Capstone Project - Recommendation System
Build a Personalized Online Course Recommender System with Machine Learning

Machine Learning Capstone Project by IBM on Coursera

Name : Mahule Roy
Country : Bharat
Introduction : ML and DL enthusiast

For future reading, I am pleased to invite you for visiting my personal repository of the Recommendation System Capstone Project by IBM on Coursera

Github : github.com/dreamboat26/legendary-mojito

Outline:

1. Capstone Overview
2. Exploratory Data Analysis and Feature Engineering
3. Unsupervised Learning based Recomendation System
4. Supervised Learning based Recomendation System
5. Deploy and showcase models on Streamlit
6. Conclusion and future work
7. Appendix

Capstone Overview –Thought Process

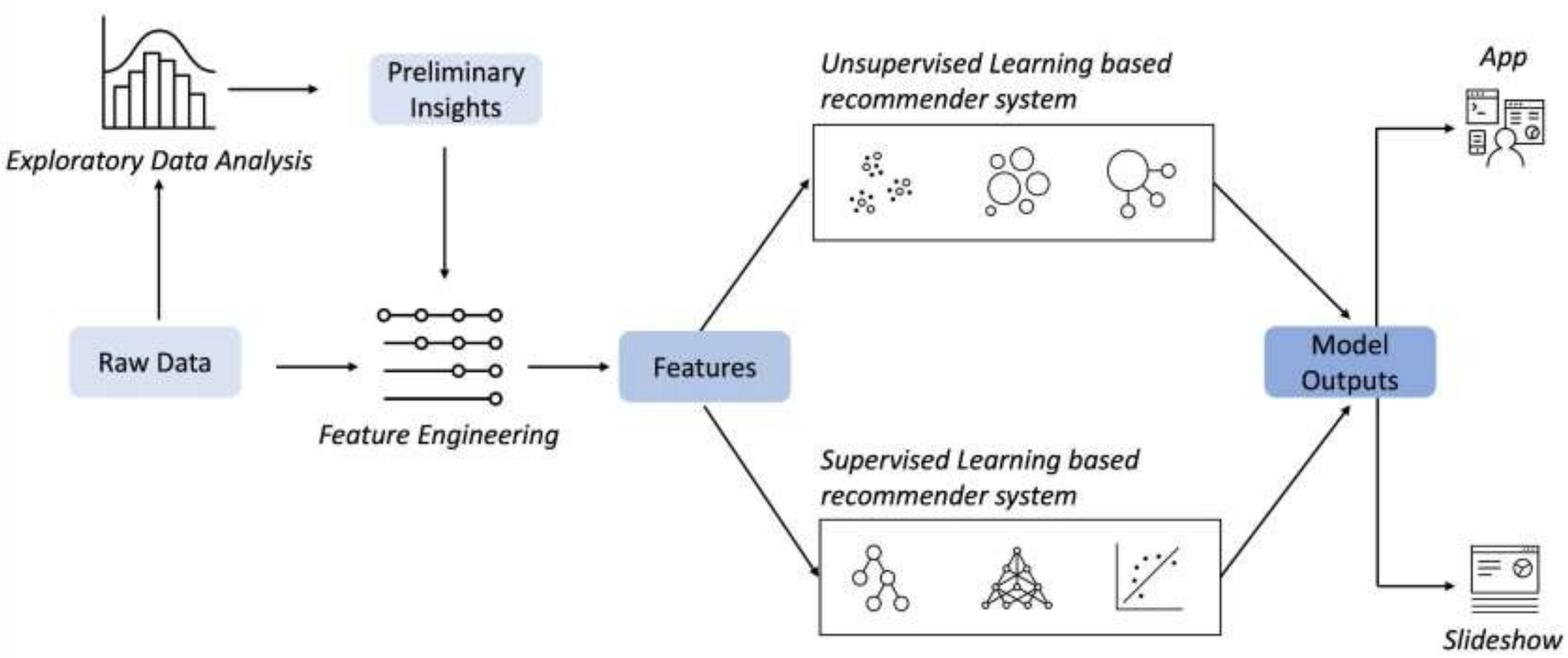
The primary objective of this research project is to enhance the overall learning experience for individuals by efficiently connecting them with courses that align with their interests and educational goals. Concurrently, we aim to analyze how increased interaction between learners and a wider array of courses through our recommender systems may positively impact the company's revenue.

At its current stage, the project is in the Proof of Concept (PoC) phase, prioritizing a rigorous exploration and comparison of various machine learning models. The ultimate goal is to identify the model that exhibits superior performance in offline evaluations. This research endeavor is poised to contribute valuable insights and innovations to the field of personalized learning recommendations.

Capstone Overview - 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.

Capstone Overview - Machine learning pipeline



Exploratory Data Analysis and Feature Engineering

Exploratory Data Analysis

Pipeline:

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

Exploratory Data Analysis

Describe the statistic of data columns

There are 307 courses at total.

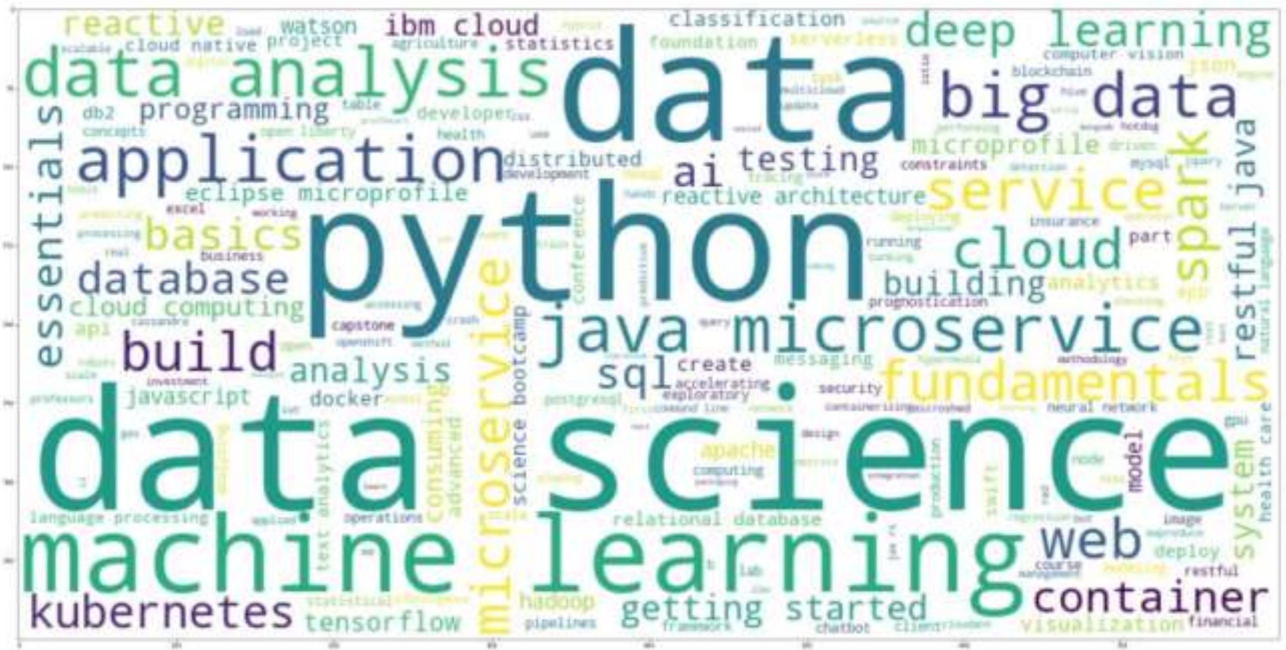
Each course is a vector of genres 1x16.

