Machine Learning using Spark MLlib

Spark MLlib allows us to perform Machina Learning and Data Science operations for huge datasets on distributed clusters.

Before we start, we should realize that not all Machine Learning algorithms are easy to implement on distributed systems. This makes ML in Spark challenging. Sparks MLlib and the recent ML library has come far in creating algorithms, but it has a lot of ground to cover. Some of the algorithms covered in MLlib are:

1. Feature Extraction
2. Basic Statistics
3. Linear and Logistic Regression
4. Support Vector Machines
5. Naïve Bayes Classifier
6. Decision Trees
7. Decision Trees

MLlib comes with its specialized data types:

1. Vector – Vector is a way of representing large arrays of values, which may have missing values. For this we have Sparse and Dense vectors.
2. Labeled Point – Supervised ML is all about labelling data points based on some rules. Labeled Point helps us in making that association.
3. Rating – This data type is specialized for data which contains ratings for products given by consumers like in movies or online shopping.

Getting the Data:

We will be working on the movie rating dataset to predict movie recommendations. The data is read from the file and split using '\t' delimiter. We only take the columns that are needed by us (UserID, movieID and Rating) and store it in the final dataframe called 'ratings'.

Creating the ML Model:

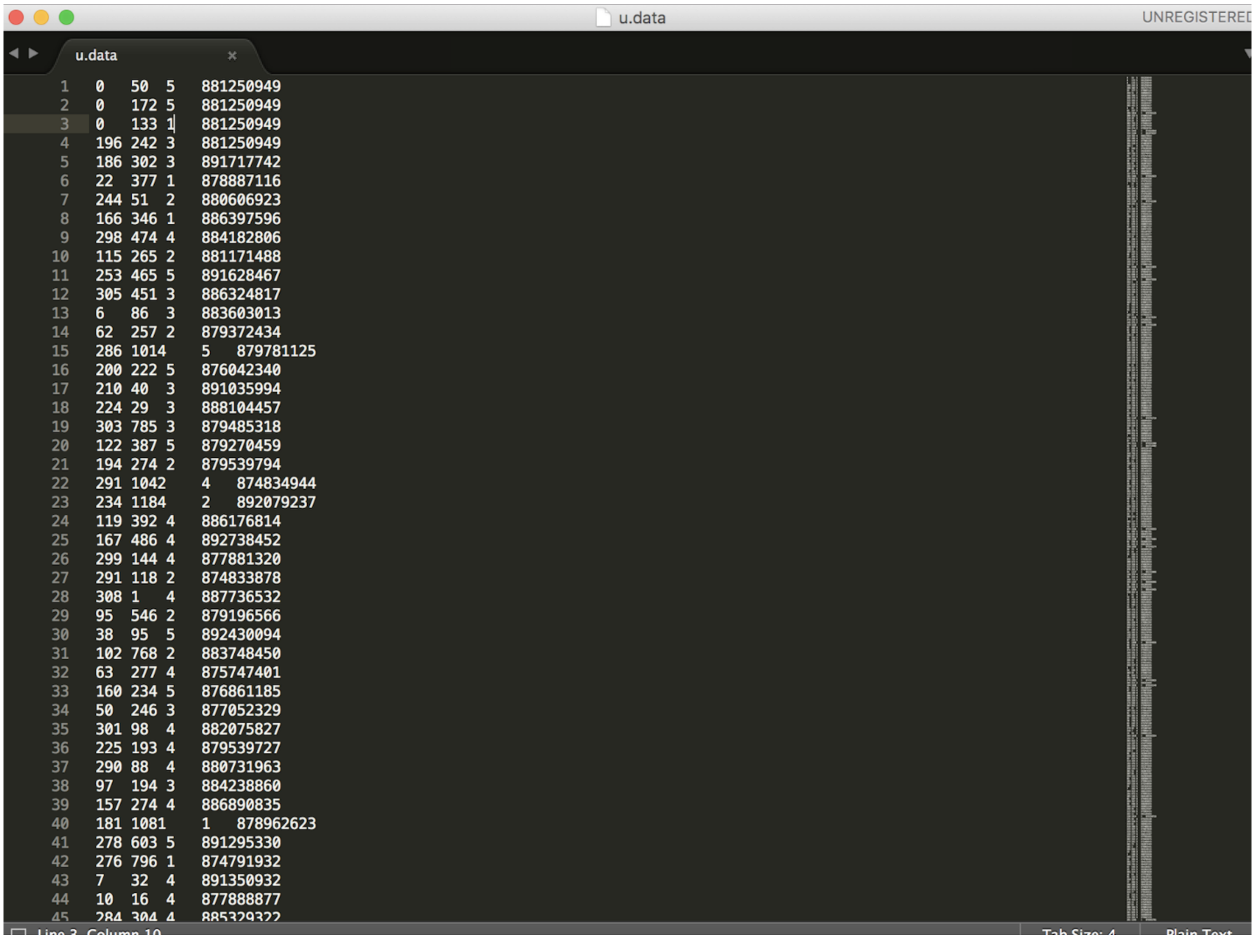
We will be using the Alternating Least Squares (ALS) model to learn from the ratings data. We will train this model using parameters like Rank and Number of Iterations on the training data (ratings dataframe). We store the trained model in the variable 'model'.

