**New Package Introduction - Recommendation Systems Part 1**

Let’s now go through some of the common functions used in the LVC case studies.

**Defaultdict**

One of the issues that we face with Python dictionaries is the frequent missing KeyError. In those cases where dictionaries are frequently used, it can be a hindrance to deal with this error. To overcome this, there is a Python library called **collections**that implements specialized container types. One such example is the Python **defaultdict** type, which is an alternative to dict that’s specifically designed to help with missing keys.

To import it the below code can be used.

from collections import defaultdict

To get more information on this function you can refer to [this link](https://docs.python.org/3/library/collections.html#collections.defaultdict).

**Surprise**

Surprise is an easy-to-use Python library that allows us to quickly build different kinds of recommender systems without having to write code from scratch. It's a single library that contains all the methods and functionalities to build many types of recommendation systems in Python. It has methods to build Collaborative Filtering-Based, Clustering-Based, and Model-Based recommendation systems. The following code is needed to install the package.

# Installing the surprise library  
!pip install surprise

To get more information about this package you can refer to [this link](https://surprise.readthedocs.io/en/stable/getting_started.html).

**Reader**

The **Reader**function in the Surprise library creates a different class to prepare the required format of the dataset to build recommendation systems. To import it, the below code can be used:

from surprise.reader import Reader

It is used in the below way to format a dataset.

# Instantiating Reader scale with expected rating scale  
reader = Reader(rating\_scale = (0, 5))  
  
# Loading the rating dataset  
data = Dataset.load\_from\_df(rating[['userId', 'movieId', 'rating']], reader)  
  
# Splitting the data into train and test datasets  
trainset, testset = train\_test\_split(data, test\_size = 0.2, random\_state = 42)

Here "Dataset" is used to format the pandas DataFrame into what's required in Surprise. The train\_test\_split is used to split the dataset into train and test sets.

To get more information about this function you can refer to [this link](https://surprise.readthedocs.io/en/stable/reader.html).

**KNNBasic**

KNNBasic is an algorithm associated with the Surprise package. It is used to find the desired similar items among a given set of items.

The following code helps in importing this function.

from surprise.prediction\_algorithms.knns import KNNBasic

The below code demonstrates one specific use of this function.

sim\_user\_user = KNNBasic(sim\_options = sim\_options, verbose = False, random\_state = 1)

Here, sim\_options contains the similarity options that need to be considered when measuring the similarity between two users or items. It contains the type of similarity measure we want to use, which may be a cosine similarity or some distance-based similarity like Manhattan distance or Euclidean distance.

To get more information about this function, you can refer to [this link](https://surprise.readthedocs.io/en/stable/knn_inspired.html).

**GridSearchCV**

This is a special method in the Surprise library that is used to perform hyperparameter tuning in order to find the best set of hyperparameters. To import this, the below code can be used:

from surprise.model\_selection import GridSearchCV

The following code shows one use case of this function.

gs = GridSearchCV(KNNBasic, param\_grid, measures = ['rmse'], cv = 3)

Here, the param\_grid is the set of values for each hyperparameter that needs to be optimized. The measures display the type of error that we consider to find the best hyperparameter set, cv tells us about the number of cross validations utilized.

To get more information about this function, you can refer to [this link](https://surprise.readthedocs.io/en/stable/model_selection.html).

**SVD**

It isa function used to perform singular value decomposition over a matrix. It provides methods to create matrix factorization based Recommendation Systems. To import it, the below code can be used:

from surprise.prediction\_algorithms.matrix\_factorization import SVD

The following code shows one of the applications of this function.

# Using SVD with matrix factorization  
svd = SVD(random\_state = 1)  
# Training the algorithm on the training dataset  
svd.fit(trainset)  
# Let us compute precision@k, recall@k, and f\_1 score with k = 10  
precision\_recall\_at\_k(svd)

To get more information about this function, you can refer to [this link](https://surprise.readthedocs.io/en/stable/matrix_factorization.html).