1 means this course is not related to this genre.

2 means this course is related to this genre.

```
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
```

Exploratory Data Analysis



Identify keywords in course titles using a WordCloud

The omnipresences following by size respectively are:

1. data ->
2. data science ->
3. python ->
4. machine learning ->
5. data analysis ->
6. application ->
7. big data ->
8. java ->
9. microservice
10. web

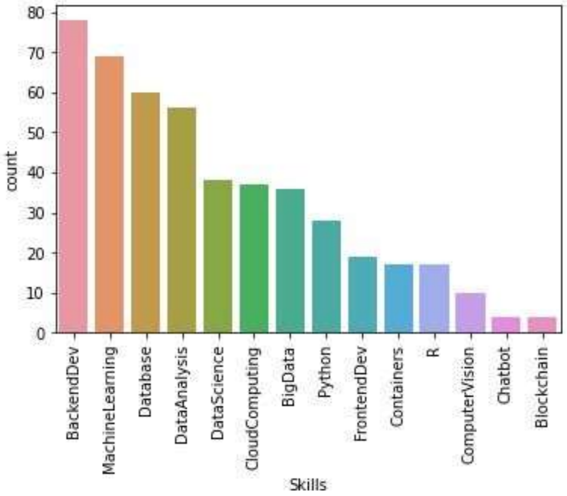
Exploratory Data Analysis

	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

Determine popular course genres

BackendDev, MachineLearning, Database are the utmost popular genres.

While Blockchain, Chatbot, ComputerVision are the most less common ones.

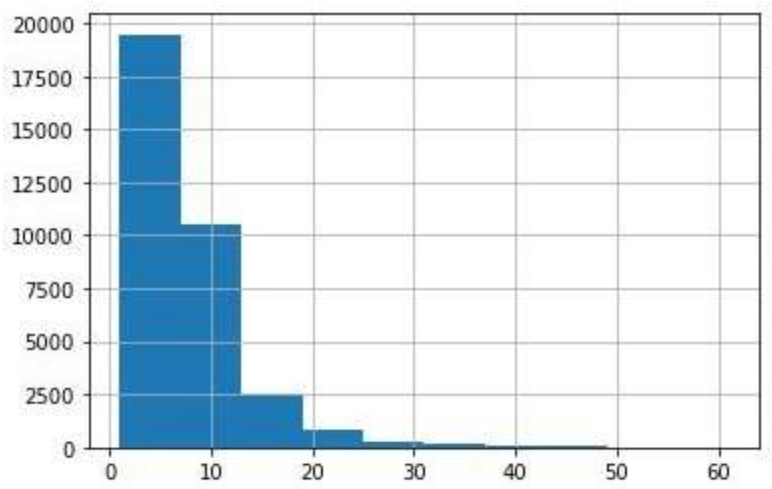


Exploratory Data Analysis

This given histogram illustrates the amount of user rating counts.

Almost users did not rate any courses or rated rarely.

A few exceptional students rated above 40 courses.



The histogram of user rating counts

This below table shows top 20 widespread couses.

9/10 courses of top 10 are belong to data topic.

Only the 4th is the part of software engineer topic.

Top 20 Most Popular Courses

	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

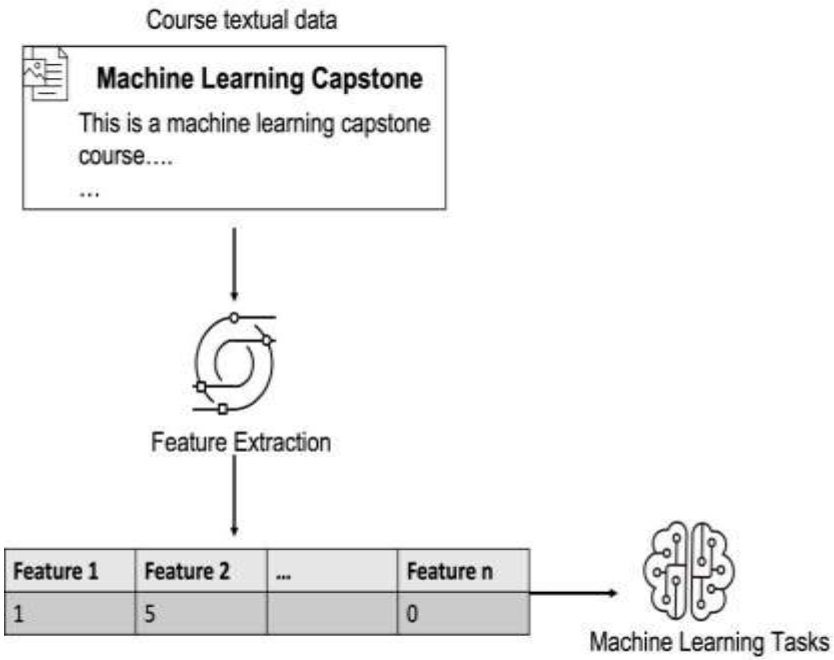
Feature Engineering

Pipeline:

- Extract Bag of Words (BoW) Features from Course Textual Content
- Bag of Words (BoW) features
- BoW dimensionality reduction
- Extract BoW features for course textual content and build a dataset
- Calculate Course Similarity using BoW Features
- Calculate the consine similarity between two example courses
- Find similar courses to the specific course

Feature Engineering

The central objective of recommender systems is to facilitate the discovery of items that align with a user's potential interests. These items can encompass a wide spectrum, including movies, restaurants, or, in our particular case, online courses.
Machine learning algorithms, however, require a numeric representation of items to operate effectively. Consequently, our initial step involves feature extraction and the mathematical representation of these items, often in the form of feature vectors.
It's noteworthy that numerous items are characterized by textual data, encompassing elements like movie or course titles and descriptions. Given that machine learning algorithms cannot directly process textual data, our research focus converges on the transformation of raw text into numeric feature vectors. This pivotal process forms the bedrock of our efforts in enhancing recommender systems for the domain of online education and aligns with the broader field of natural language processing and machine learning.



Feature Engineering - BoW

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]
```

Feature Engineering - BoW

Bag of Words (BoW) features

```
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	

Feature Engineering - BoW

BoW dimensionality reduction

- A document may contain tens of thousands of words which makes the dimension of the BoW feature vector huge.
- To reduce the dimensionality, one common way is to filter the relatively meaningless tokens such as stop words or sometimes add position and adjective words.

Feature Engineering - BoW

BoW dimensionality reduction

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.

