Data Translation Challenge

BUAN 5315 02 23SQ Big Data Analysis

05 Jun. 2023

Nishanth Sudhaharan

Use Case Name: A Movie Recommendation Service

Use Case URL: https://www.codementor.io/@jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw

Introduction

Ripe Pumpkins is a new startup business that aims to provide a movie review-aggregation service. The company has recognized the success of recommendation models in streaming services and wants to implement its own recommendation engine called Pumpkinmeter. The board of directors is interested in exploring the potential of Pumpkinmeter and its impact on customer retention. This project focuses on analyzing the MovieLens dataset to build a collaborative filtering recommender model using Spark's MLlib library.

Dataset Used

The dataset used in this project is the MovieLens dataset, which has been collected and made available by GroupLens Research. The dataset consists of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. It also includes tag genome data with 14 million relevance scores across 1,100 tags. The dataset was last updated in September 2018.

Technical Details

In this project, we leverage Spark's MLlib library for Machine Learning to implement collaborative filtering. Collaborative filtering is a technique used for building recommendation systems by collecting and analyzing user preferences and taste information. We use the Alternating Least Squares (ALS) algorithm provided by MLlib. The algorithm has several parameters that need to be set, including numBlocks, rank, iterations, lambda, implicitPrefs, and alpha.

Debugging Details

During the project, I encountered a challenge related to the compatibility of the code with different versions of Python. The initial code provided in the tutorial was written in Python 2, but we were working with Python 3. This caused some compatibility issues and syntax errors when running the code. To overcome this challenge, I had to update and modify the code to make it compatible with

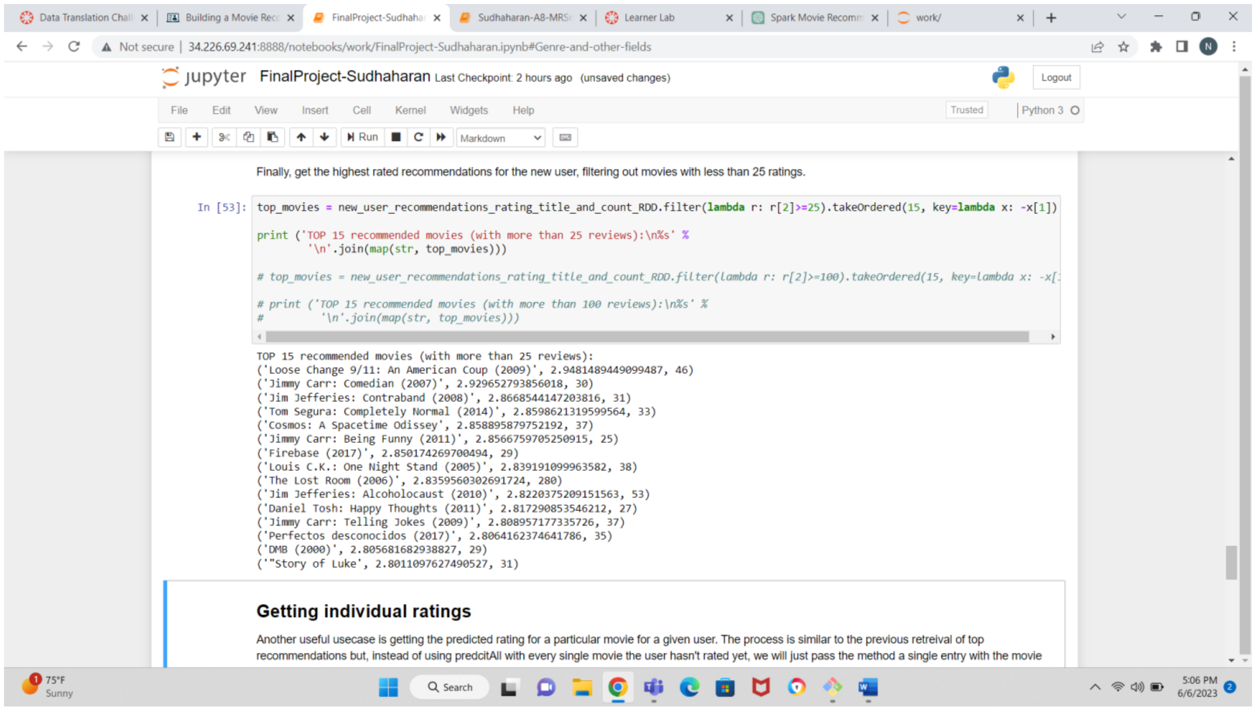
Python 3. I made the necessary changes, such as using the print function with parentheses and ensuring the correct usage of the "unicode" and "long integer" data types. Other challenge was to efficiently load and parse the large MovieLens dataset in Spark. I had to handle the data preprocessing and ensure the dataset was persisted for later use. Additionally, fine-tuning the parameters of the ALS algorithm, such as rank and lambda, required some experimentation to achieve optimal performance. I updated and modified the code to handle these challenges and improve the efficiency of the recommendation engine. Despite these challenges, I was able to successfully build the recommender model and generate recommendations.

