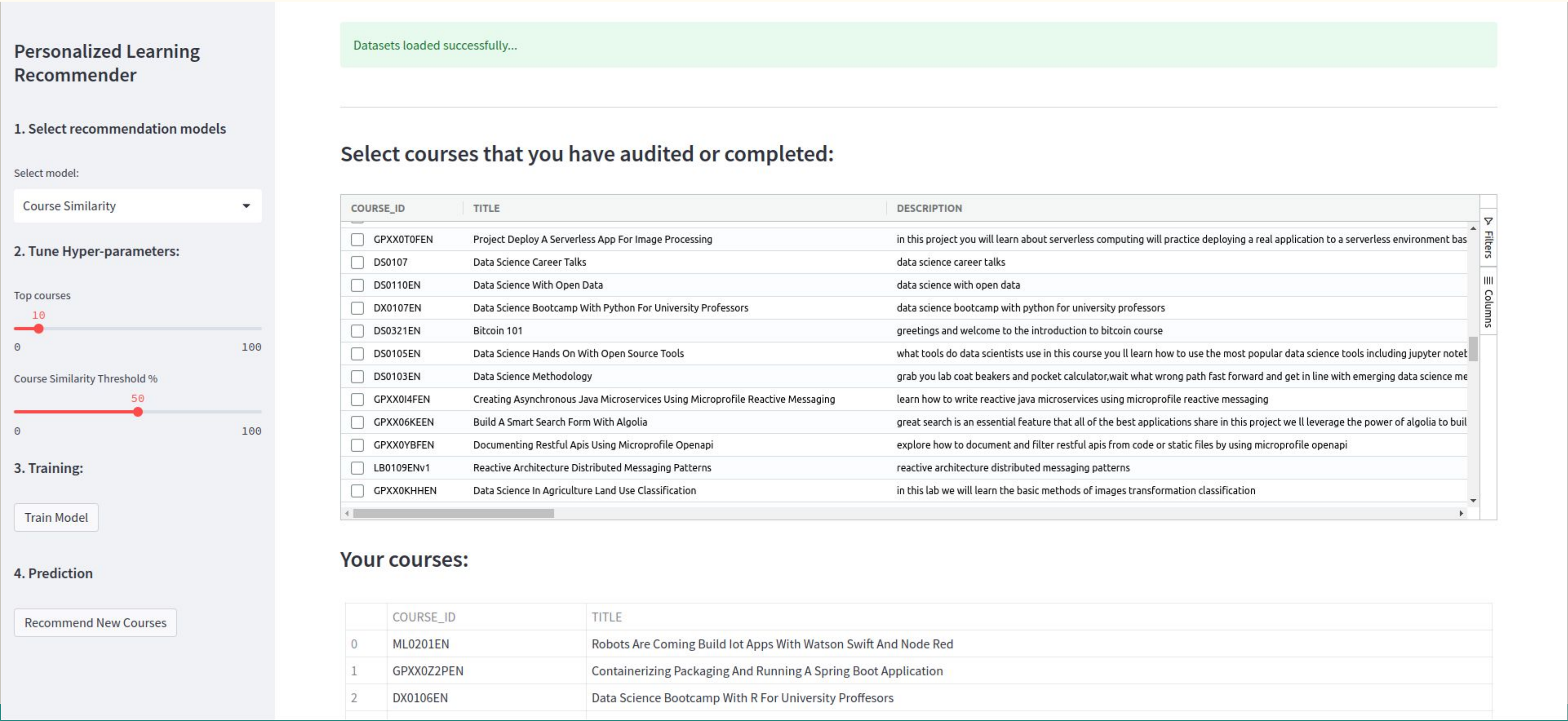

Model Vs AUC
AUC
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
LogisticRegression
DecisionTreeClassifier
SVC
BaggingClassifier
GradientBoostingClassifier
Model


Model Vs F1-score
F1-score
0.992
0.990
0.988
0.986
0.984
0.982
0.980
0.978
0.976
LogisticRegression
DecisionTreeClassifier
SVC
BaggingClassifier
GradientBoostingClassifier
Model

Comparison of Models



Deployment on Streamlit



Personalized Learning Recommender

1. Select recommendation models

Select model:

Course Similarity

2. Tune Hyper-parameters:

Top courses

10

0 100

Course Similarity Threshold %

50

0 100

3. Training:

Train Model

4. Prediction

Recommend New Courses

Your courses:

	COURSE_ID	TITLE
0	ML0201EN	Robots Are Coming Build Iot Apps With Watson Swift And Node Red
1	GPXX0Z2PEN	Containerizing Packaging And Running A Spring Boot Application
2	DX0106EN	Data Science Bootcamp With R For University Proffesors
3	RAVSCTEST1	Scorm Test 1

Recommendations generated!

	SCORE	TITLE	DESCRIPTION
0	0.9476	Data Science Bootcamp	a multi day intensive in person data science bootcamp offered by big data university
1	0.6823	Data Science Bootcamp With Python For University Professors	data science bootcamp with python for university professors
2	0.6685	Data Science Bootcamp With Python For University Professors Advance	data science bootcamp with python for university professors advance
3	0.6499	Data Science Bootcamp With Python	data science bootcamp with python
4	0.6065	Data Science With Open Data	data science with open data

Future work

This project shows how a end-to-end machine learning pipeline work. There are several enhancements can be applied for better accuracy and performance:

- Experience with real customer data
- Apply more pre-processing techniques
- Deal with spare data which can cause full of memory

References

- ➔ Deep Learning, Goodfellow et al
- ➔ Pattern Recognition and Machine Learning, Christopher Bishop
- ➔ Machine Learning, Tom M. Mitchell
- ➔ The Elements of Statistical Learning, Trevor Hastie et al