Feature Engineering - BoW

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
rate': 195, 'digital': 196, 'essentials': 197, 'hands': 198, 'integration': 199, 'pak': 200, 'received': 201, 'short': 202, 'analytics': 203, 'assemble': 204, 'base': 205, 'basics': 206,
```

Feature Engineering - BoW

Extract BoW features for course textual content and build a dataset

Create a new course_bow dataframe based on the extracted BoW features. he new dataframe needs to include the following columns (you may include other relevant columns as well):

- '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

	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

Feature Engineering - Course Similarity using BoW Features

The quantification of item similarity constitutes a fundamental pillar underpinning a plethora of recommendation algorithms, with particular salience in the realm of content-based recommendation systems. This is manifestly evident when considering the scenario wherein a novel course offering is assessed for its resemblance to courses in which a user has previously enrolled. In this context, the fundamental principle at play is the identification of latent thematic affinities between items, thereby facilitating the tailored presentation of recommendations to users. Additionally, the concept extends to the domain of user-to-user similarity, wherein the parallelism of interests between distinct users becomes pivotal.

In the domain of similarity measurement, a diverse array of mathematical techniques comes into play, each tailored to the specific characteristics of the data at hand. Among these techniques, notable exemplars include the cosine similarity, the Jaccard index, and the Euclidean distance. These methods are versatile, capable of handling not only pairwise comparisons between two vectors but also, in certain scenarios, comparisons involving sets, matrices, or tensors.

In the preceding section, we delved into the extraction of Bag of Words (BoW) features from the textual content of courses. With the BoW feature vectors in our possession, we are bestowed with a facile means to engage in similarity measurement. This process, graphically represented in the forthcoming figure, furnishes us with a means to gauge the proximity of courses in terms of their thematic content, thus enabling the discernment of courses that may be of interest to users based on their historical enrollment patterns.

This approach conveys a research-oriented perspective, which leverages sophisticated mathematical and computational methods to glean insights and facilitate the refinement of recommendation systems.

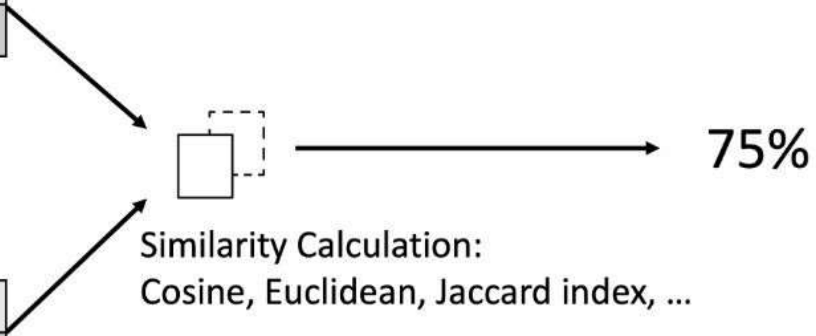
Feature Engineering - Course Similarity using BoW Features

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, ...

Unsupervised Learning based Recomendation System

Unsupervised Learning based Recomendation System

Outline:

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

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

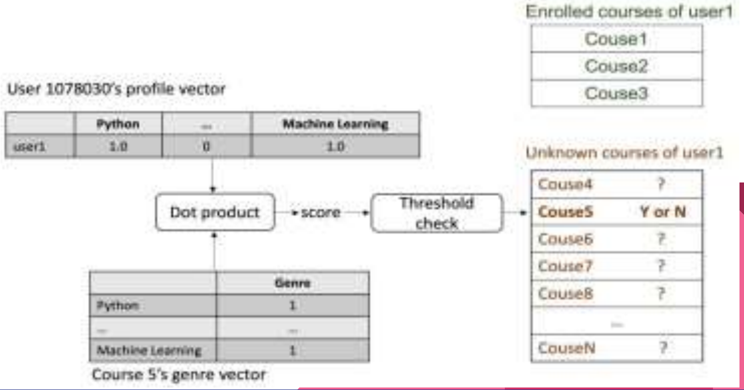
Preeminent among recommendation systems is the ubiquitous content-based approach, wherein items are judiciously suggested to users in consonance with their distinct profiles. The user's profile, a digital reflection of their idiosyncratic preferences and proclivities, is meticulously crafted through the assimilation of user-generated ratings and interactions. These interactions encompass a gamut of user behaviors, including the frequency of clicks, the proclivity for liking particular items, and the navigation patterns across the digital terrain.

The crux of this recommendation paradigm pivots upon the discernment of similarity, an intricate calculus rooted in the congruity of items based on their intrinsic content characteristics. Content, in this context, encapsulates a panoply of attributes, such as an item's categorical classification, tags, genres, and various other features that epitomize its essence.

In essence, the content-based recommendation system orchestrates a harmonious interplay between user profiles and the manifold facets of item content, with the ultimate aim of orchestrating a symphony of personalized recommendations. This process represents the apotheosis of data-driven refinement, wherein algorithms tirelessly scrutinize and juxtapose content attributes to curate a bespoke selection of items that resonate with the user's unique tastes and preferences.