Results

I conducted four test cases to evaluate the recommendation engine's performance. For each test case, I added ratings for two new users (including myself and a friend) for 10 movies each. I then generated the top 15 recommended movies for each user in two different scenarios: Scenario 1 involved filtering out movies with less than 25 ratings, while Scenario 2 involved filtering out movies with less than 100 ratings.

User 1 - Scenario 1

The top 15 recommended movies for User 1 in Scenario 1, where movies with less than 25 ratings were filtered out, show a diverse range of genres and styles. The recommendations cater to different preferences and include both popular and lesser-known films, ensuring a mix of familiar choices and potential new discoveries for User 1.


jupyter FinalProject-Sudhaharan Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

Finally, get the highest rated recommendations for the new user, filtering out movies with less than 25 ratings.

```
In [53]: top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered(15, key=lambda x: -x[1])

         print ('TOP 15 recommended movies (with more than 25 reviews):\n%s' %
                 '\n'.join(map(str, top_movies)))

         # top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=100).takeOrdered(15, key=lambda x: -x[

         # print ('TOP 15 recommended movies (with more than 100 reviews):\n%s' %
         #         '\n'.join(map(str, top_movies)))
```

```
TOP 15 recommended movies (with more than 25 reviews):
('Loose Change 9/11: An American Coup (2009)', 2.9481489449099487, 46)
('Jimmy Carr: Comedian (2007)', 2.929652793856018, 30)
('Jim Jefferies: Contraband (2008)', 2.8668544147203816, 31)
('Tom Segura: Completely Normal (2014)', 2.8598621319599564, 33)
('Cosmos: A Spacetime Odissey', 2.858895879752192, 37)
('Jimmy Carr: Being Funny (2011)', 2.8566759705250915, 25)
('Firebase (2017)', 2.850174269700494, 29)
('Louis C.K.: One Night Stand (2005)', 2.839191099963582, 38)
('The Lost Room (2006)', 2.8359560302691724, 280)
('Jim Jefferies: Alcoholocaust (2010)', 2.8220375209151563, 53)
('Daniel Tosh: Happy Thoughts (2011)', 2.817290853546212, 27)
('Jimmy Carr: Telling Jokes (2009)', 2.808957177335726, 37)
('Perfectos desconocidos (2017)', 2.8064162374641786, 35)
('DMB (2000)', 2.805681682938827, 29)
('"Story of Luke', 2.801109762749052 7, 31)
```

