MOVIE RECOMENDATION

ACHYUTH PILLY

TABLE OF CONTENTS

PROJECT REQUIREMENTS

Project overview

By the time we finish this project, we would have learned:

- How to load the raw data from a file.
- How to extract information from the dataset.
- Import Spark's ALS recommendation model and inspect the train method.
- Import a class and inspect it.
- Construct the RDD of objects.
- Train the ALS model.
- Make predictions.

Prerequisite

The software technology we are going to be needing is:

- Java installed Hadoop concepts
- Hadoop installed Java concepts
- Spark installed
- Ubuntu OS

Understanding the data

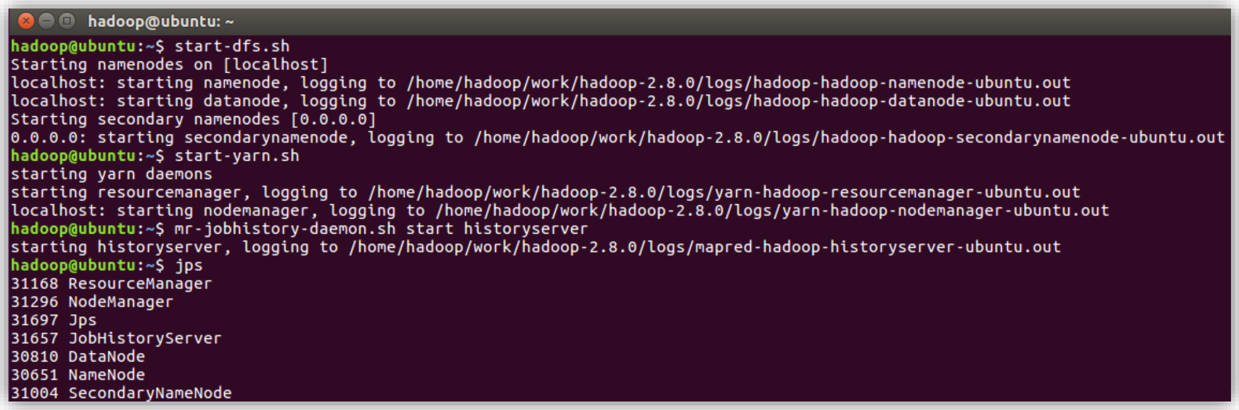
The MovieLens 100k dataset is a set of 100,000 data points related to ratings given by a set of users to a set of movies. It also contains movie metadata and user profiles.

The readme file contains more information on the dataset, including the variables present in each data file.

- The u.user file contains the user id, age, gender, occupation, and ZIP code fields.
- The u.item file contains the movie id, title, release data, and IMDB link fields and a set of fields related to movie category data.
- The u.data file contains the user id, movie id, rating (1-5 scale), and timestamp fields.

MOVING THE DATA TO HDFS

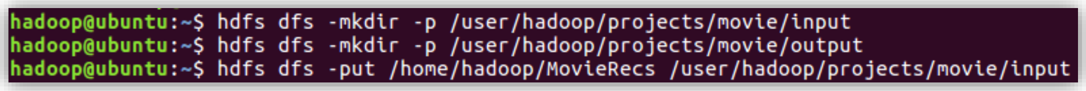
We need to start Hadoop if we haven't done so. We use the commands **start-dfs.sh**, **start-yarn.sh** and **mr-jobhistory-daemon.sh start historyserver** to do so.



```
hadoop@ubuntu: ~
hadoop@ubuntu:~$ start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/work/hadoop-2.8.0/logs/hadoop-hadoop-namenode-ubuntu.out
localhost: starting datanode, logging to /home/hadoop/work/hadoop-2.8.0/logs/hadoop-hadoop-datanode-ubuntu.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/hadoop/work/hadoop-2.8.0/logs/hadoop-hadoop-secondarynamenode-ubuntu.out
hadoop@ubuntu:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/hadoop/work/hadoop-2.8.0/logs/yarn-hadoop-resourcemanager-ubuntu.out
localhost: starting nodemanager, logging to /home/hadoop/work/hadoop-2.8.0/logs/yarn-hadoop-nodemanager-ubuntu.out
hadoop@ubuntu:~$ mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /home/hadoop/work/hadoop-2.8.0/logs/mapred-hadoop-historyserver-ubuntu.out
hadoop@ubuntu:~$ jps
31168 ResourceManager
31296 NodeManager
31697 Jps
31657 JobHistoryServer
30810 DataNode
30651 NameNode
31004 SecondaryNameNode
```

