Item Based Collaborative Filtering

Item Based Collaborative Filtering is an algorithm widely used in Recommendation systems. In this algorithm, we try to find a relation between products using customer behavior.

Some steps of doing this are:

1. Find every pair of movies that were watched by the same person.

2. Measure similarity of their ratings across all users who watched those same movies.

3. Sort by movie, then by similarity strengths.

We can convert this to a Spark problem by:

1. Map the input ratings to the user ID in the format – (userID, (movieID, rating))

2. Find every movie pairs rated by the same user. This can be done through self-join in a dataset. Now we have – (userID, ((movieID, rating1), (movieID, rating1),…))

3. Filter out duplicate pairs.

4. Make movie pairs the key and map it like this – ((movieID1, movieID2), (rating1, rating2))

5. Use the groupByKey function to get every rating pair found for every movie pair.

6. Compute the similarity between ratings for each movie in the pair

7. Sort, save and display results.

We will cache the RDD and persist it so that we are using so that Spark does not have to create it from scratch every time we perform an action on it.

We are finding the similarity between any two movies using the Cosine Similarity index. This score is basically a ratio which is calculated using the ratings of a movie pair.

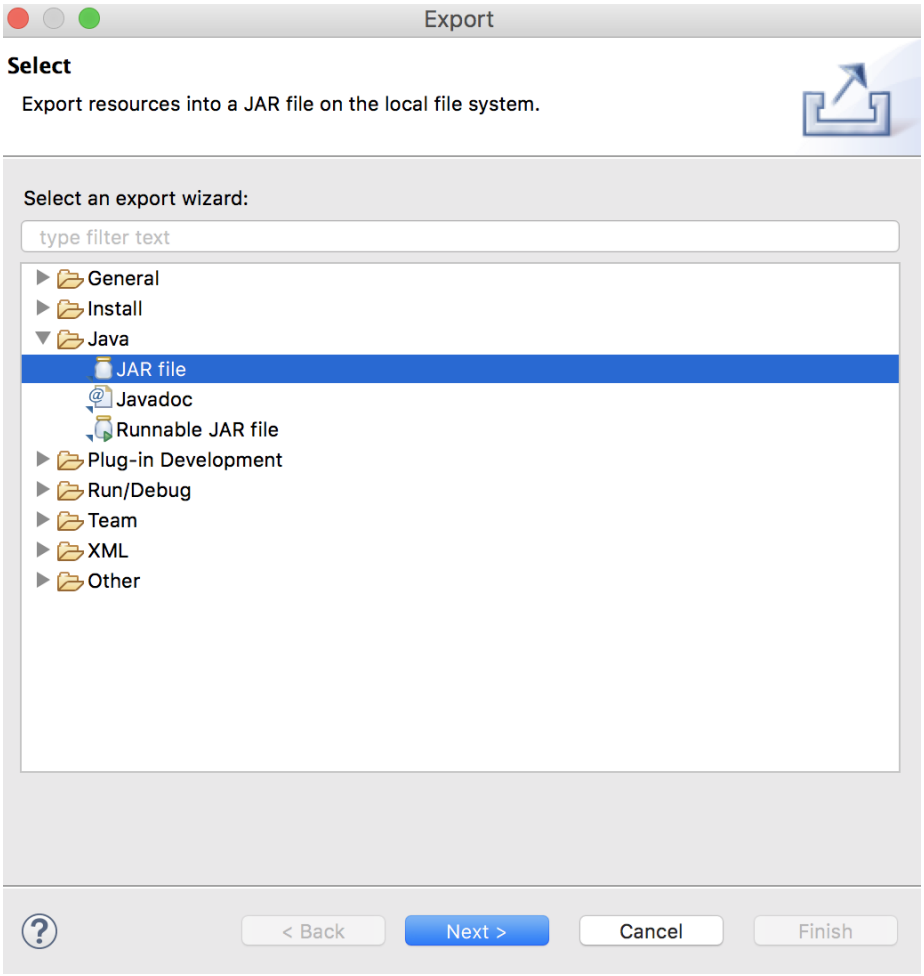
We have also set the Score Threshold and Co-occurrence Threshold for similarity as 0.97 and 50.0 respectively. This threshold acts as a filter against the same movie being used as input to be given as result.