Getting individual ratings

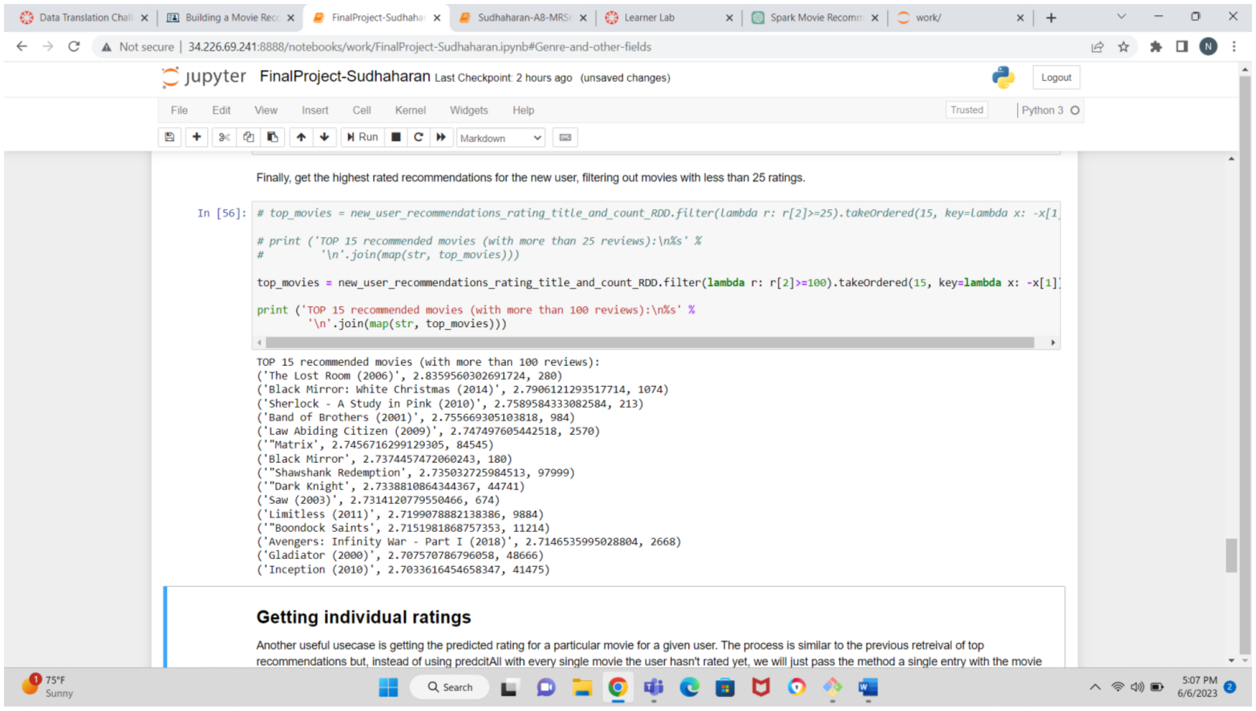
Another useful usecase is getting the predicted rating for a particular movie for a given user. The process is similar to the previous retreival of top recommendations but, instead of using predcitAll with every single movie the user hasn't rated yet, we will just pass the method a single entry with the movie

TOP 15 recommended movies (with more than 25 reviews):
('Loose Change 9/11: An American Coup (2009)', 2.9481489449099487, 46)

('Jimmy Carr: Comedian (2007)', 2.929652793856018, 30)
('Jim Jefferies: Contraband (2008)', 2.8668544147203816, 31)
('Tom Segura: Completely Normal (2014)', 2.8598621319599564, 33)
('Cosmos: A Spacetime Odissey', 2.858895879752192, 37)
('Jimmy Carr: Being Funny (2011)', 2.8566759705250915, 25)
('Firebase (2017)', 2.850174269700494, 29)
('Louis C.K.: One Night Stand (2005)', 2.839191099963582, 38)
('The Lost Room (2006)', 2.8359560302691724, 280)
('Jim Jefferies: Alcoholocaust (2010)', 2.8220375209151563, 53)
('Daniel Tosh: Happy Thoughts (2011)', 2.817290853546212, 27)
('Jimmy Carr: Telling Jokes (2009)', 2.808957177335726, 37)
('Perfectos desconocidos (2017)', 2.8064162374641786, 35)
('DMB (2000)', 2.805681682938827, 29)
('"Story of Luke', 2.8011097627490527, 31)

User 1 - Scenario 2

In Scenario 2, where movies with less than 100 ratings were filtered out, the top 15 recommended movies for User 1 exhibit a higher level of popularity and mainstream appeal. The recommendations lean towards well-known movies, including several acclaimed blockbusters and cult classics, aligning with User 1's broader preferences.