**New Package Introduction - Recommendation Systems Part 2**

Now, let’s go through some of the common functions used in this LVC case studies.

#### ****CoClustering****

Clustering based recommendation system is one important way to build recommendation system using the concepts of clustering. **CoClustering** is a method in the Surprise library that is used to build the clustering based recommendation system in Python. To import this, the below code can be used:

from surprise import CoClustering

One sample application of this function is shown below:

# Using CoClustering algorithm  
clust\_baseline = CoClustering(random\_state = 1)  
  
# Training the algorithm on the train set  
clust\_baseline.fit(trainset)  
  
# Let us compute precision@k, recall@k, and F\_1 score with k = 10  
precision\_recall\_at\_k(clust\_baseline)

To get more information about this function, you can refer to [this](https://surprise.readthedocs.io/en/stable/co_clustering.html) link.

#### ****nltk****

**Content based recommendation system** is a very important way to build recommendation system. It uses the features of the items to find similar items to recommend. The concepts of natural language processing can be used to prepare relevant data if the item data is in text format. **nltk (Natural Language Toolkit)** is a library that is used to do text preprocessing and prepare the numerical vectors required for building the recommendation system. To import this library, the below code can be used:

import nltk

Some of the helpful methods in nltk that is used for text preprocessing is as follows:

#### ****Word\_tokenize****

from nltk import word\_tokenize

This is used to perform tokenization of a text document.

#### ****Stopwords****

from nltk.corpus import stopwords

This is used to remove stopwords of a certain language from the given text.

To get more information about nltk, you can refer to [this](https://www.nltk.org/) link.

#### ****TfidfVectorizer****

One of the basic requirement while working with text data is to convert it into a numeric vector. The TfidfVectorizer is a way to create such vectors. It considers both the term frequency and the inverse document frequency to create this vector. Term frequency is the frequency of a word in a document while the inverse document frequency depends on the number of documents in which the considered word appears. The below code is used to import this method:

from sklearn.feature\_extraction.text import TfidfVectorizer

The below code shows one sample application of the same:

# Creating the TF-IDF object  
tfidf = TfidfVectorizer(tokenizer = tokenize)  
  
movie\_tfidf = tfidf.fit\_transform(final\_ratings['text'].values).toarray()

To get more information about this function, you can refer to [this](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) link.

#### ****Cosine\_similarity****

This is a method used to find the cosine similarity of two numeric vectors in python. This is a method from the sklearn library in python. To import it, the below code can be used:

from sklearn.metrics.pairwise import cosine\_similarity

To get more information on this function, you can refer to [this](https://surprise.readthedocs.io/en/stable/similarities.html) link.

Happy learning!

**New Package Introduction - Recommendation Systems Part 3**

Now, let’s go through some of the common functions used in this LVC case studies.

#### ****mSSA****

mSSA is a library used to apply **Multivariate Singular Spectrum Analysis (mSSA)** over a time series data in Python.

To import the library, the below line of code can be used:

# Installing the mSSA library  
from mssa.mssa import mSSA

The below code demonstrates the application of this library.

# Let us define the model variable  
model = mSSA(rank = 20)  
  
# Updating the model  
model.update\_model(train\_data)

To get more information about this function, you can refer to [this](https://github.com/AbdullahO/mSSA) link.

#### ****Seaborn****

This is an advanced library to perform data visualization in python. It is based on the matplotlib library and it provides a high-level interface for drawing attractive and informative statistical graphics. To import this, the below line of code can be used:

import seaborn as sns

The sample application of this library can be seen in the below line of code:

# Create the plot   
figure = plt.figure(figsize = (16, 8))  
sns.lineplot(data = combined\_data,  
 x = 'index',  
 y = 'MT\_001',  
 marker = 'o');

Here, this library is used to create the lineplot using some sample data.

To get more information about this function, you can refer to [this](https://seaborn.pydata.org/) link.

Happy learning!