Figure 1. *Starting Hadoop.*

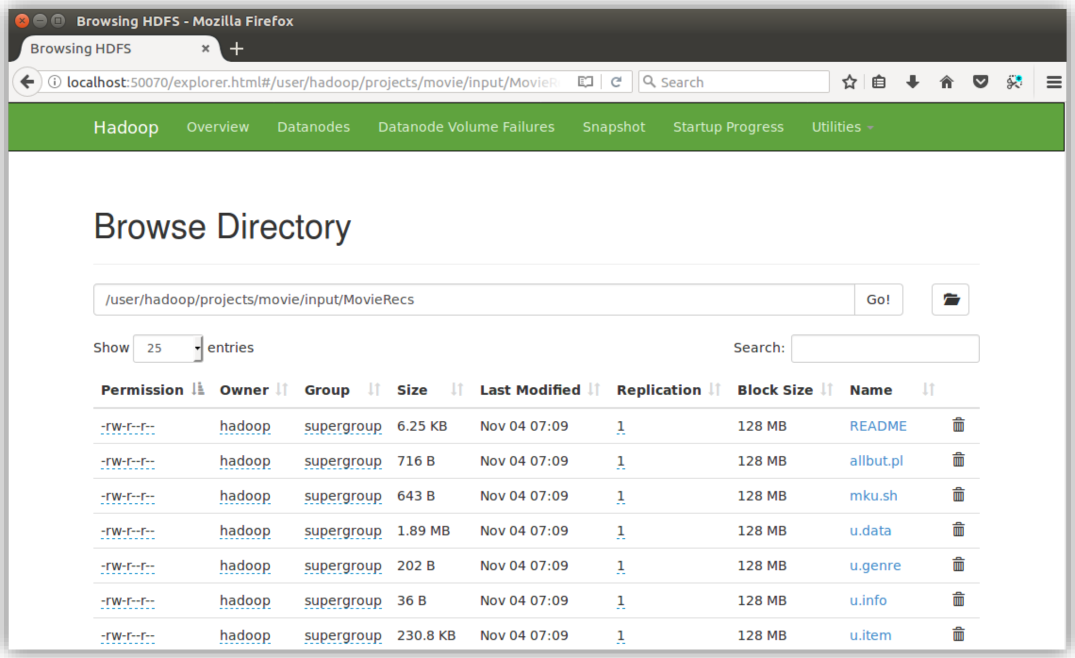
In order to move our data to HDFS we have first to create a folder for it. We create the folder and parent folders, if they don't exist, with the command **hdfs dfs -mkdir -p /user/hadoop/projects/movie/input**. Then we use the command **hdfs dfs -put /home/hadoop/MovieRecs /user/hadoop/projects/movie/input** to copy the files into HDFS.



```
hadoop@ubuntu:~$ hdfs dfs -mkdir -p /user/hadoop/projects/movie/input
hadoop@ubuntu:~$ hdfs dfs -mkdir -p /user/hadoop/projects/movie/output
hadoop@ubuntu:~$ hdfs dfs -put /home/hadoop/MovieRecs /user/hadoop/projects/movie/input
```

Figure 2. *Putting our data in HDFS.*

We can see that our data has been copied in HDFS by browsing to the folder.



Browse Directory

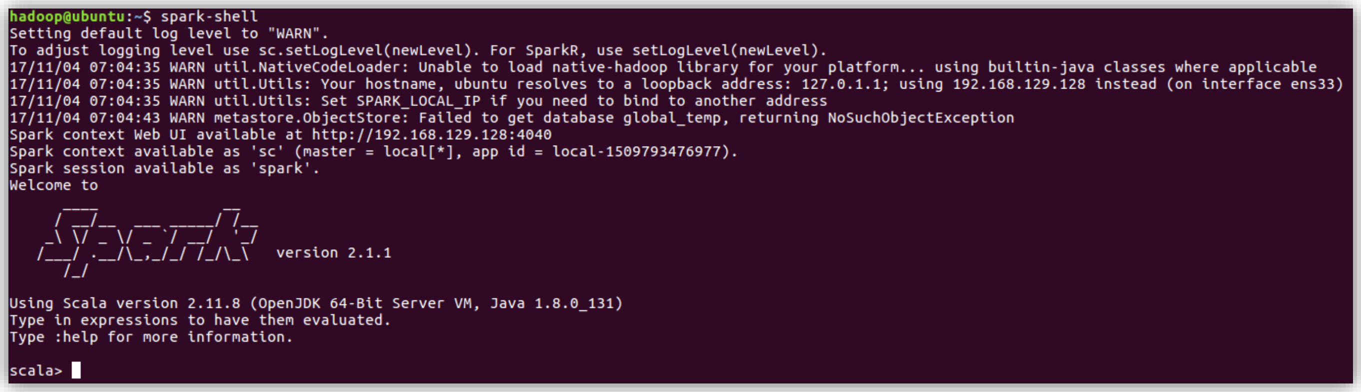
/user/hadoop/projects/movie/input/MovieRecs