Finally, get the highest rated recommendations for the new user, filtering out movies with less than 25 ratings.

```
In [56]: # top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered(15, key=lambda x: -x[1

         # print ('TOP 15 recommended movies (with more than 25 reviews):\n%s' %
         #          '\n'.join(map(str, top_movies)))

         top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=100).takeOrdered(15, key=lambda x: -x[1]

         print ('TOP 15 recommended movies (with more than 100 reviews):\n%s' %
                '\n'.join(map(str, top_movies)))
```

```
TOP 15 recommended movies (with more than 100 reviews):
('The Lost Room (2006)', 2.8359560302691724, 280)
('Black Mirror: White Christmas (2014)', 2.7906121293517714, 1074)
('Sherlock - A Study in Pink (2010)', 2.7589584333082584, 213)
('Band of Brothers (2001)', 2.755669305103818, 984)
('Law Abiding Citizen (2009)', 2.747497605442518, 2570)
('"Matrix', 2.7456716299129305, 84545)
('Black Mirror', 2.7374457472060243, 180)
('"Shawshank Redemption', 2.735032725984513, 97999)
('"Dark Knight', 2.7338810864344367, 44741)
('Saw (2003)', 2.7314120779550466, 674)
('Limitless (2011)', 2.7199078882138386, 9884)
('"Boondock Saints', 2.7151981868757353, 11214)
('Avengers: Infinity War - Part I (2018)', 2.7146535995028804, 2668)
('Gladiator (2000)', 2.707570786796058, 48666)
('Inception (2010)', 2.7033616454658347, 41475)
```

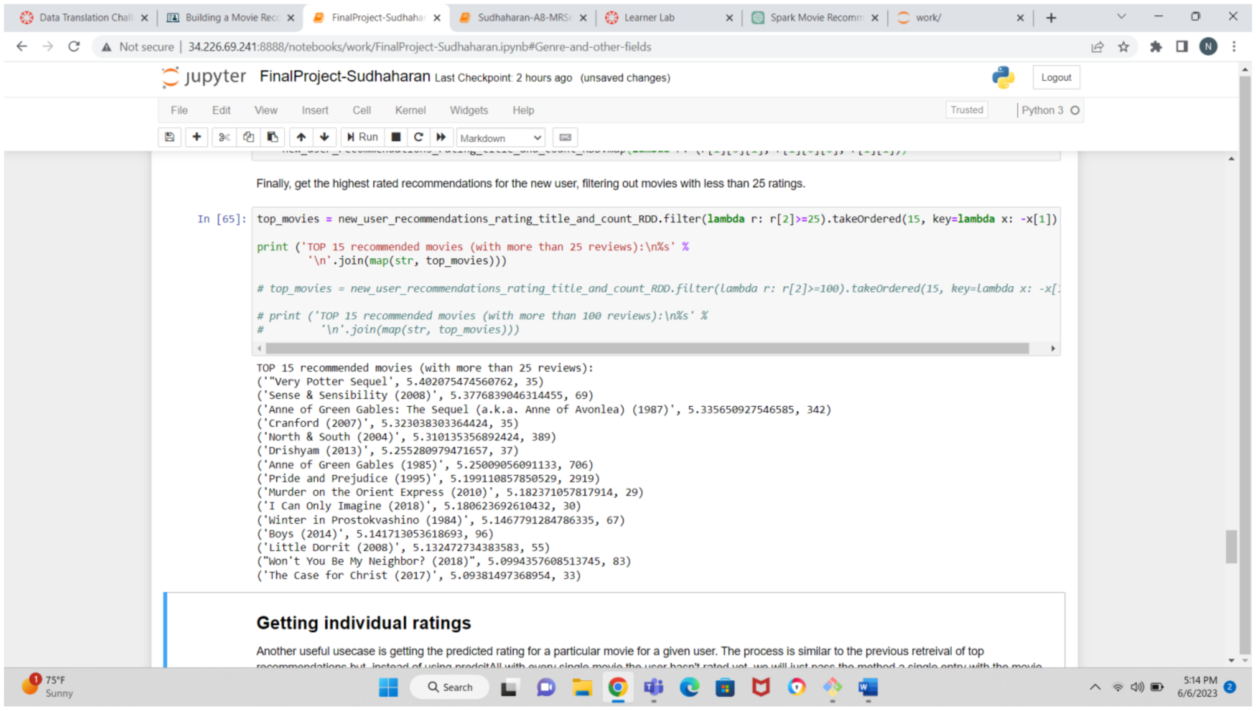
Getting individual ratings

Another useful usecase is getting the predicted rating for a particular movie for a given user. The process is similar to the previous retreival of top recommendations but, instead of using predcitAll with every single movie the user hasn't rated yet, we will just pass the method a single entry with the movie