On the movie ID entered as input, we will return 10 recommendations which are most similar to that movie.

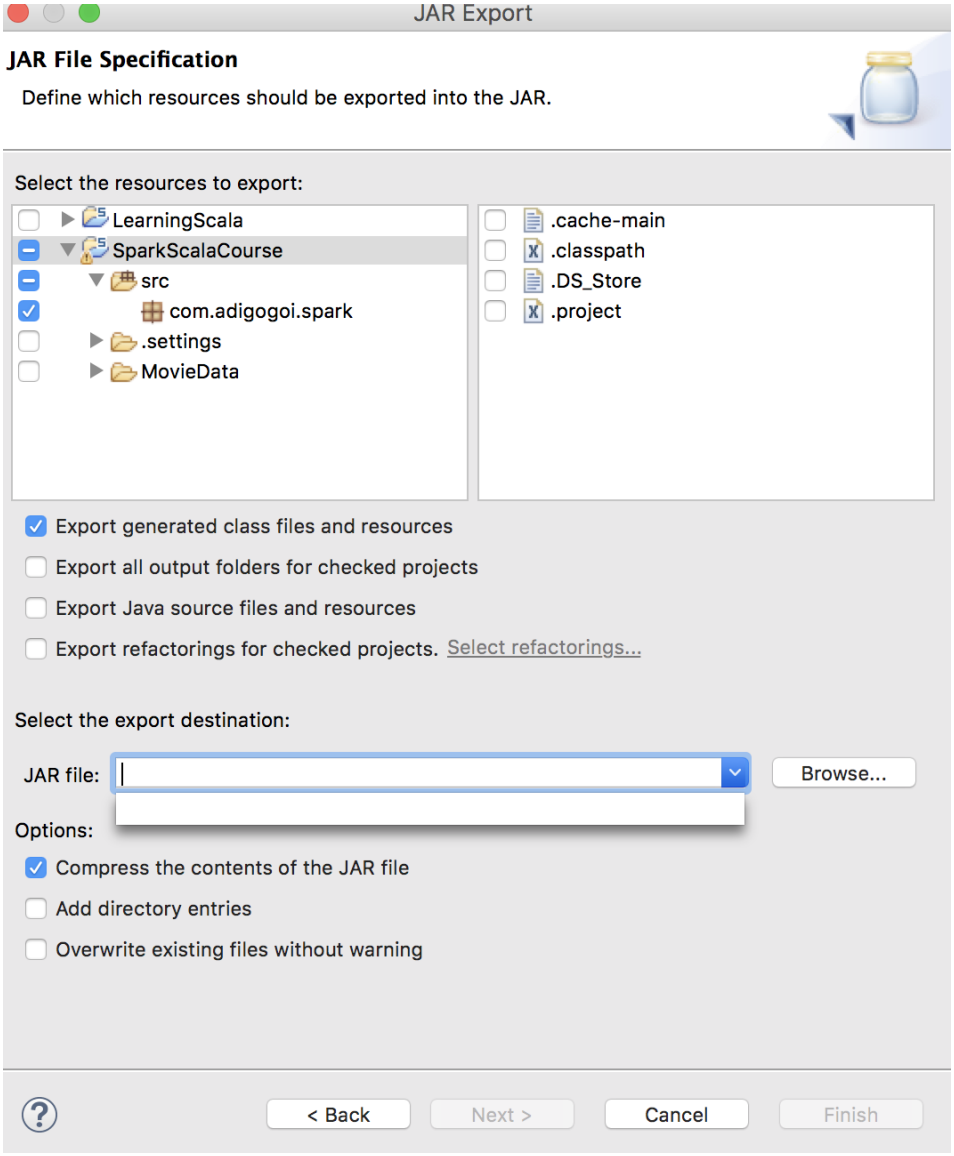
Compiling and Running the Program:

In this case, we will use a different method to compile and run the program – through a JAR file. We will compile the project into a JAR file and then run it.