We later on use the 'recommendProducts' function to create 10 recommendations for any given userID provided through the command line.

Running the Program:

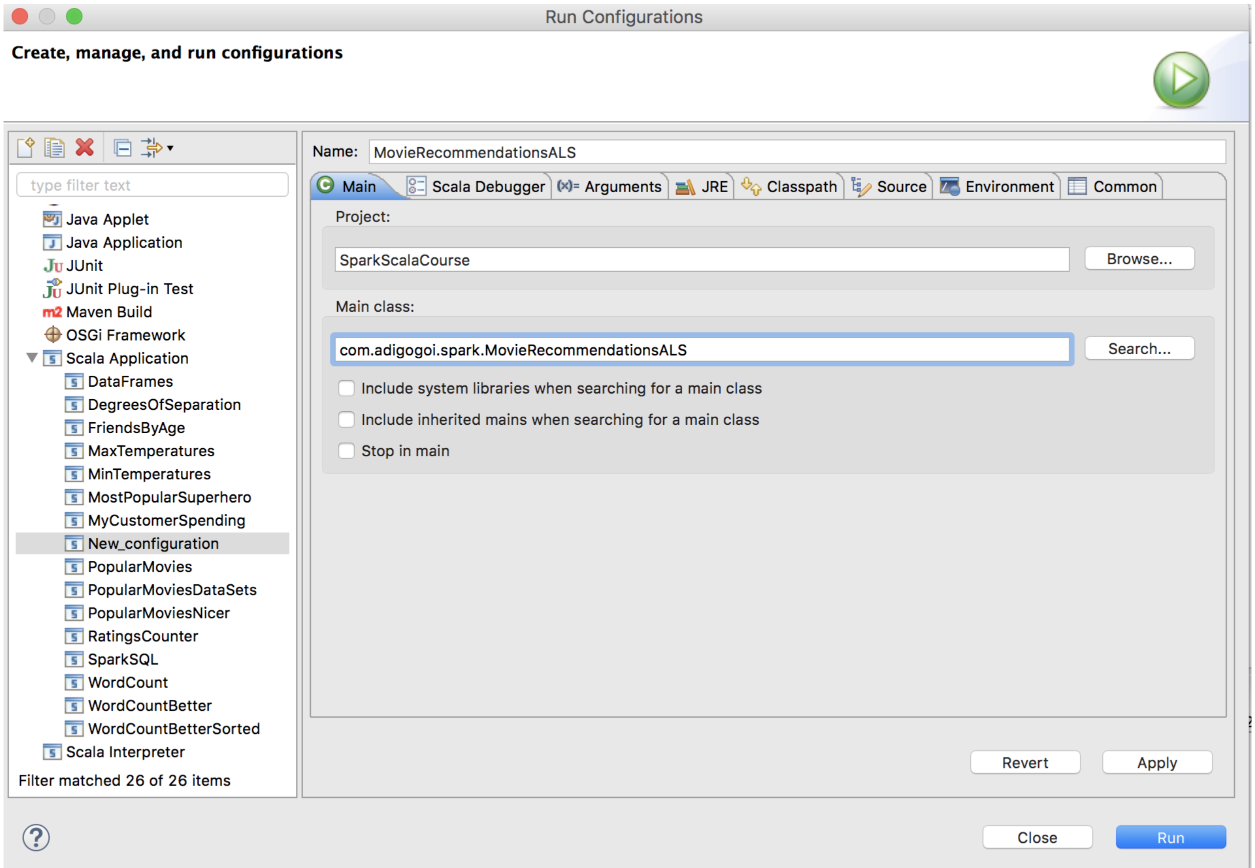
To test how ALS model of MLlib works, we will have to make a few adjustments to the dataset and then run the code in a specific way.