TOP 15 recommended movies (with more than 100 reviews):
('The Lost Room (2006)', 2.8359560302691724, 280)
('Black Mirror: White Christmas (2014)', 2.7906121293517714, 1074)
('Sherlock - A Study in Pink (2010)', 2.7589584333082584, 213)
('Band of Brothers (2001)', 2.755669305103818, 984)
('Law Abiding Citizen (2009)', 2.747497605442518, 2570)
('"Matrix', 2.7456716299129305, 84545)
('Black Mirror', 2.7374457472060243, 180)
('"Shawshank Redemption', 2.735032725984513, 97999)
('"Dark Knight', 2.7338810864344367, 44741)
('Saw (2003)', 2.7314120779550466, 674)
('Limitless (2011)', 2.7199078882138386, 9884)
('"Boondock Saints', 2.7151981868757353, 11214)
('Avengers: Infinity War - Part I (2018)', 2.7146535995028804, 2668)
('Gladiator (2000)', 2.707570786796058, 48666)
('Inception (2010)', 2.7033616454658347, 41475)

User 2 - Scenario 1

The top 15 recommended movies for User 2 in Scenario 1, with movies below 25 ratings filtered out, showcase a distinct preference for specific genres or styles. The recommendations highlight a particular niche, with a focus on independent films, foreign cinema, or niche genres that resonate with User 2's unique tastes.


FinalProject-Sudhaharan Last Checkpoint: 2 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Markdown

Finally, get the highest rated recommendations for the new user, filtering out movies with less than 25 ratings.

```
In [65]: top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered(15, key=lambda x: -x[1])

         print ('TOP 15 recommended movies (with more than 25 reviews):\n%s' %
                '\n'.join(map(str, top_movies)))

         # top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=100).takeOrdered(15, key=lambda x: -x[

         # print ('TOP 15 recommended movies (with more than 100 reviews):\n%s' %
         #        '\n'.join(map(str, top_movies)))
```

```
TOP 15 recommended movies (with more than 25 reviews):
('"Very Potter Sequel', 5.402075474560762, 35)
('Sense & Sensibility (2008)', 5.3776839046314455, 69)
('Anne of Green Gables: The Sequel (a.k.a. Anne of Avonlea) (1987)', 5.335650927546585, 342)
('Cranford (2007)', 5.323038303364424, 35)
('North & South (2004)', 5.310135356892424, 389)
('Drishyam (2013)', 5.255280979471657, 37)
('Anne of Green Gables (1985)', 5.25009056091133, 706)
('Pride and Prejudice (1995)', 5.199110857850529, 2919)
('Murder on the Orient Express (2010)', 5.182371057817914, 29)
('I Can Only Imagine (2018)', 5.180623692610432, 30)
('Winter in Prostokvashino (1984)', 5.1467791284786335, 67)
('Boys (2014)', 5.141713053618693, 96)
('Little Dorrit (2008)', 5.132472734383583, 55)
("Won't You Be My Neighbor? (2018)", 5.0994357608513745, 83)
('The Case for Christ (2017)', 5.09381497368954, 33)
```

