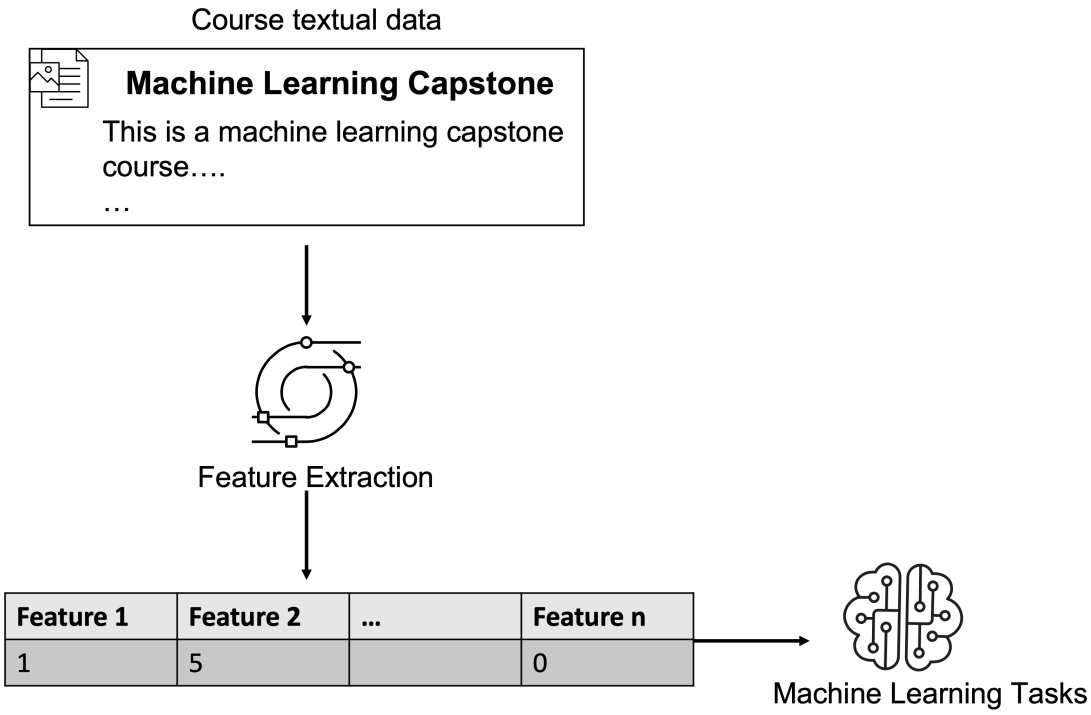
Extract Bag of Words (BoW) Features from Course Textual Content

Estimated time needed: **60** minutes

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.



In this lab, you will be learning to extract the bag of words (BoW) features from course titles and descriptions. The BoW feature is a simple but effective feature characterizing textual data and is widely used in many textual machine learning tasks.

Objectives

After completing this lab you will be able to:

- Extract Bag of Words (BoW) features from course titles and descriptions
- Build a course BoW dataset to be used for building a content-based recommender system later

---

Prepare and setup the lab environment

First, let's install and import required libraries:

In [2]:
```
!pip install nltk==3.6.7
!pip install gensim
!pip install scipy==1.10
!pip install pandas
!pip install nltk
!pip install matplotlib
```

```
Requirement already satisfied: nltk==3.6.7 in /opt/conda/lib/python3.12/site-pack
ages (3.6.7)
Requirement already satisfied: click in /opt/conda/lib/python3.12/site-packages
(from nltk==3.6.7) (8.3.1)
Requirement already satisfied: joblib in /opt/conda/lib/python3.12/site-packages
(from nltk==3.6.7) (1.5.3)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.12/site-
packages (from nltk==3.6.7) (2025.11.3)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.12/site-packages (f
rom nltk==3.6.7) (4.67.1)
Requirement already satisfied: gensim in /opt/conda/lib/python3.12/site-packages
(4.4.0)
Requirement already satisfied: numpy>=1.18.5 in /opt/conda/lib/python3.12/site-pa
ckages (from gensim) (2.4.1)
Requirement already satisfied: scipy>=1.7.0 in /opt/conda/lib/python3.12/site-pac
kages (from gensim) (1.17.0)
Requirement already satisfied: smart_open>=1.8.1 in /opt/conda/lib/python3.12/sit
e-packages (from gensim) (7.5.0)
Requirement already satisfied: wrapt in /opt/conda/lib/python3.12/site-packages
(from smart_open>=1.8.1->gensim) (2.0.1)
ERROR: Ignored the following yanked versions: 1.11.0, 1.14.0rc1
ERROR: Could not find a version that satisfies the requirement scipy==1.10 (from
versions: 0.8.0, 0.9.0, 0.10.0, 0.10.1, 0.11.0, 0.12.0, 0.12.1, 0.13.0, 0.13.1,
0.13.2, 0.13.3, 0.14.0, 0.14.1, 0.15.0, 0.15.1, 0.16.0, 0.16.1, 0.17.0, 0.17.1,
0.18.0, 0.18.1, 0.19.0, 0.19.1, 1.0.0, 1.0.1, 1.1.0, 1.2.0, 1.2.1, 1.2.2, 1.2.3,
1.3.0, 1.3.1, 1.3.2, 1.3.3, 1.4.0, 1.4.1, 1.5.0, 1.5.1, 1.5.2, 1.5.3, 1.5.4, 1.6.
0, 1.6.1, 1.9.2, 1.9.3, 1.11.0rc1, 1.11.0rc2, 1.11.1, 1.11.2, 1.11.3, 1.11.4, 1.1
2.0rc1, 1.12.0rc2, 1.12.0, 1.13.0rc1, 1.13.0, 1.13.1, 1.14.0rc2, 1.14.0, 1.14.1,
1.15.0rc1, 1.15.0rc2, 1.15.0, 1.15.1, 1.15.2, 1.15.3, 1.16.0rc1, 1.16.0rc2, 1.16.
0, 1.16.1, 1.16.2, 1.16.3, 1.17.0rc1, 1.17.0rc2, 1.17.0)
ERROR: No matching distribution found for scipy==1.10
Requirement already satisfied: pandas in /opt/conda/lib/python3.12/site-packages
(2.3.3)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-pa
ckages (from pandas) (2.4.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.1
2/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-pac
kages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-p
ackages (from pandas) (2025.3)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-package
s (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: nltk in /opt/conda/lib/python3.12/site-packages
(3.6.7)
Requirement already satisfied: click in /opt/conda/lib/python3.12/site-packages
(from nltk) (8.3.1)
Requirement already satisfied: joblib in /opt/conda/lib/python3.12/site-packages
(from nltk) (1.5.3)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.12/site-
packages (from nltk) (2025.11.3)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.12/site-packages (f
rom nltk) (4.67.1)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.12/site-packa
ges (3.10.8)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site
-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/sit
```

```
e-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib) (1.4.9)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-pack
ages (from matplotlib) (2.4.1)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-
packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packag
es (from matplotlib) (12.1.0)
Requirement already satisfied: pyparsing>=3 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/
site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-package
s (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

In [3]:
```
import gensim
import pandas as pd
import nltk as nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from gensim import corpora

%matplotlib inline
```

Download stopwords

In [4]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /home/jupyterlab/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/jupyterlab/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jupyterlab/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

Out[4]: True

In [5]:
```
# also set a random state
rs = 123
```

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:

In [6]:
```
course1 = "this is an introduction data science course which introduces data sci
```

In [7]:
```
course2 = "machine learning for beginners"
```

In [8]:
```
courses = [course1, course2]
courses
```

Out[8]:
```
['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):

In [9]:
```
# Tokenize the two courses
tokenized_courses = [word_tokenize(course) for course in courses]
```

In [10]:
```
tokenized_courses
```

Out[10]:
```
[['this',
  'is',
  'an',
  'introduction',
  'data',
  'science',
  'course',
  'which',
  'introduces',
  'data',
  'science',
  'to',
  'beginners'],
 ['machine', 'learning', 'for', 'beginners']]
```

As you can see from the cell output, two courses have been tokenized and turned into two token arrays.