1. Content-based Course Recommender System using User Profile and Course Genres - Pipeline

Raw data	Feature engineering	Get recommend score	Recomendation
Course genres dataframe: `course_id, title, [genre_x, genre_y,...]` User dataframe: `user_id, [genre_interest_x, genre_interest_y,...]`	Analysis data Drop the exceptions Normalise data	Use dot product between single user vector and specific course to get recommend score	Make prediction by comparing score with determine threshold



1. Content-based Course Recommender System using User Profile and Course Genres - Result

	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

⊗

	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

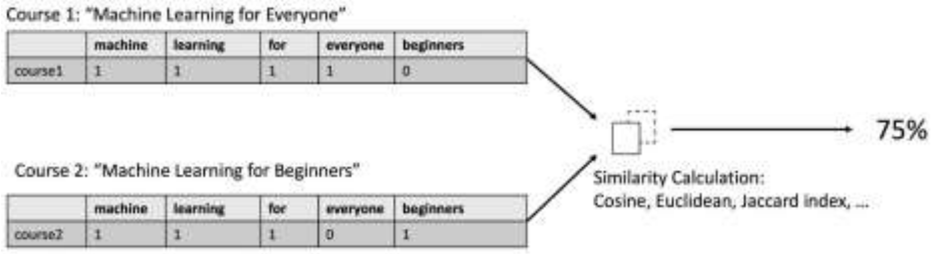
2. Content-based Course Recommender System using Course Similarities

In this lab, we focus on content-based recommender systems, which rely on item similarity calculations. We measure item similarity based on content features, including course genres and Bag of Words (BoW) values representing course text.

Our goal is to recommend new courses similar to those a user has already enrolled in. We use course genres and BoW features to find courses that align with the user's existing choices, using advanced similarity calculations. This process offers personalized recommendations tailored to the user's educational preferences.

2. Content-based Course Recommender System using Course Similarities - Pipeline

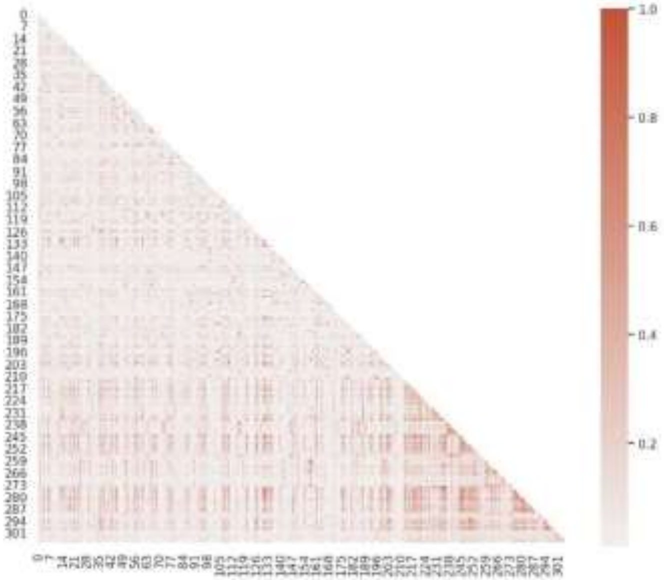
Raw data	Features	Score	Prediction
1.Course similarity matrix 2. Course dataframe: `course_id, title, description`	Visualise the similarly metric Handle data text in title and description • Remove stop words • word2vec	Turn a course description to a vector Check similarity by compare 2 vector	Determine rely on similar score



2. Content-based Course Recommender System using Course Similarities

Course similarity matrix:

	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
...	...	...	...	...	...



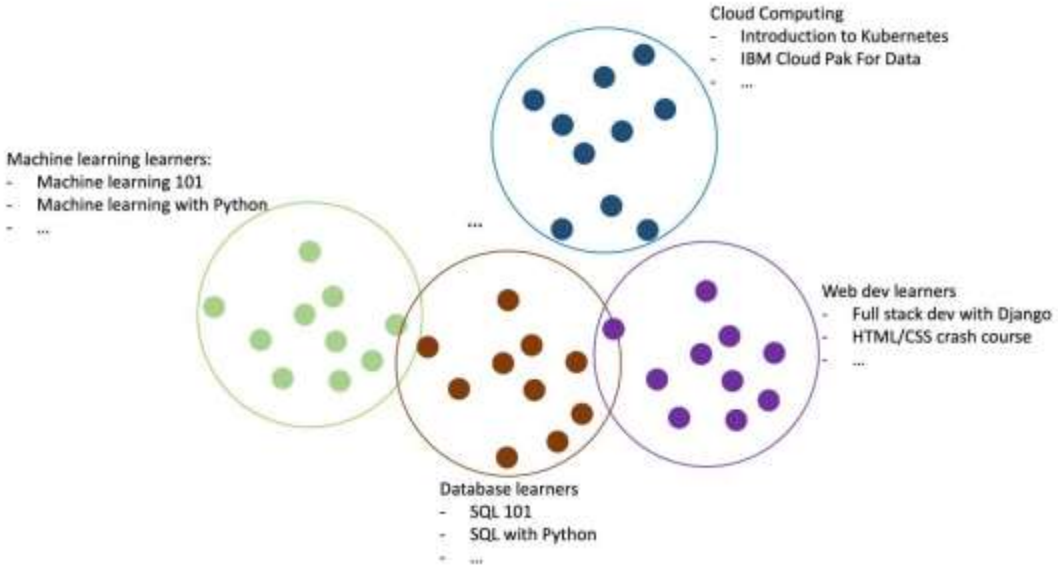
2. Content-based Course Recommender System using Course Similarities - Result

Generate course recommendations based on course similarities for all test users

	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

3. Clustering based Course Recommender System

We could perform clustering algorithms such as K-means or DBSCAN to group users with similar learning interests. For example, in the below user clusters, we have user clusters whom have learned courses related to machine learning, cloud computing, databases, and web development, etc.



3. Clustering based Course Recommender System - Pipeline

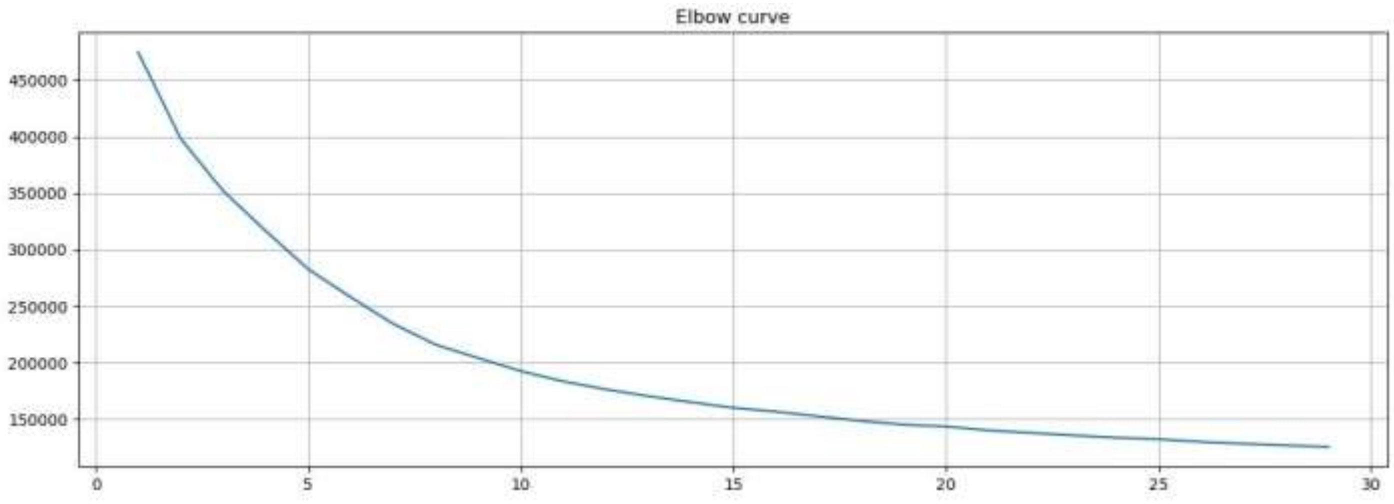
Raw data → Features → Clusters → Prediction

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

3. Clustering based Course Recommender System

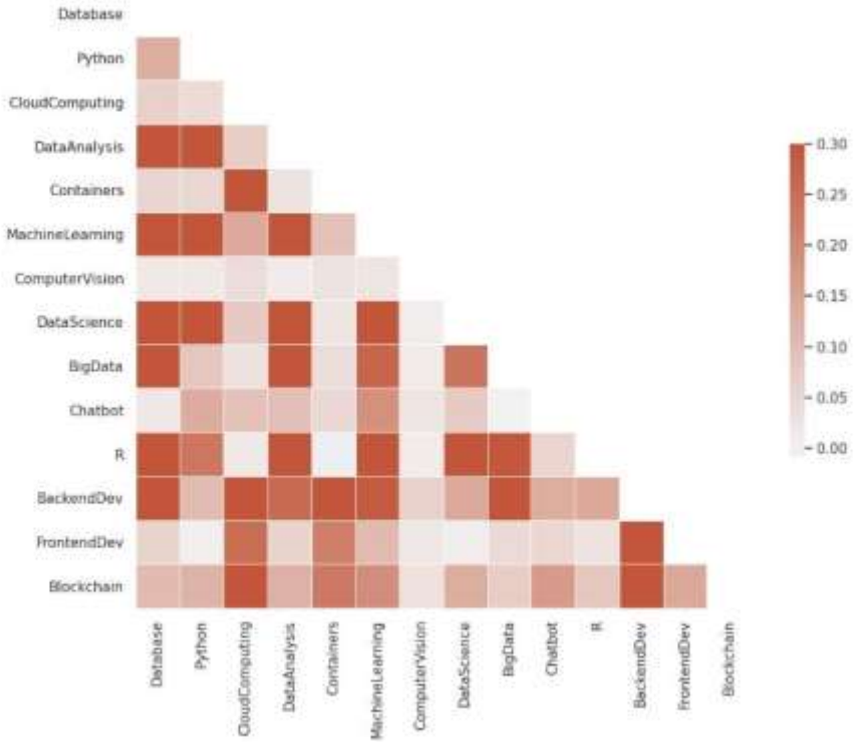
For KMeans algorithm, one important hyperparameter is the number of clusters *n_cluster*, and a good way to find the optimized *n_cluster* is using to grid search a list of candidates and find the one with the best or optimized clustering evaluation metrics such as minimal sum of squared distance.

Grid search the optimized n_cluster for KMeans() model



3. Clustering based Course Recommender System

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

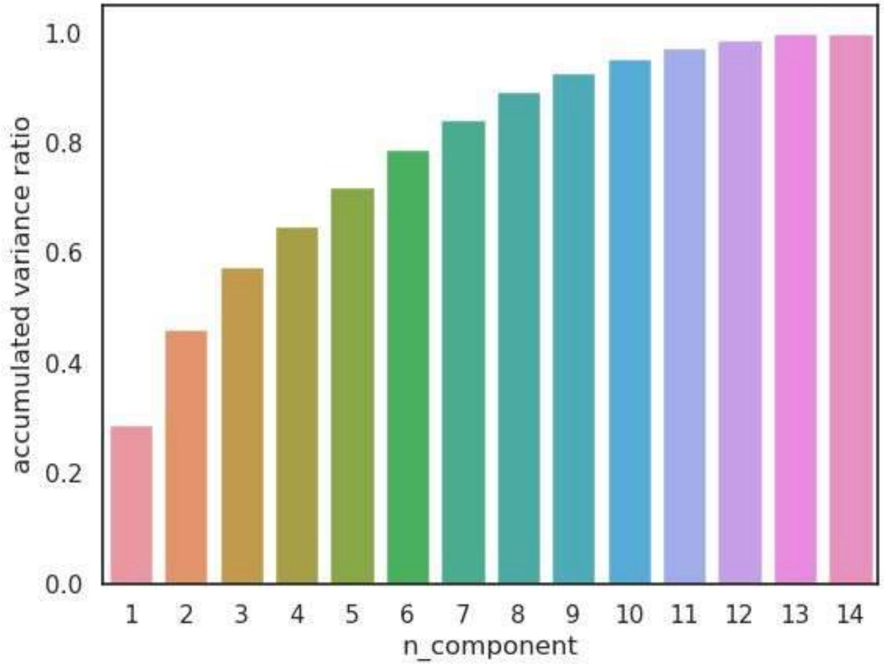


3. Clustering based Course Recommender System

Apply the PCA() provided by scikit-learn to find the main components in user profile feature vectors and see if we can reduce its dimensions by only keeping the main components.

If the accumulated variances ratio of a candidate n_components is larger than a threshold, e.g., 90%, then we can say the transformed n_components could explain about 90% of variances of the original data variance and can be considered as an optimized components size.

We select n_component = 8, due to the minimal ratio > 0.9



3. Clustering based Course Recommender System

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 KMean 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

3. Clustering based Course Recommender System

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