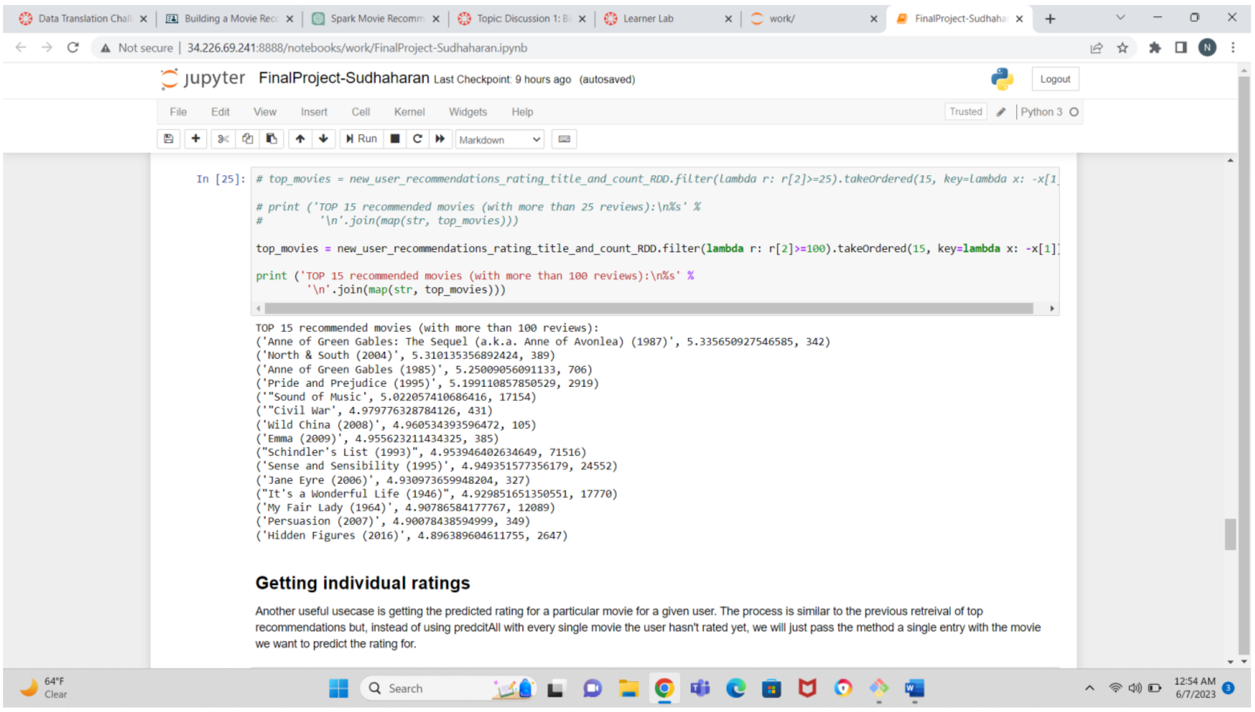
Getting individual ratings

Another useful usecase is getting the predicted rating for a particular movie for a given user. The process is similar to the previous retreival of top

TOP 15 recommended movies (with more than 25 reviews):
('"Very Potter Sequel', 5.402075474560762, 35)
('Sense & Sensibility (2008)', 5.3776839046314455, 69)
('Anne of Green Gables: The Sequel (a.k.a. Anne of Avonlea) (1987)', 5.335650927546585, 342)
('Cranford (2007)', 5.323038303364424, 35)
('North & South (2004)', 5.310135356892424, 389)
('Drishyam (2013)', 5.255280979471657, 37)
('Anne of Green Gables (1985)', 5.25009056091133, 706)
('Pride and Prejudice (1995)', 5.199110857850529, 2919)
('Murder on the Orient Express (2010)', 5.182371057817914, 29)
('I Can Only Imagine (2018)', 5.180623692610432, 30)
('Winter in Prostokvashino (1984)', 5.1467791284786335, 67)
('Boys (2014)', 5.141713053618693, 96)
('Little Dorrit (2008)', 5.132472734383583, 55)
("Won't You Be My Neighbor? (2018)", 5.0994357608513745, 83)
('The Case for Christ (2017)', 5.09381497368954, 33)