Next, we want to create a token dictionary to index all tokens. Basically, we want to assign a key/index for each token. One way to index tokens is to use the `gensim` package which is another popular package for processing textual data:

In [11]:
```
# Create a token dictionary for the two courses
tokens_dict = gensim.corpora.Dictionary(tokenized_courses)
```

In [12]:
```
print(tokens_dict.token2id)
```

```
{'an': 0, 'beginners': 1, 'course': 2, 'data': 3, 'introduces': 4, 'introductio
n': 5, 'is': 6, 'science': 7, 'this': 8, 'to': 9, 'which': 10, 'for': 11, 'learni
ng': 12, 'machine': 13}
```

With the token dictionary, we can easily count each token in the two example courses and output two BoW feature vectors. However, more conveniently, the `gensim` package provides us a `doc2bow` method to generate BoW features out-of-box.

In [13]:
```
# Generate BoW features for each course
courses_bow = [tokens_dict.doc2bow(course) for course in tokenized_courses]
```

In [14]:
```
courses_bow
```

Out[14]:
```
[[(0, 1),
  (1, 1),
  (2, 1),
  (3, 2),
  (4, 1),
  (5, 1),
  (6, 1),
  (7, 2),
  (8, 1),
  (9, 1),
  (10, 1)],
 [(1, 1), (11, 1), (12, 1), (13, 1)]]
```

It outputs two BoW arrays where each element is a tuple, e.g., (0, 1) and (7, 2). The first element of the tuple is the token ID and the second element is its count. So `(0, 1)` means `(``an``, 1)` and `(7, 2)` means `(``science``, 2)`.

We can use the following code snippet to print each token and its count:

In [15]:
```
# Enumerate through each course and its bag-of-words representation
for course_idx, course_bow in enumerate(courses_bow):
    # Print the index of the current course and a label
    print(f"Bag of words for course {course_idx}:")
    # For each token index, print its bow value (word count)
    for token_index, token_bow in course_bow:
        # Retrieve the token from the tokens dictionary based on its index
        token = tokens_dict.get(token_index)
        # Print the token and its bag-of-words value
        print(f"--Token: '{token}', Count:{token_bow}")
```

```
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	

Bag of Words feature vectors

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.

We can use the english stop words provided in `nltk`:

In [16]:
```
stop_words = set(stopwords.words('english'))
```

In [17]:
```
stop_words
```

Out[17]:
```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
 'been',
 'before',
 'being',
 'below',
 'between',
 'both',
 'but',
 'by',
 'can',
 'couldn',
 "couldn't",
 'd',
 'did',
 'didn',
 "didn't",
 'do',
 'does',
 'doesn',
 "doesn't",
 'doing',
 'don',
 "don't",
 'down',
 'during',
 'each',
 'few',
 'for',
 'from',
 'further',
 'had',
 'hadn',
 "hadn't",
 'has',
 'hasn',
 "hasn't",
 'have',
 'haven',
 "haven't",
 'having',
 'he',
 "he'd",
```

```
                "he'll",
                "he's",
                'her',
                'here',
                'hers',
                'herself',
                'him',
                'himself',
                'his',
                'how',
                'i',
                "i'd",
                "i'll",
                "i'm",
                "i've",
                'if',
                'in',
                'into',
                'is',
                'isn',
                "isn't",
                'it',
                "it'd",
                "it'll",
                "it's",
                'its',
                'itself',
                'just',
                'll',
                'm',
                'ma',
                'me',
                'mightn',
                "mightn't",
                'more',
                'most',
                'mustn',
                "mustn't",
                'my',
                'myself',
                'needn',
                "needn't",
                'no',
                'nor',
                'not',
                'now',
                'o',
                'of',
                'off',
                'on',
                'once',
                'only',
                'or',
                'other',
                'our',
                'ours',
                'ourselves',
                'out',
                'over',
                'own',
```

```
're',
's',
'same',
'shan',
"shan't",
'she',
"she'd",
"she'll",
"she's",
'should',
"should've",
'shouldn',
"shouldn't",
'so',
'some',
'such',
't',
'than',
'that',
"that'll",
'the',
'their',
'theirs',
'them',
'themselves',
'then',
'there',
'these',
'they',
"they'd",
"they'll",
"they're",
"they've",
'this',
'those',
'through',
'to',
'too',
'under',
'until',
'up',
've',
'very',
'was',
'wasn',
"wasn't",
'we',
"we'd",
"we'll",
"we're",
"we've",
'were',
'weren',
"weren't",
'what',
'when',
'where',
'which',
'while',
'who',
```

