ML0101EN-RecSys-Content-Based-movies-py-v1

July 12, 2022

1 Content Based Filtering

Estimated time needed: **25** minutes

1.1 Objectives

After completing this lab you will be able to:

- Create a recommendation system using Content Based filtering

Recommendation systems are a collection of algorithms used to recommend items to users based on information taken from the user. These systems have become ubiquitous, and can be commonly seen in online stores, movies databases and job finders. In this notebook, we will explore Content-based recommendation systems and implement a simple version of one using Python and the Pandas library.

1.1.1 Table of contents

```
<ol>
    <li><a href="https://#ref1">Acquiring the Data</a></li>
    <li><a href="https://#ref2">Preprocessing</a></li>
    <li><a href="https://#ref3">Content-Based Filtering</a></li>
</ol>
```

2 Acquiring the Data

To acquire and extract the data, simply run the following Bash scripts:
Dataset acquired from GroupLens. Let's download the dataset. To download the data, we will use **!wget** to download it from IBM Object Storage.
Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: Sign up now for free

[ ]:
```
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.
 ↪appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/
 ↪Module%205/data/moviedataset.zip
print('unziping ...')
!unzip -o -j moviedataset.zip
```

Now you're ready to start working with the data!

3 Preprocessing

First, let's get all of the imports out of the way:

[3]:
```
#Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Now let's read each file into their Dataframes:

[4]:
```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('ratings.csv')
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipykernel_91/2524670424.py in <module>
      1 #Storing the movie information into a pandas dataframe
----> 2 movies_df = pd.read_csv('movies.csv')
      3 #Storing the user information into a pandas dataframe
      4 ratings_df = pd.read_csv('ratings.csv')
      5 #Head is a function that gets the first N rows of a dataframe. N's␣
 ↪default is 5.

~/conda/envs/python/lib/python3.7/site-packages/pandas/util/_decorators.py in␣
 ↪wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/parsers/readers.py in␣
 ↪read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
 ↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
 ↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
 ↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates␣
 ↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
 ↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
 ↪quotechar, quoting, doublequote, escapechar, comment, encoding,␣
 ↪encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines,␣
 ↪delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    584     kwds.update(kwds_defaults)
    585
```

```
--> 586     return _read(filepath_or_buffer, kwds)
    587
    588

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/parsers/readers.py in
 ↪_read(filepath_or_buffer, kwds)
    480
    481     # Create the parser.
--> 482     parser = TextFileReader(filepath_or_buffer, **kwds)
    483
    484     if chunksize or iterator:

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/parsers/readers.py in
 ↪__init__(self, f, engine, **kwds)
    809             self.options["has_index_names"] = kwds["has_index_names"]
    810
--> 811         self._engine = self._make_engine(self.engine)
    812
    813     def close(self):

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/parsers/readers.py in
 ↪_make_engine(self, engine)
   1038             )
   1039         # error: Too many arguments for "ParserBase"
-> 1040         return mapping[engine](self.f, **self.options)  # type:␣
 ↪ignore[call-arg]
   1041
   1042     def _failover_to_python(self):

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/parsers/
 ↪c_parser_wrapper.py in __init__(self, src, **kwds)
     49
     50         # open handles
---> 51         self._open_handles(src, kwds)
     52         assert self.handles is not None
     53

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/parsers/base_parser.p␣
 ↪in _open_handles(self, src, kwds)
    227             memory_map=kwds.get("memory_map", False),
    228             storage_options=kwds.get("storage_options", None),
--> 229             errors=kwds.get("encoding_errors", "strict"),
    230         )
    231

~/conda/envs/python/lib/python3.7/site-packages/pandas/io/common.py in␣
 ↪get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,␣
 ↪errors, storage_options)
```

```
    705                 encoding=ioargs.encoding,
    706                 errors=errors,
--> 707                 newline="",
    708             )
    709         else:

FileNotFoundError: [Errno 2] No such file or directory: 'movies.csv'
```

Let's also remove the year from the **title** column by using pandas' replace function and store in a new **year** column.

[ ]:
```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in␣
 ↪their titles
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters␣
 ↪that may have appeared
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

With that, let's also split the values in the **Genres** column into a **list of Genres** to simplify for future use. This can be achieved by applying Python's split string function on the correct column.