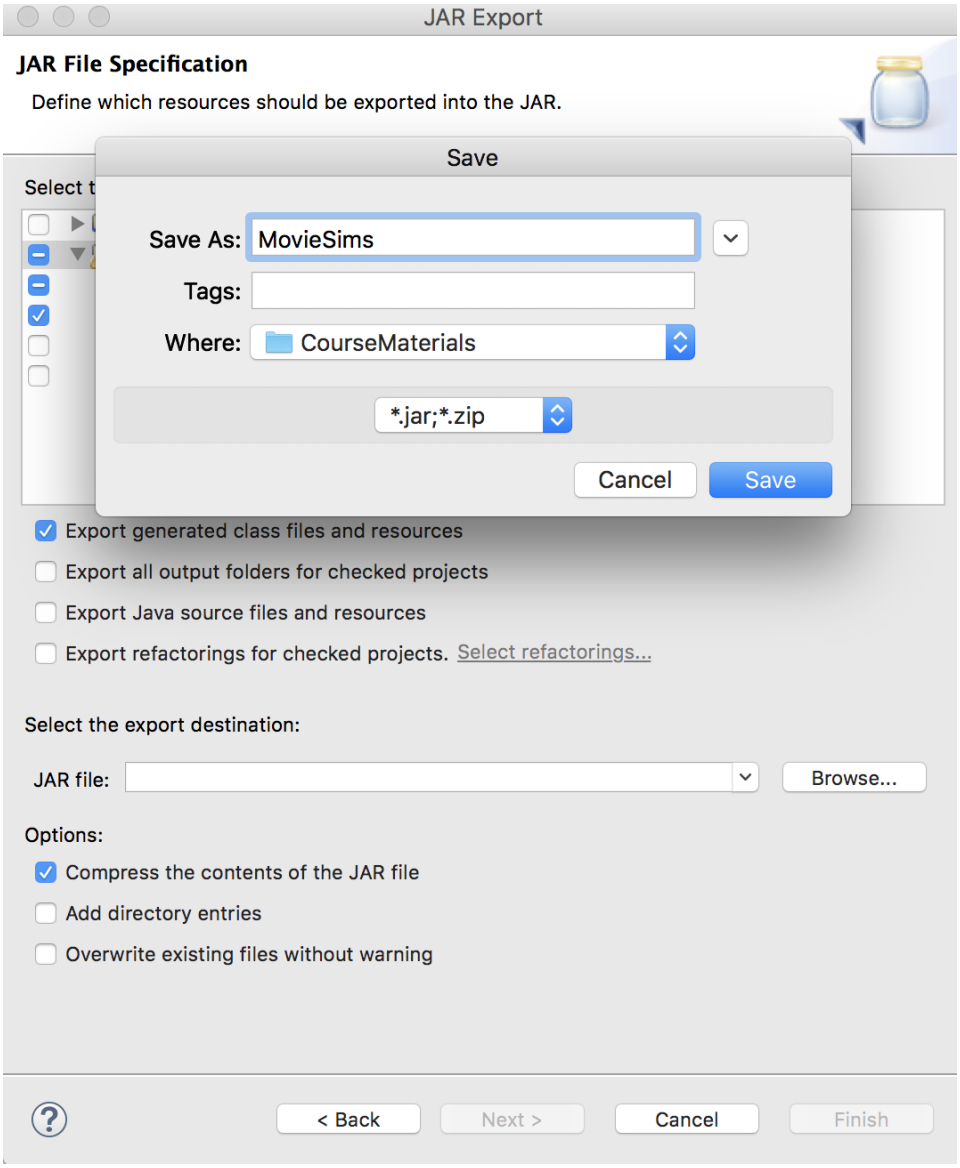
1. Select the export command from the IDE you are using. Select Java > JAR file and click next.



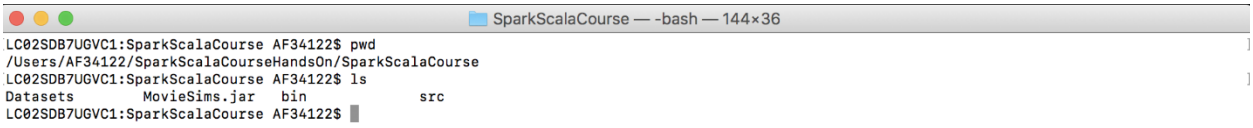
2. Drill-down to the collection you want to export and tick it. Then click on the Browse button of the ‘Select the export destination:’ field.