```
 'whom',
 'why',
 'will',
 'with',
 'won',
 "won't",
 'wouldn',
 "wouldn't",
 'y',
 'you',
 "you'd",
 "you'll",
 "you're",
 "you've",
 'your',
 'yours',
 'yourself',
 'yourselves'}
```

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

In [18]:
```
# Tokens in course 1
tokenized_courses[0]
```

Out[18]:
```
['this',
 'is',
 'an',
 'introduction',
 'data',
 'science',
 'course',
 'which',
 'introduces',
 'data',
 'science',
 'to',
 'beginners']
```

In [19]:
```
processed_tokens = [w for w in tokenized_courses[0] if not w.lower() in stop_wor
```

In [20]:
```
processed_tokens
```

Out[20]:
```
['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.

In [21]:
```
tags = nltk.pos_tag(tokenized_courses[0])
tags
```

Out[21]:
```
[('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.

TASK: Extract BoW features for course textual content and build a dataset

By now you have learned what a BoW feature is, so let's start extracting BoW features from some real course textual content.

In [22]:
```
course_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
course_content_df = pd.read_csv(course_url)
```

In [23]:
```
course_content_df.iloc[0, :]
```

Out[23]:
```
COURSE_ID                                                ML0201EN
TITLE          robots are coming  build iot apps with watson ...
DESCRIPTION    have fun with iot and learn along the way  if ...
Name: 0, dtype: object
```

The course content dataset has three columns `COURSE_ID` , `TITLE` , and `DESCRIPTION` . `TITLE` and `DESCRIPTION` are all text upon which we want to extract BoW features.

Let's join those two text columns together.

In [24]:
```
# Merge TITLE and DESCRIPTION title
course_content_df['course_texts'] = course_content_df[['TITLE', 'DESCRIPTION']].
course_content_df = course_content_df.reset_index()
course_content_df['index'] = course_content_df.index
```

In [25]:
```
course_content_df.iloc[0, :]
```

Out[25]:
```
index                                                        0
COURSE_ID                                                ML0201EN
TITLE          robots are coming  build iot apps with watson ...
DESCRIPTION    have fun with iot and learn along the way  if ...
course_texts   robots are coming  build iot apps with watson ...
Name: 0, dtype: object
```

We have used the `tokenize_course()` method to tokenize the course content:

In [26]:
```
def tokenize_course(course, keep_only_nouns=True):
    # Get English stop words
    stop_words = set(stopwords.words('english'))
    # Tokenize the course text
    word_tokens = word_tokenize(course)
    # Remove English stop words and numbers
    word_tokens = [w for w in word_tokens if (not w.lower() in stop_words) and (
    # Only keep nouns
    if keep_only_nouns:
        # Define a filter list of non-noun POS tags
        filter_list = ['WDT', 'WP', 'WRB', 'FW', 'IN', 'JJR', 'JJS', 'MD', 'PDT'
                       'RP']
        # Tag the word tokens with POS tags
        tags = nltk.pos_tag(word_tokens)
        # Filter out non-nouns based on POS tags
        word_tokens = [word for word, pos in tags if pos not in filter_list]

    return word_tokens
```

Let's try it on the first course.

In [27]:
```
a_course = course_content_df.iloc[0, :]['course_texts']
a_course
```

Out[27]:
```
'robots are coming  build iot apps with watson  swift  and node red have fun wi
th iot and learn along the way  if you re a swift developer and want to learn m
ore about iot and watson ai services in the cloud  raspberry pi  and node red
you ve found the right place  you ll build iot apps to read temperature data  t
ake pictures with a raspcam  use ai to recognize the objects in those pictures
and program an irobot create 2 robot  '
```

In [58]:
```
course_id = course_content_df.iloc[0, :]['COURSE_ID']
print(course_id)
```