[ ]:
```
#Every genre is separated by a | so we simply have to call the split function␣
 ↪on |
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

Since keeping genres in a list format isn't optimal for the content-based recommendation system technique, we will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature. This encoding is needed for feeding categorical data. In this case, we store every different genre in columns that contain either 1 or 0. 1 shows that a movie has that genre and 0 shows that it doesn't. Let's also store this dataframe in another variable since genres won't be important for our first recommendation system.

[ ]:
```
#Copying the movie dataframe into a new one since we won't need to use the␣
 ↪genre information in our first case.
moviesWithGenres_df = movies_df.copy()

#For every row in the dataframe, iterate through the list of genres and place a␣
 ↪1 into the corresponding column
for index, row in movies_df.iterrows():
    for genre in row['genres']:
```

```
        moviesWithGenres_df.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that␣
 ↪column's genre
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

Next, let's look at the ratings dataframe.

[ ]:
```
ratings_df.head()
```

Every row in the ratings dataframe has a user id associated with at least one movie, a rating and a timestamp showing when they reviewed it. We won't be needing the timestamp column, so let's drop it to save memory.

[ ]:
```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

4 Content-Based recommendation system

Now, let's take a look at how to implement **Content-Based** or **Item-Item recommendation systems**. This technique attempts to figure out what a user's favourite aspects of an item is, and then recommends items that present those aspects. In our case, we're going to try to figure out the input's favorite genres from the movies and ratings given.

Let's begin by creating an input user to recommend movies to:

Notice: To add more movies, simply increase the amount of elements in the **userInput**. Feel free to add more in! Just be sure to write it in with capital letters and if a movie starts with a "The", like "The Matrix" then write it in like this: 'Matrix, The' .

[ ]:
```
userInput = [
            {'title':'Breakfast Club, The', 'rating':5},
            {'title':'Toy Story', 'rating':3.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Add movieId to input user With the input complete, let's extract the input movie's ID's from the movies dataframe and add them into it.

We can achieve this by first filtering out the rows that contain the input movie's title and then merging this subset with the input dataframe. We also drop unnecessary columns for the input to save memory space.

[ ]:
```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

We're going to start by learning the input's preferences, so let's get the subset of movies that the input has watched from the Dataframe containing genres defined with binary values.

[ ]:
```
#Filtering out the movies from the input
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].
 ↪isin(inputMovies['movieId'].tolist())]
userMovies
```

We'll only need the actual genre table, so let's clean this up a bit by resetting the index and dropping the movieId, title, genres and year columns.

[ ]:
```
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres',␣
 ↪1).drop('year', 1)
userGenreTable
```

Now we're ready to start learning the input's preferences!

To do this, we're going to turn each genre into weights. We can do this by using the input's reviews and multiplying them into the input's genre table and then summing up the resulting table by column. This operation is actually a dot product between a matrix and a vector, so we can simply accomplish by calling the Pandas "dot" function.

[ ]:
```
inputMovies['rating']
```

[ ]:
```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

Now, we have the weights for every of the user's preferences. This is known as the User Profile. Using this, we can recommend movies that satisfy the user's preferences.

Let's start by extracting the genre table from the original dataframe:

[ ]:
```
#Now let's get the genres of every movie in our original dataframe
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).
 ↪drop('year', 1)
genreTable.head()
```

[ ]:
```
genreTable.shape
```

With the input's profile and the complete list of movies and their genres in hand, we're going to take the weighted average of every movie based on the input profile and recommend the top twenty movies that most satisfy it.

[ ]:
```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.
 ↪sum())
recommendationTable_df.head()
```

[ ]:
```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

Now here's the recommendation table!

[ ]:
```
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

4.0.1 Advantages and Disadvantages of Content-Based Filtering

Advantages

- Learns user's preferences
- Highly personalized for the user

Disadvantages

- Doesn't take into account what others think of the item, so low quality item recommendations might happen
- Extracting data is not always intuitive
- Determining what characteristics of the item the user dislikes or likes is not always obvious

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

4.0.2 Thank you for completing this lab!

4.1 Author

Saeed Aghabozorgi

4.1.1 Other Contributors

Joseph Santarcangelo

4.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-11-03	2.1	Lakshmi	Updated URL of csv
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

##