User 2 - Scenario 2

Scenario 2, where movies with less than 100 ratings were filtered out, presents the top 15 recommended movies for User 2. These recommendations reflect a broader range of popular and critically acclaimed films, including both mainstream and niche choices. User 2's preferences demonstrate a mix of popular choices and lesser-known gems from various genres.


```
In [25]: # top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered(15, key=lambda x: -x[1

         # print ('TOP 15 recommended movies (with more than 25 reviews):\n%s' %
         #         '\n'.join(map(str, top_movies)))

         top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=100).takeOrdered(15, key=lambda x: -x[1]

         print ('TOP 15 recommended movies (with more than 100 reviews):\n%s' %
                 '\n'.join(map(str, top_movies)))
```

```
TOP 15 recommended movies (with more than 100 reviews):
('Anne of Green Gables: The Sequel (a.k.a. Anne of Avonlea) (1987)', 5.335650927546585, 342)
('North & South (2004)', 5.310135356892424, 389)
('Anne of Green Gables (1985)', 5.25009056091133, 706)
('Pride and Prejudice (1995)', 5.199110857850529, 2919)
('"Sound of Music', 5.022057410686416, 17154)
('"Civil War', 4.979776328784126, 431)
('Wild China (2008)', 4.960534393596472, 105)
('Emma (2009)', 4.955623211434325, 385)
("Schindler's List (1993)", 4.953946402634649, 71516)
('Sense and Sensibility (1995)', 4.949351577356179, 24552)
('Jane Eyre (2006)', 4.930973659948204, 327)
("It's a Wonderful Life (1946)", 4.929851651350551, 17770)
('My Fair Lady (1964)', 4.90786584177767, 12089)
('Persuasion (2007)', 4.90078438594999, 349)
('Hidden Figures (2016)', 4.896389604611755, 2647)
```

Getting individual ratings

Another useful usecase is getting the predicted rating for a particular movie for a given user. The process is similar to the previous retreival of top recommendations but, instead of using predcitAll with every single movie the user hasn't rated yet, we will just pass the method a single entry with the movie we want to predict the rating for.

```
TOP 15 recommended movies (with more than 100 reviews):
('Anne of Green Gables: The Sequel (a.k.a. Anne of Avonlea) (1987)', 5.335650927546585, 342)
('North & South (2004)', 5.310135356892424, 389)
('Anne of Green Gables (1985)', 5.25009056091133, 706)
('Pride and Prejudice (1995)', 5.199110857850529, 2919)
('"Sound of Music', 5.022057410686416, 17154)
('"Civil War', 4.979776328784126, 431)
('Wild China (2008)', 4.960534393596472, 105)
('Emma (2009)', 4.955623211434325, 385)
("Schindler's List (1993)", 4.953946402634649, 71516)
('Sense and Sensibility (1995)', 4.949351577356179, 24552)
('Jane Eyre (2006)', 4.930973659948204, 327)
("It's a Wonderful Life (1946)", 4.929851651350551, 17770)
('My Fair Lady (1964)', 4.90786584177767, 12089)
('Persuasion (2007)', 4.90078438594999, 349)
('Hidden Figures (2016)', 4.896389604611755, 2647)
```

Insights

The analysis of the results revealed interesting insights into the movie preferences of the users. The recommendations generated for each user demonstrate the power of collaborative filtering in providing personalized movie suggestions. By analyzing the preferences and ratings of other users, the recommendation engine identifies movies that align with the unique tastes of each user, enhancing their movie-watching experience. These insights can have significant business implications for Ripe Pumpkins. By understanding individual customer preferences and providing personalized recommendations, the company can enhance customer satisfaction and increase the likelihood of customers staying with their services. Additionally, the analysis can help Ripe Pumpkins identify opportunities to differentiate themselves from competitors and improve their recommendation engine.

References

1. Codementor. “Building a Movie Recommendation Service using Apache Spark” Codementor, 14 Sep. 2015, https://www.codementor.io/@jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw
2. ProjectsBasedLearning. “Creating a Movies Recommendation Engine” ProjectsBasedLearning, 07 Jun. 2021, https://projectsbasedlearning.com/apache-spark-machine-learning/machine-learning-project-creating-movies-recommendation-engine-using-apache-spark/
3. Medium. “Movie Recommendation System using Spark MLlib” Medium, 10 May. 2017, https://medium.com/edureka/spark-mllib-e87546ac268
4. Databricks. “Movie Recommendation using Spark MLlib” Databricks, n.d., https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3741049972324885/1723574684687027/4413065072037724/latest.html