1. We will create a fictional user. This user loves Sci-fi movies and has hence rated Star Wars (Movie ID 50) a 5. This user hates romantic movies and has given Gone With The Wind (Movie ID 133) a 1 rating. This will be stored in our data file as User ID 0.

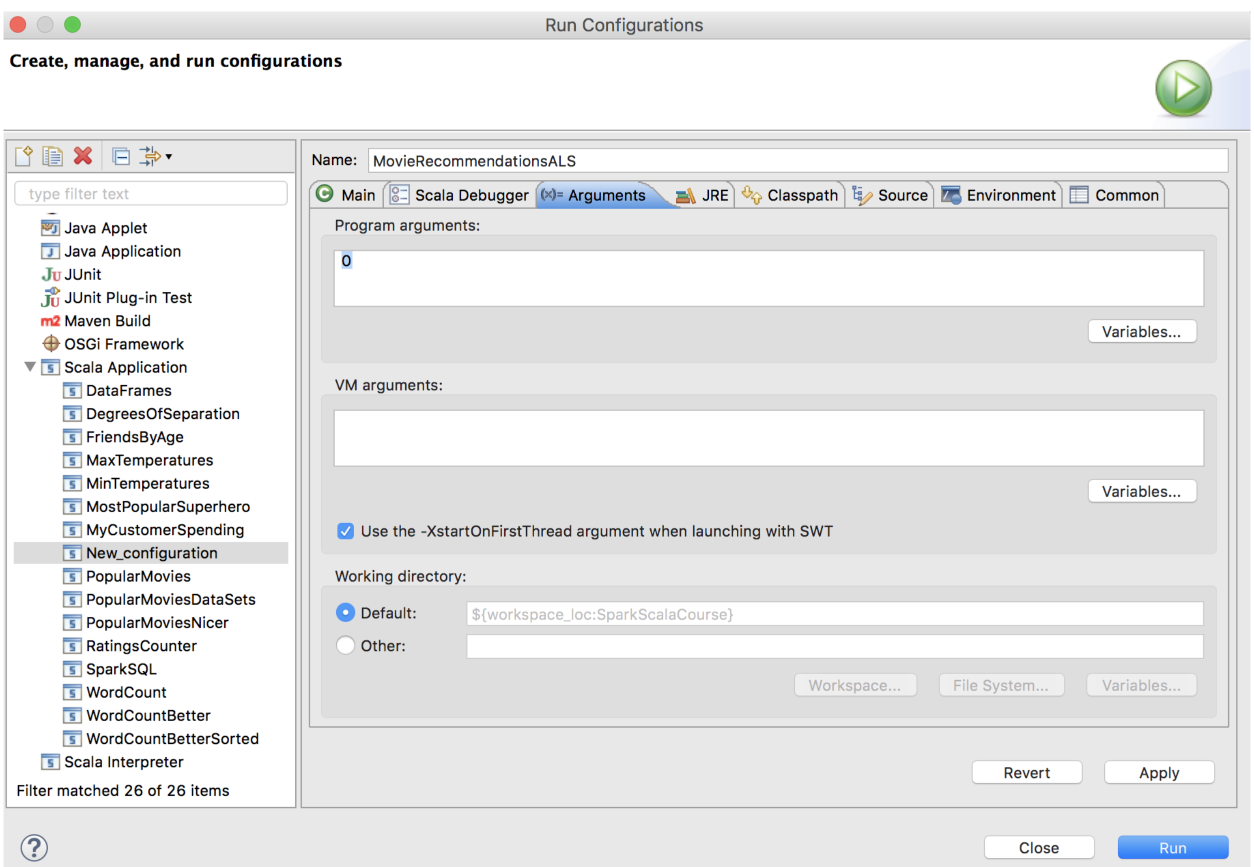



```
0	50	5	881250949
0	172	5	881250949
0	133	1	881250949
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817
6	86	3	883603013
62	257	2	879372434
286	1014	5	879781125
200	222	5	876042340
210	40	3	891035994
224	29	3	888104457
303	785	3	879485318
122	387	5	879270459
194	274	2	879539794
291	1042	4	874834944
234	1184	2	892079237
119	392	4	886176814
167	486	4	892738452
299	144	4	877881320
291	118	2	874833878
308	1	4	887736532
95	546	2	879196566
38	95	5	892430094
102	768	2	883748450
63	277	4	875747401
160	234	5	876861185
50	246	3	877052329
301	98	4	882075827
225	193	4	879539727
290	88	4	880731963
97	194	3	884238860
157	274	4	886890835
181	1081	1	878962623
278	603	5	891295330
276	796	1	874791932
7	32	4	891350932
10	16	4	877888877
284	304	4	885329322
```