```
ML0201EN
```

In [28]:
```
tokenize_course(a_course)
```

Out[28]:
```
['robots',
 'coming',
 'build',
 'iot',
 'apps',
 'watson',
 'swift',
 'red',
 'fun',
 'iot',
 'learn',
 'way',
 'swift',
 'developer',
 'want',
 'learn',
 'iot',
 'watson',
 'ai',
 'services',
 'cloud',
 'raspberry',
 'pi',
 'node',
 'red',
 'found',
 'place',
 'build',
 'iot',
 'apps',
 'read',
 'temperature',
 'data',
 'take',
 'pictures',
 'raspcam',
 'use',
 'ai',
 'recognize',
 'objects',
 'pictures',
 'program',
 'irobot',
 'create',
 'robot']
```

Next, you will need to write some code snippets to generate the BoW features for each course. Let's start by tokenzing all courses in the `courses_df`:

TODO: Use provided tokenize_course() method to tokenize all courses in courses_df['course_texts'].

In [44]:
```
# WRITE YOUR CODE HERE
tokens_dict= tokenize_course(a_course, True)
arr = []
arr.append(tokens_dict)
print(arr)
```

```
['robots', 'coming', 'build', 'iot', 'apps', 'watson', 'swift', 'red', 'fun', 'io
t', 'learn', 'way', 'swift', 'developer', 'want', 'learn', 'iot', 'watson', 'ai',
'services', 'cloud', 'raspberry', 'pi', 'node', 'red', 'found', 'place', 'build',
'iot', 'apps', 'read', 'temperature', 'data', 'take', 'pictures', 'raspcam', 'us
e', 'ai', 'recognize', 'objects', 'pictures', 'program', 'irobot', 'create', 'rob
ot']
[['robots', 'coming', 'build', 'iot', 'apps', 'watson', 'swift', 'red', 'fun', 'i
ot', 'learn', 'way', 'swift', 'developer', 'want', 'learn', 'iot', 'watson', 'a
i', 'services', 'cloud', 'raspberry', 'pi', 'node', 'red', 'found', 'place', 'bui
ld', 'iot', 'apps', 'read', 'temperature', 'data', 'take', 'pictures', 'raspcam',
'use', 'ai', 'recognize', 'objects', 'pictures', 'program', 'irobot', 'create',
'robot']]
```

▶ Click here for Hints

Then we need to create a token dictionary `tokens_dict`

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

In [46]:
```
# WRITE YOUR CODE HERE
tokens_dict = gensim.corpora.Dictionary(arr)
print(tokens_dict.token2id)
tokenized_words = 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, 'nod
e': 13, 'objects': 14, 'pi': 15, 'pictures': 16, 'place': 17, 'program': 18, 'ras
pberry': 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}
```

Then we can use `doc2bow()` method to generate BoW features for each tokenized course.

TODO: Use tokens_dict.doc2bow() to generate BoW features for each tokenized course.

In [50]:
```
# WRITE YOUR CODE HERE
courses_bows = [tokens_dict.doc2bow(course) for course in arr]
print(courses_bows)
```

```
[[(0, 2), (1, 2), (2, 2), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1),
(10, 4), (11, 1), (12, 2), (13, 1), (14, 1), (15, 1), (16, 2), (17, 1), (18, 1),
(19, 1), (20, 1), (21, 1), (22, 1), (23, 2), (24, 1), (25, 1), (26, 1), (27, 2),
(28, 1), (29, 1), (30, 1), (31, 1), (32, 2), (33, 1)]]
```

▶ Click here for Hints

Lastly, you need to append the BoW features for each course into a new BoW dataframe. The 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

TODO: Create a new course_bow dataframe based on the extracted BoW features.

In [51]:
```
# WRITE YOUR CODE HERE

#  ...
#  bow_dicts = {"doc_index": doc_indices,
#               "doc_id": doc_ids,
#               "token": tokens,
#               "bow": bow_values}
#  pd.DataFrame(bow_dicts)

for doc_index,doc_bow in enumerate(courses_bows):
    print(f"Bag of words for course {doc_bow}:")
    for token_index,token_bow in doc_bow:
        token = tokens_dict.get(token_index)
        # Print the token and its bag-of-words value
        print(f"--Token: '{token}', Count:{token_bow}")
```