```
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']
```

Supervised Learning based Recomendation System

Supervised Learning based Recomendation System

Outline:

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

1. 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

1. CF using K Nearest Neighbor

User-based collaborative filtering hinges on the identification of users who exhibit similarity in their preferences. This method bears a resemblance to the user clustering approach we previously employed, wherein explicit user profiles were leveraged to ascertain user similarity. However, a pivotal challenge arises when these user profiles are not readily available. In such cases, the question arises: How can we determine the similarity between users?

In the realm of collaborative filtering-based recommender systems, the quintessential dataset format takes the shape of a 2-D matrix known as the user-item interaction matrix. Within this matrix, rows correspond to user identities or indices, columns correspond to item identities or indices, and the matrix elements (i, j) denote the ratings assigned by user i to item j.

Below is a simple example of a user-item interaction matrix:

User-Item interaction matrix

	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
...	...	...	...	...	

Similar users

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

1. CF using K Nearest Neighbor

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:

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

1. CF using K Nearest Neighbor - Pipeline & Result

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

	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

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

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

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

```
: 0.19350741218895207
```

2. CF using Non-negative Matrix Factorization

Non-negative matrix factorization (NMF) is a powerful dimensionality reduction algorithm designed to address challenges posed by large and sparse matrices. Its core concept involves decomposing a substantial user-item interaction matrix into two smaller yet denser matrices. One of these matrices encapsulates the transformed user features, while the other encapsulates the transformed item features.

NMF stands as an effective solution to the intricacies posed by extensive datasets. Its fundamental premise lies in the extraction of meaningful patterns and representations from the original data, thereby enabling more efficient processing and analysis. By breaking down the large and sparsely populated user-item interaction matrix into these smaller, more compact representations, NMF facilitates the discovery of latent structures within the data, paving the way for enhanced recommendation systems and data analysis.

2. CF using Non-negative Matrix Factorization

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 $\hat{}_{jk}$.

For example, if we preform 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.

2. CF using Non-negative Matrix Factorization

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	...	...	...

2. CF using Non-negative Matrix Factorization - Pipeline & Result

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