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	6.25 KB	Nov 04 07:09	1	128 MB	README
-rw-r--r--	hadoop	supergroup	716 B	Nov 04 07:09	1	128 MB	allbut.pl
-rw-r--r--	hadoop	supergroup	643 B	Nov 04 07:09	1	128 MB	mku.sh
-rw-r--r--	hadoop	supergroup	1.89 MB	Nov 04 07:09	1	128 MB	u.data
-rw-r--r--	hadoop	supergroup	202 B	Nov 04 07:09	1	128 MB	u.genre
-rw-r--r--	hadoop	supergroup	36 B	Nov 04 07:09	1	128 MB	u.info
-rw-r--r--	hadoop	supergroup	230.8 KB	Nov 04 07:09	1	128 MB	u.item

Figure 3. *Our data in HDFS.*

PART 1 – DATA PROCESSING

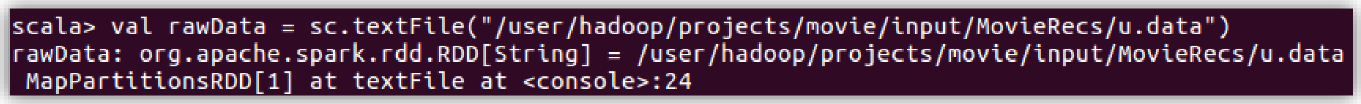
First, we need to start our Spark. We do it using the command **spark-shell**.



```
hadoop@ubuntu:~$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/11/04 07:04:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/11/04 07:04:35 WARN util.Utils: Your hostname, ubuntu resolves to a loopback address: 127.0.1.1; using 192.168.129.128 instead (on interface ens33)
17/11/04 07:04:35 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/11/04 07:04:43 WARN metastore.ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://192.168.129.128:4040
Spark context available as 'sc' (master = local[*], app id = local-1509793476977).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_131)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```

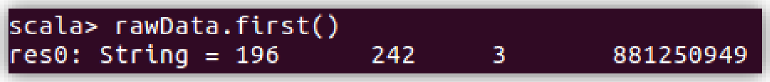
Figure 4. *Starting Spark.*

We load the raw ratings data from the u.data file using **val rawData = sc.textFile("/user/hadoop/projects/movie/input/MovieRecs/u.data")** and inspect the dataset using **rawData.first()**.



```
scala> val rawData = sc.textFile("/user/hadoop/projects/movie/input/MovieRecs/u.data")
rawData: org.apache.spark.rdd.RDD[String] = /user/hadoop/projects/movie/input/MovieRecs/u.data
 MapPartitionsRDD[1] at textFile at <console>:24
```

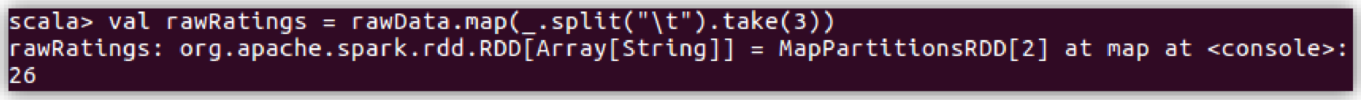
Figure 5. *Loading our raw ratings data.*



```
scala> rawData.first()
res0: String = 196      242     3       881250949
```

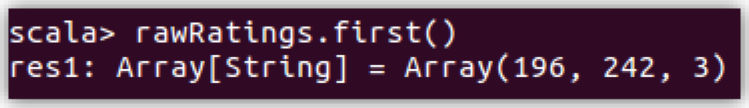
Figure 6. *Inspecting our raw data.*

We extract the user id, movie id and rating only from the dataset using **val rawRatings = rawData.map(_.split("\t").take(3))** and inspect the extracted dataset with **rawRatings.first()**.