```
Bag of words for course [(0, 2), (1, 2), (2, 2), (3, 1), (4, 1), (5, 1), (6, 1),
(7, 1), (8, 1), (9, 1), (10, 4), (11, 1), (12, 2), (13, 1), (14, 1), (15, 1), (1
6, 2), (17, 1), (18, 1), (19, 1), (20, 1), (21, 1), (22, 1), (23, 2), (24, 1), (2
5, 1), (26, 1), (27, 2), (28, 1), (29, 1), (30, 1), (31, 1), (32, 2), (33, 1)]:
--Token: 'ai', Count:2
--Token: 'apps', Count:2
--Token: 'build', Count:2
--Token: 'cloud', Count:1
--Token: 'coming', Count:1
--Token: 'create', Count:1
--Token: 'data', Count:1
--Token: 'developer', Count:1
--Token: 'found', Count:1
--Token: 'fun', Count:1
--Token: 'iot', Count:4
--Token: 'irobot', Count:1
--Token: 'learn', Count:2
--Token: 'node', Count:1
--Token: 'objects', Count:1
--Token: 'pi', Count:1
--Token: 'pictures', Count:2
--Token: 'place', Count:1
--Token: 'program', Count:1
--Token: 'raspberry', Count:1
--Token: 'raspcam', Count:1
--Token: 'read', Count:1
--Token: 'recognize', Count:1
--Token: 'red', Count:2
--Token: 'robot', Count:1
--Token: 'robots', Count:1
--Token: 'services', Count:1
--Token: 'swift', Count:2
--Token: 'take', Count:1
--Token: 'temperature', Count:1
--Token: 'use', Count:1
--Token: 'want', Count:1
--Token: 'watson', Count:2
--Token: 'way', Count:1
```

In [60]:
```
bow_dicts = {"doc_index": doc_index,
             "doc_id": doc_bow,
             "token": token,
```

```
            "bow": token_bow}
pd.DataFrame(bow_dicts)
```

Out[60]:

	doc_index	doc_id	token	bow
0	0	(0, 2)	way	1
1	0	(1, 2)	way	1
2	0	(2, 2)	way	1
3	0	(3, 1)	way	1
4	0	(4, 1)	way	1
5	0	(5, 1)	way	1
6	0	(6, 1)	way	1
7	0	(7, 1)	way	1
8	0	(8, 1)	way	1
9	0	(9, 1)	way	1
10	0	(10, 4)	way	1
11	0	(11, 1)	way	1
12	0	(12, 2)	way	1
13	0	(13, 1)	way	1
14	0	(14, 1)	way	1
15	0	(15, 1)	way	1
16	0	(16, 2)	way	1
17	0	(17, 1)	way	1
18	0	(18, 1)	way	1
19	0	(19, 1)	way	1
20	0	(20, 1)	way	1
21	0	(21, 1)	way	1
22	0	(22, 1)	way	1
23	0	(23, 2)	way	1
24	0	(24, 1)	way	1
25	0	(25, 1)	way	1
26	0	(26, 1)	way	1
27	0	(27, 2)	way	1
28	0	(28, 1)	way	1
29	0	(29, 1)	way	1
30	0	(30, 1)	way	1
31	0	(31, 1)	way	1
32	0	(32, 2)	way	1

	doc_index	doc_id	token	bow
33	0	(33, 1)	way	1

▶ Click here for Hints

Your course BoW dataframe may look like the following:

You may refer to previous code examples in this lab if you need help with creating the BoW dataframe.

Other popular textual features

In addition to the basic token BoW feature, there are two other types of widely used textual features. If you are interested, you may explore them yourself to learn how to extract them from the course textual content:

- **tf-idf**: tf-idf refers to Term Frequency–Inverse Document Frequency. Similar to BoW, the tf-idf also counts the word frequencies in each document. Furthermore, tf-idf will offset the number of documents in the corpus that contain the word in order to adjust for the fact that some words appear more frequently in general. The higher the tf-idf normally means the greater the importance the word/token is.
- **Text embedding vector**. Embedding means projecting an object into a latent feature space. We normally employ neural networks or deep neural networks to learn the latent features of a textual object such as a word, a sentence, or the entire document. The learned latent feature vectors will be used to represent the original textual entities.

Summary

Congratulations, you have completed the BoW feature extraction lab. In this lab, you have learned and practiced extracting BoW features from course titles and descriptions. Once the feature vectors on the courses has been built, we can then apply machine learning algorithms such as similarity measurements, clustering, or classification on the courses in later labs.

Authors

Yan Luo

Other Contributors

```
toggle##
```

```
toggle|Date

toggle|-|-|-|-|

toggle|2021-10-25|1.0|Yan|Created
```