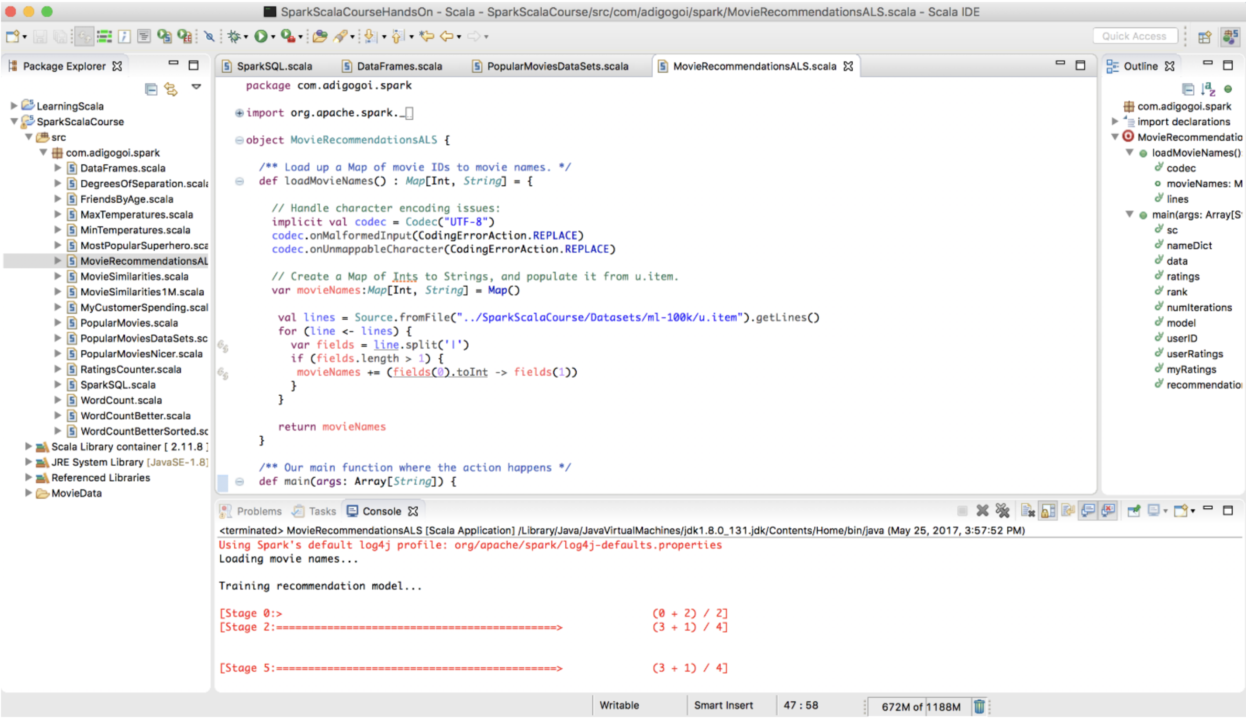
2. After that adjustment, we will run the code as if through a command line. For this we have to go to the 'Run Configuration' option in the Scala IDE for Eclipse that we are using and then give the project name and the Main class of my project.