```
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
```

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

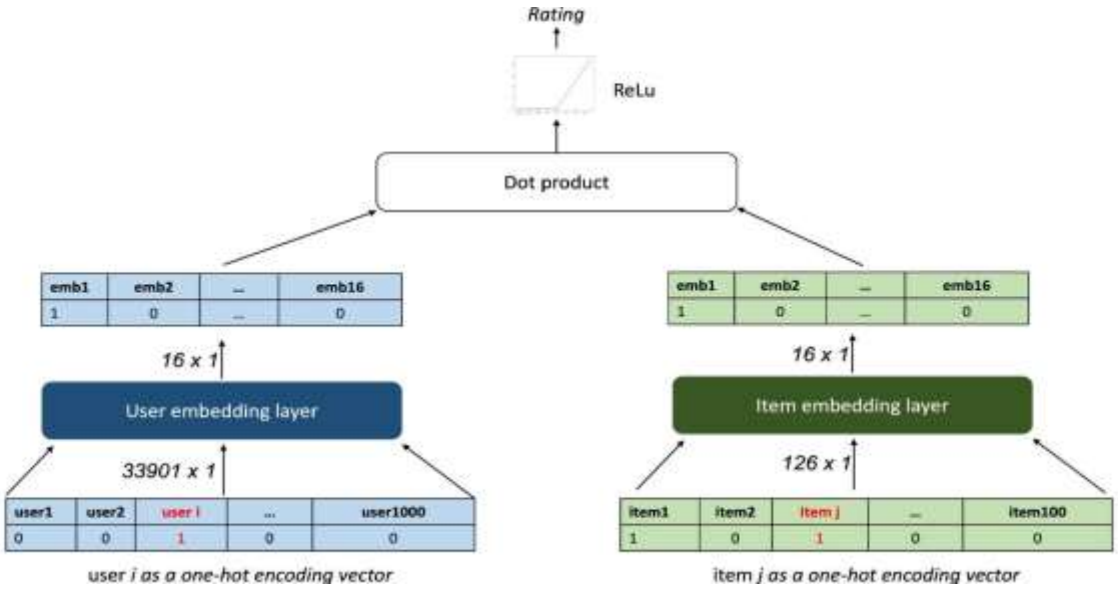
Item matrix: **I** 16 x 100

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

3. 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:



3. Course Rating Prediction using Neural Networks

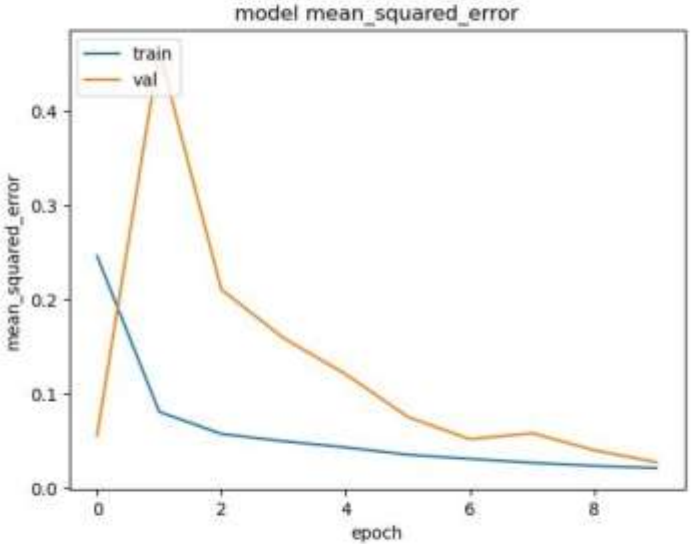
Model:

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

```
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
```

3. Course Rating Prediction using Neural Networks - Result

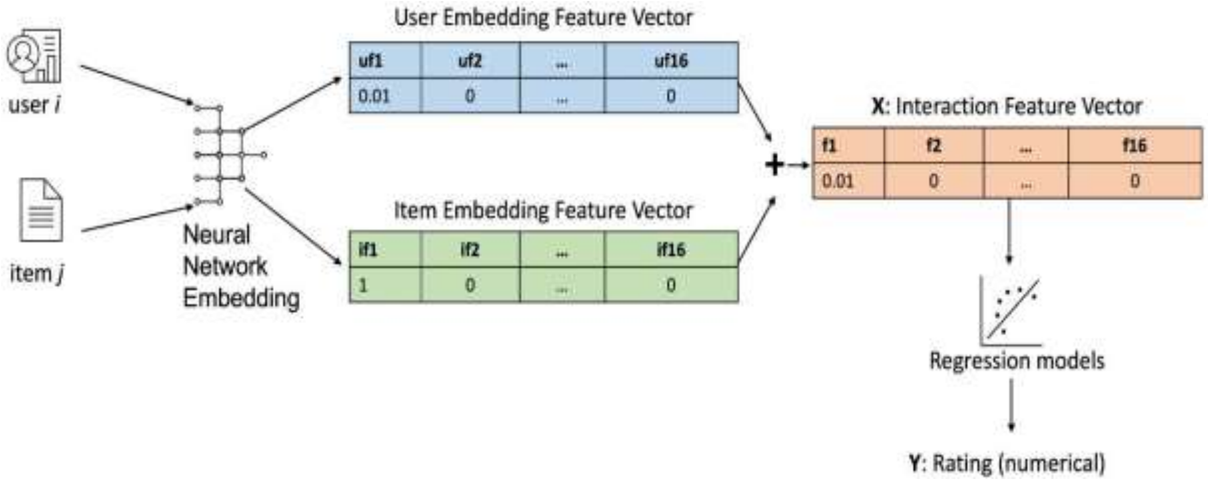
Train and Validate:



Result on test data:

Mean squared error: 0.258
Root mean squared error: 0.508

4. 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.

4. Regression-Based Rating Score Prediction Using Embedding Features - Result

```
# 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: 0.4142883808303368 7
MSE: 0.9932500760760065
RMSE: 0.9966193235513781
```

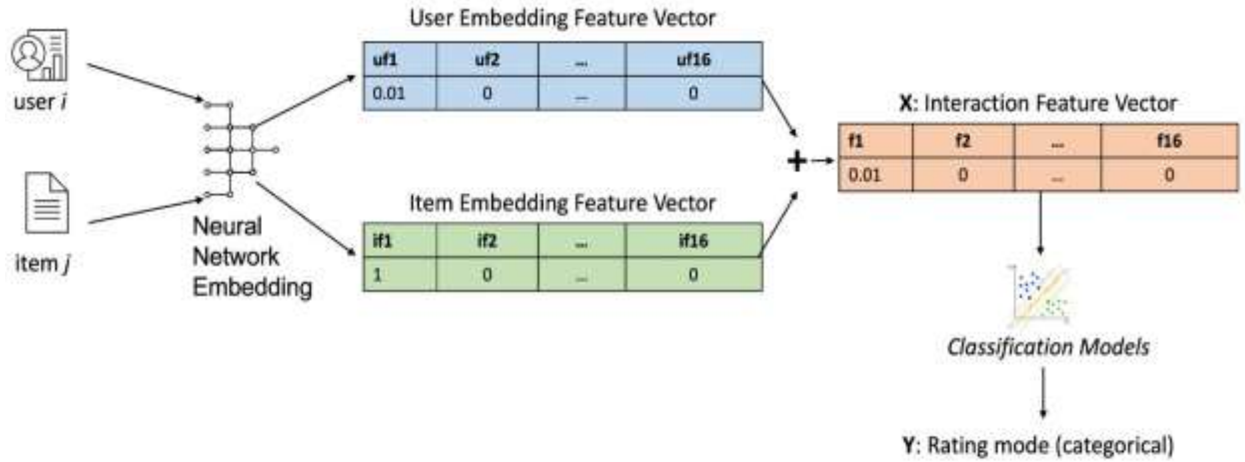
TODO: Try different regression models such as Ridge, Lasso, Elastic

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