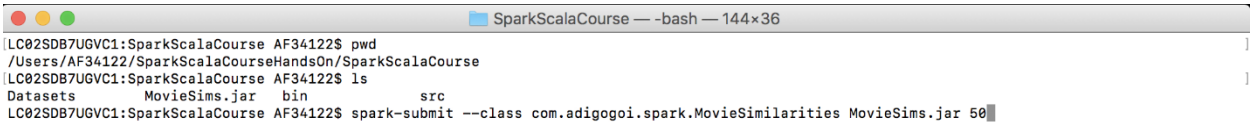
3. Select the folder where you want to store the JAR file and type the name you want for the JAR file. Click Save.



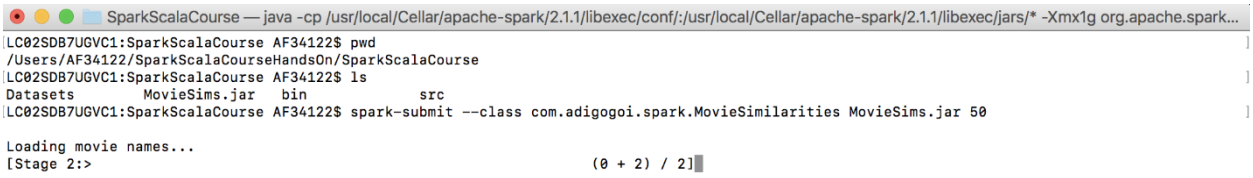
4. Go to the same folder where you have saved the JAR file. Ensure that the data files are in the correct relative folder.



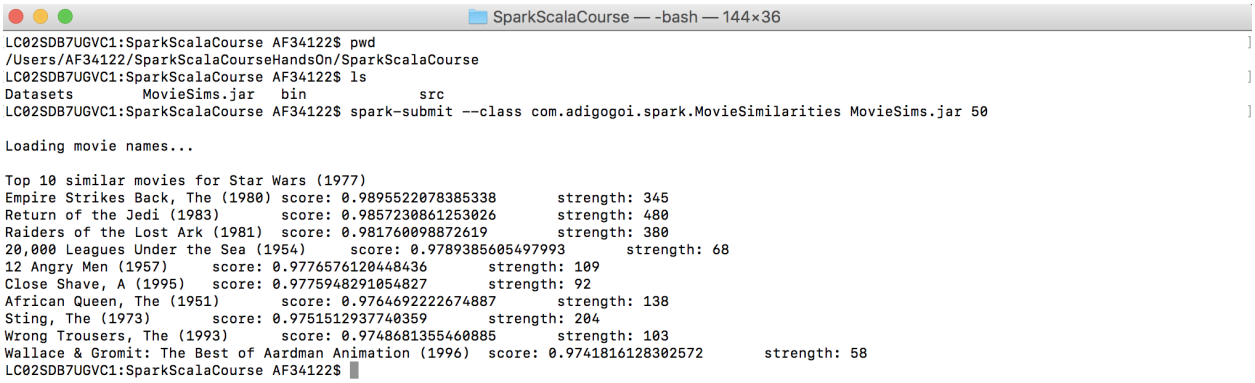
5. We will use the 'spark-submit' command to execute the JAR file. The '—class' parameter is followed by the collection name of our project, followed by the JAR file name and the movie ID we want similarities to. In this case we have used Movie ID 50, which is Star Wars.



6. The algorithm will take some time to run in stages.



7. The algorithm returns 10 recommendations based on similarities between these movies and Star Wars.



With these accurate recommendations I came to the following conclusions:

1. This simpler approach was much more accurate as compared to the sophisticated ALS MLlib approach (done in a different project). Although this was based on the deep knowledge that I had of the dataset.

2. This approach is only restricted to recommendations of products by users and their ratings. If the data were any different, things would have become much more difficult.