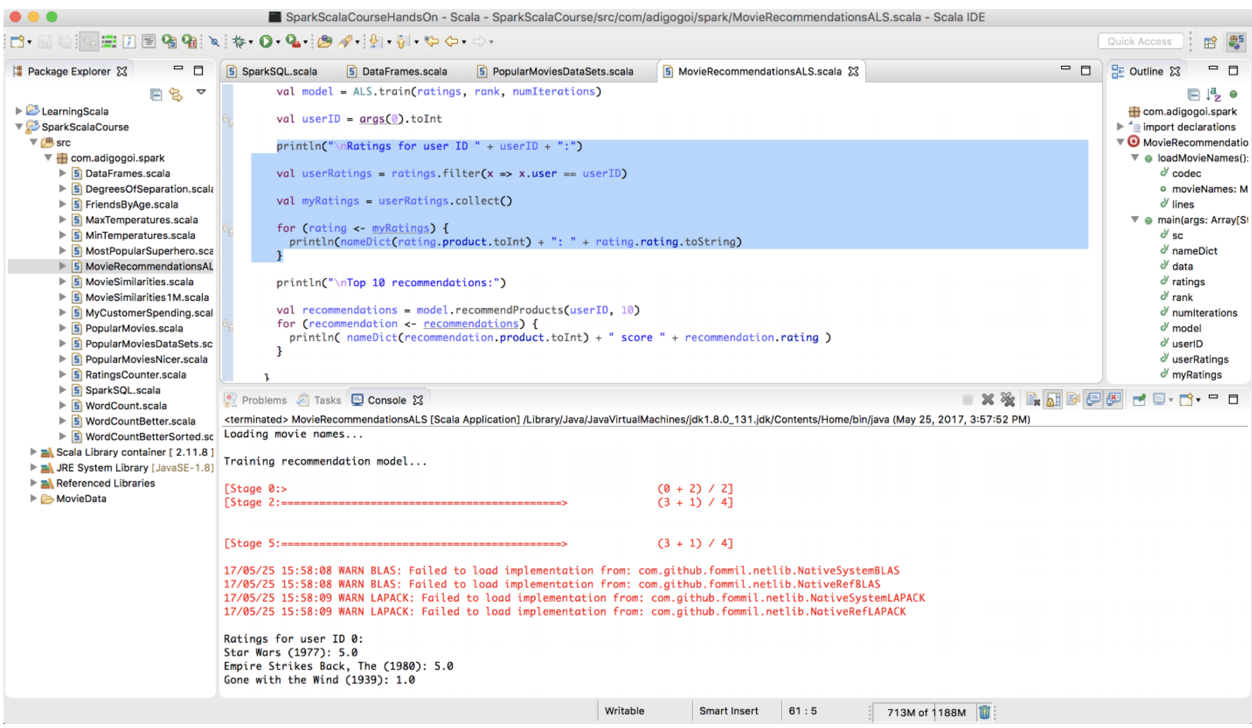
3. Next we go to the ‘Arguments’ window and enter ‘0’, which is the User ID we want to predict our recommendations for.