```
### 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: 0.416784802268 1181
MSE: 1.0000000000000002
RMSE: 1.0
```

Classification-based Rating Mode Prediction using Embedding Features

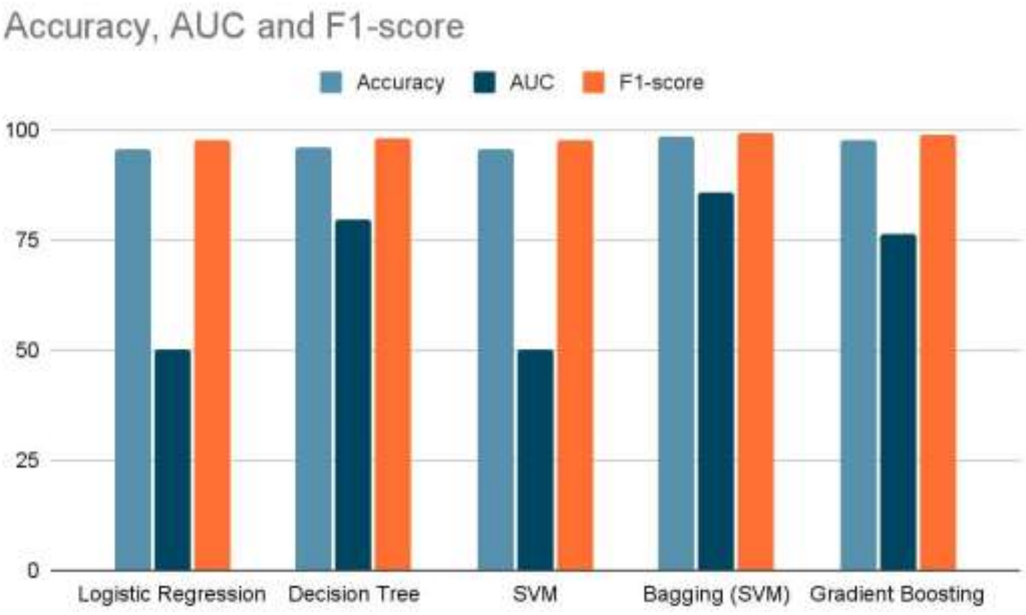


We first extract two embedding matrices out of the neural network, and aggregate them to be a single interaction feature vector as input data X.

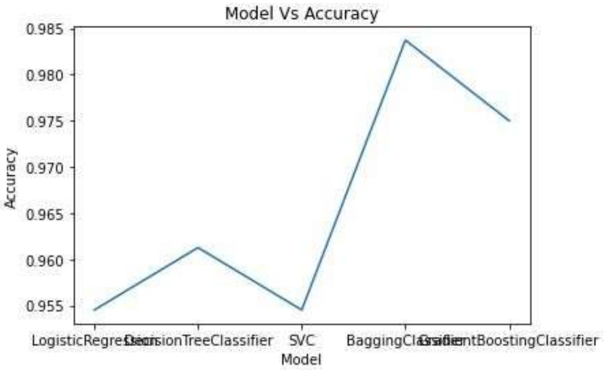
This time, with the interaction label Y as categorical rating mode, we can build classification models to approximate the mapping from X to Y, as shown in the above flowchart.

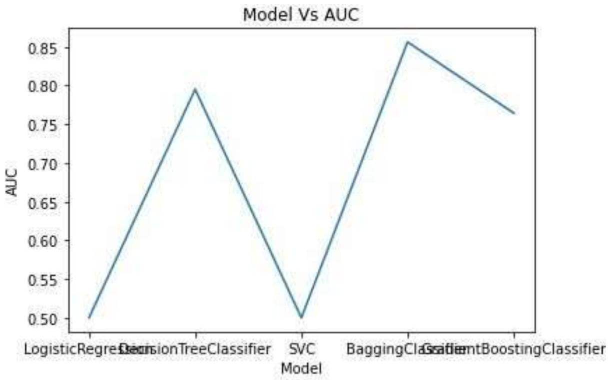
5. Classification-based Rating Mode Prediction using Embedding Features

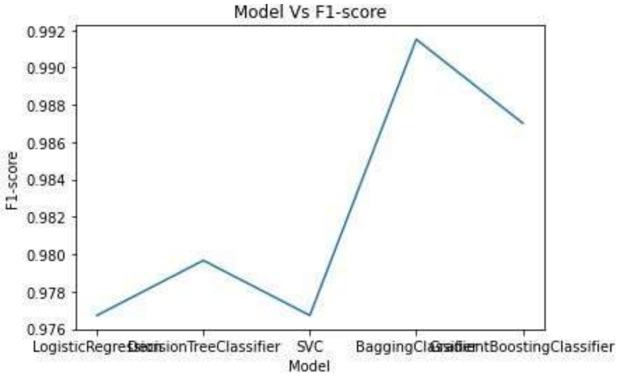
	Accuracy	AUC	F1-score
Logistic Regression	95.45	0.5	97.67
Decision Tree	96.12	79.51	97.97
SVM	95.45	0.5	97.67
Bagging (SVM)	98.37	85.62	99.15
Gradient Boosting	97.5	76.44	98.7



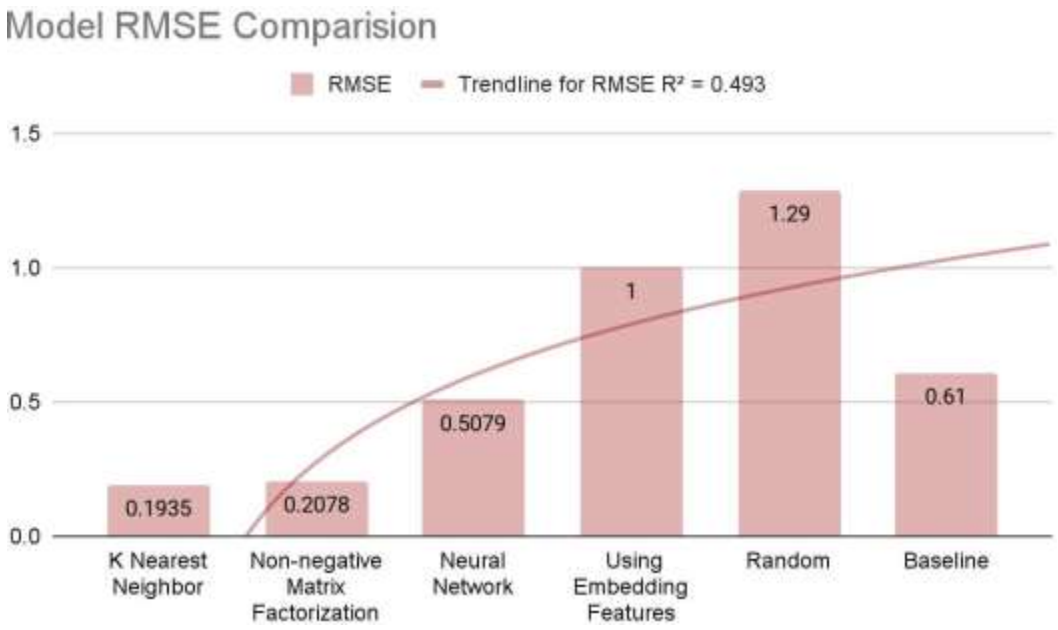
5. Classification-based Rating Mode Prediction using Embedding Features







Compare the performance of collaborative-filtering models

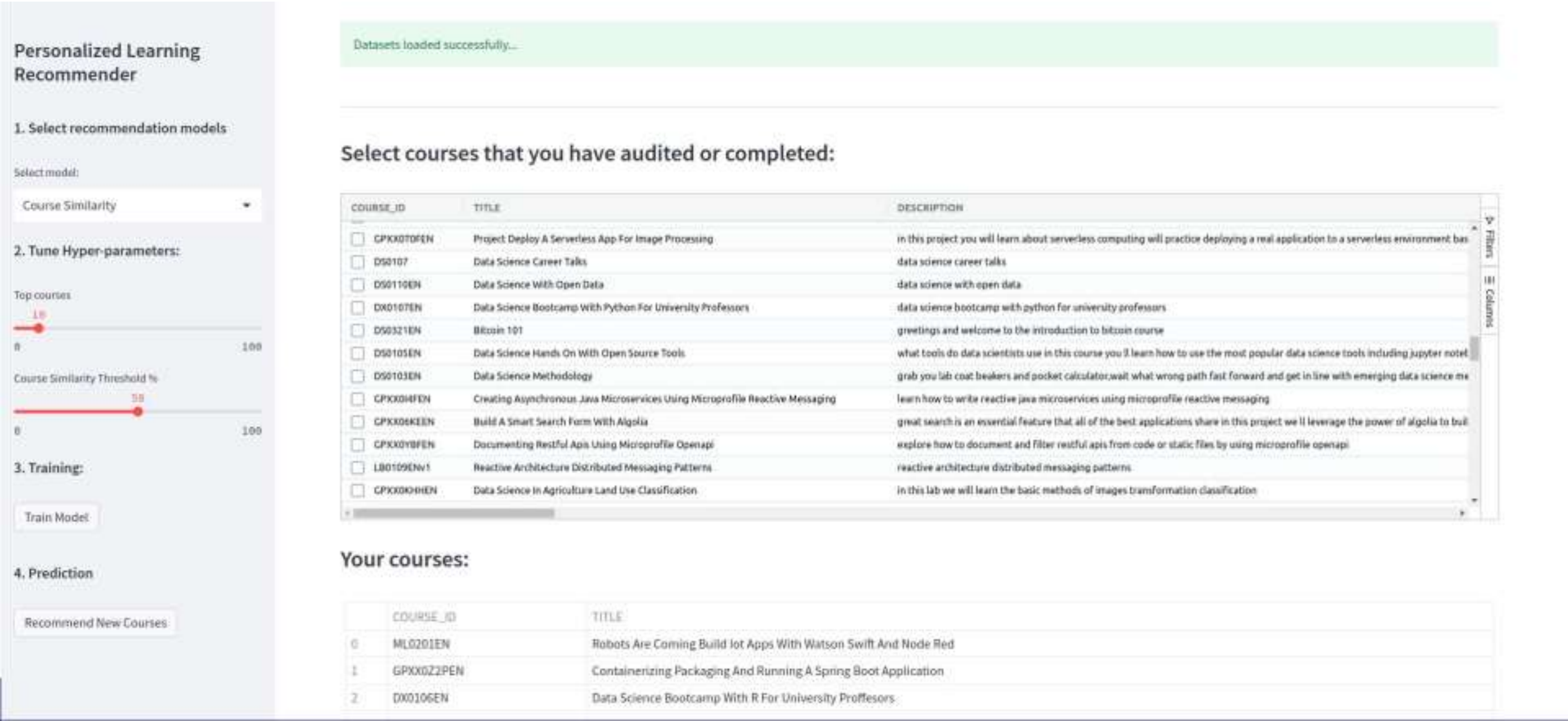


Insights:

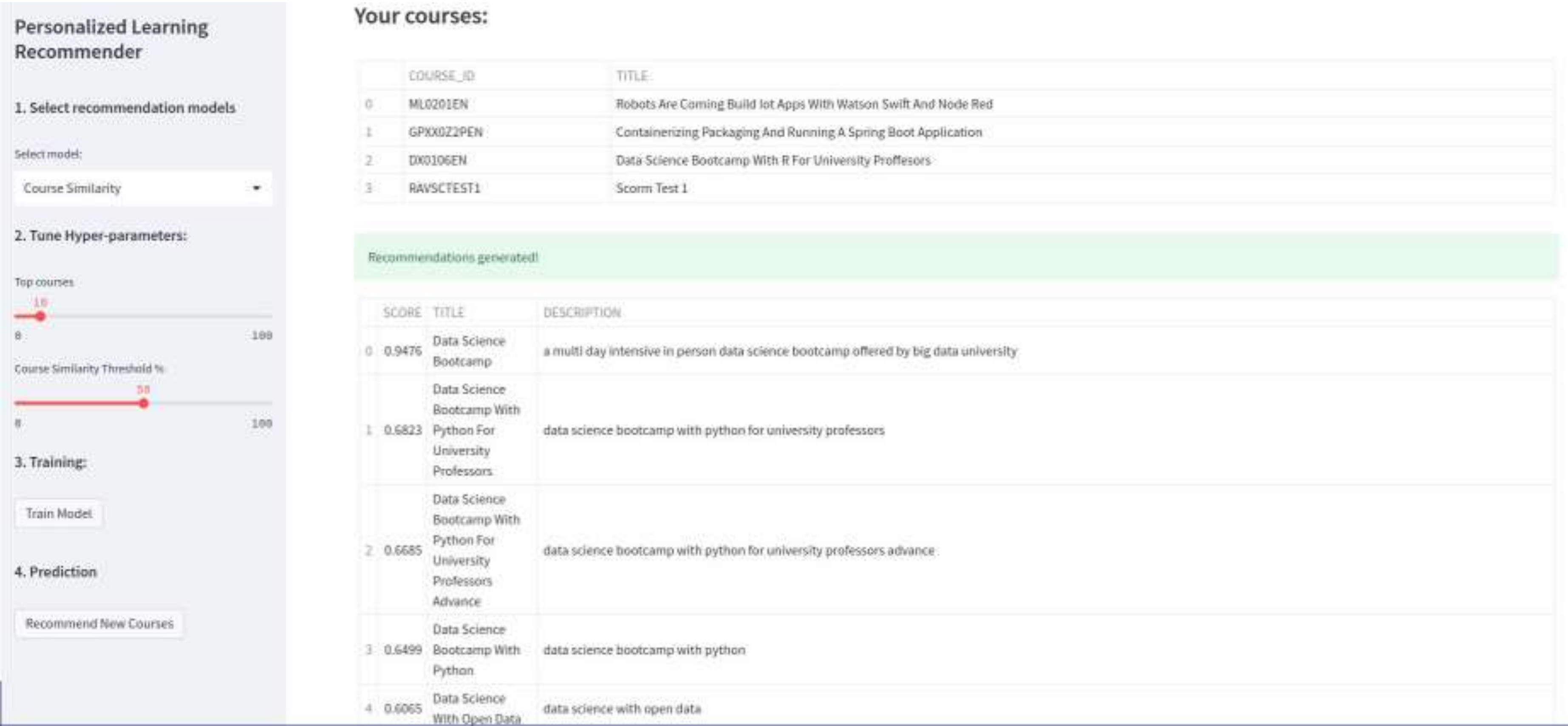
- Random prediction is obviously worst
- KNN method have

Deploy and showcase models on Streamlit

Deploy and showcase models on Streamlit



Deploy and showcase models on Streamlit



Future work

This project showcases an end-to-end machine learning pipeline that fulfills the course creator's requirements. However, there is room for improvement:

Real Data Integration: Incorporate actual customer data for a more authentic and relevant model.

Advanced Pre-processing: Implement advanced data cleaning and feature engineering techniques to enhance data quality.

Sparse Data Management: Develop strategies to handle memory issues caused by sparse data efficiently.

These enhancements will contribute to better accuracy and a more robust solution.

Appendix

- ➔ Author: Mahule Roy (github, linkedin)
- ➔ Github repository: IBM Machine Learning

Thanks for your reading!