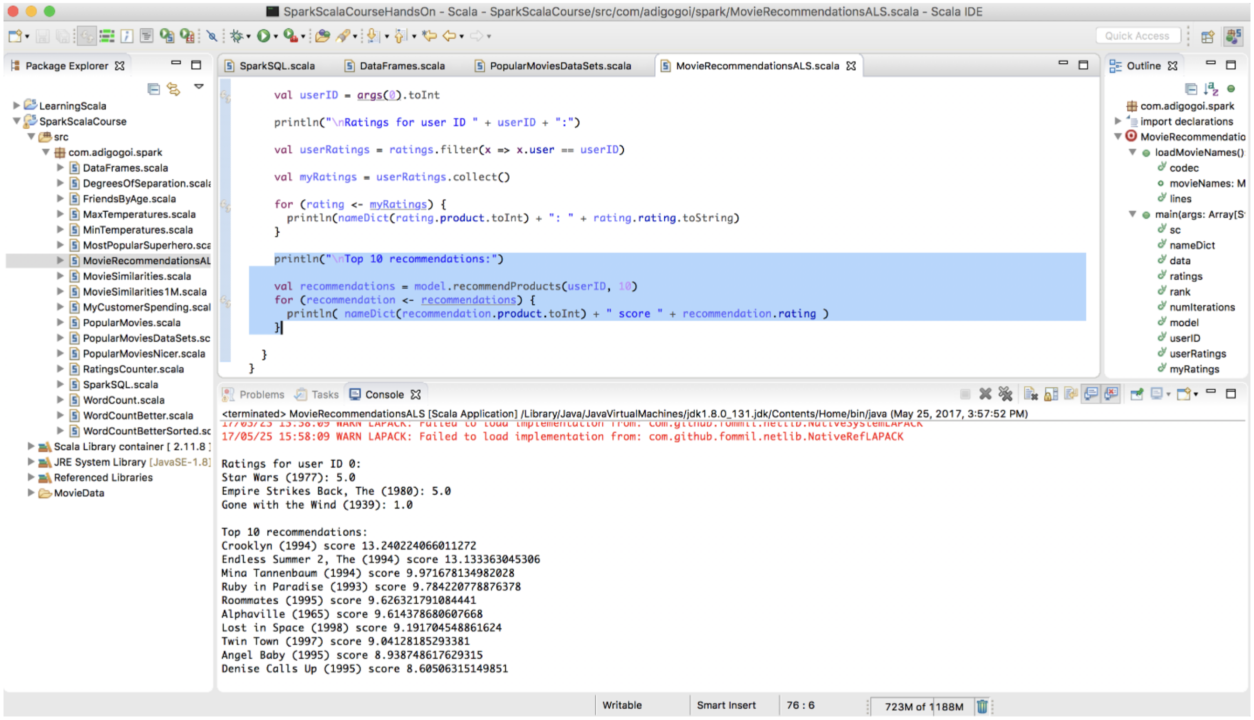
4. The program will the train the model for the number of stages specified by us.



5. According to my code, the next thing to be displayed will be the movies rated by the User ID 0 and the rating given for each movie.



6. The ALS model will then make predictions and provide 10 recommendations for User ID 0.



Conclusions:

After watching the model work and seeing its inaccurate predictions, I have come to the following conclusions:

1. The ALS algorithm is very sensitive to the parameters provided to it. It takes a little time to find the ideal parameters for a dataset. But due to the black-box nature of the algorithms, it becomes difficult to determine the correct parameters.

2. Complicated is not always better. Sometimes a simple approach to the problem may provide better results.

3. Never blindly trust results when analyzing Big Data. Small problems in algorithms magnify when dealing with such large datasets. Quality of data is the real issue here.