```
scala> val rawRatings = rawData.map(_.split("\t").take(3))
rawRatings: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:
26
```

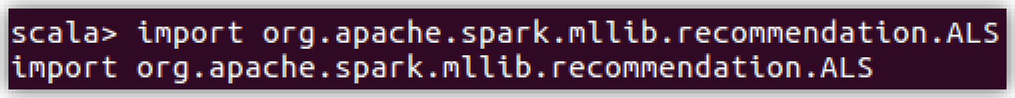
Figure 7. *Extracting rating data.*



```
scala> rawRatings.first()
res1: Array[String] = Array(196, 242, 3)
```

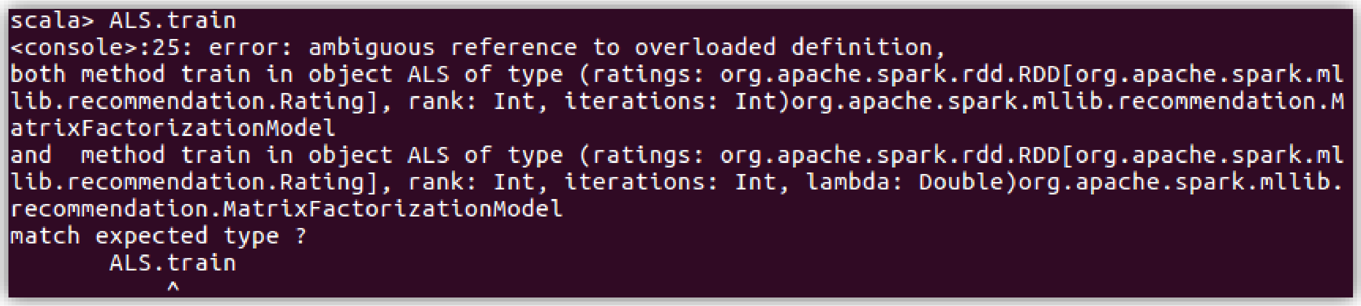
Figure 8. *Inspecting our extracted data.*

We import Spark's ALS recommendation model using **import org.apache.spark.mllib.recommendation.ALS** and inspect the train method with the **ALS.train** command.



```
scala> import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.ALS
```

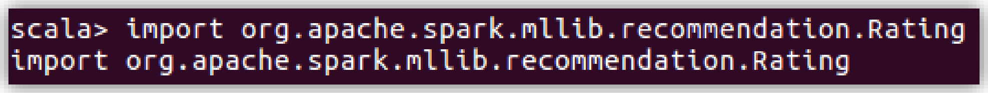
Figure 9. *Importing Spark's ALS recommendation model.*



```
scala> ALS.train
<console>:25: error: ambiguous reference to overloaded definition,
both method train in object ALS of type (ratings: org.apache.spark.rdd.RDD[org.apache.spark.ml
lib.recommendation.Rating], rank: Int, iterations: Int)org.apache.spark.mllib.recommendation.M
atrixFactorizationModel
and  method train in object ALS of type (ratings: org.apache.spark.rdd.RDD[org.apache.spark.ml
lib.recommendation.Rating], rank: Int, iterations: Int, lambda: Double)org.apache.spark.mllib.
recommendation.MatrixFactorizationModel
match expected type ?
       ALS.train
           ^
```

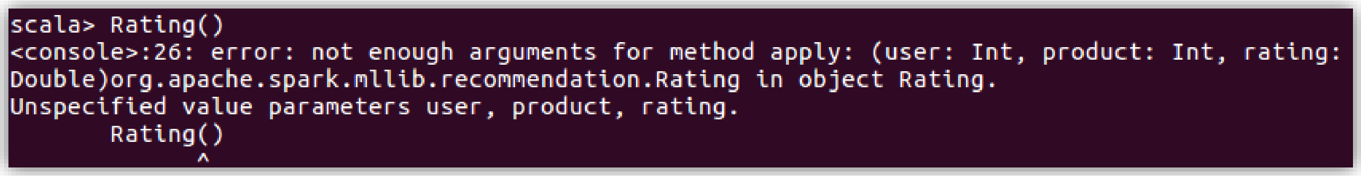
Figure 10. *Inspecting the train method.*

We import the Rating class **import org.apache.spark.mllib.recommendation.Rating** using and inspect it with **Rating()**.



```
scala> import org.apache.spark.mllib.recommendation.Rating
import org.apache.spark.mllib.recommendation.Rating
```

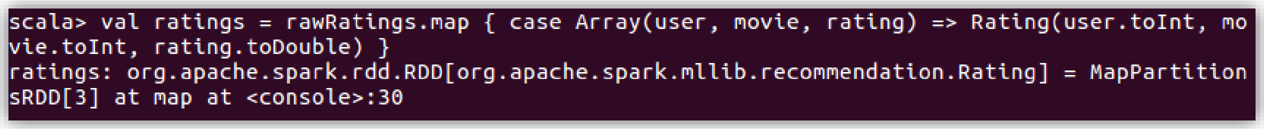
Figure 11. *Importing the Rating class.*



```
scala> Rating()
<console>:26: error: not enough arguments for method apply: (user: Int, product: Int, rating:
Double)org.apache.spark.mllib.recommendation.Rating in object Rating.
Unspecified value parameters user, product, rating.
       Rating()
             ^
```

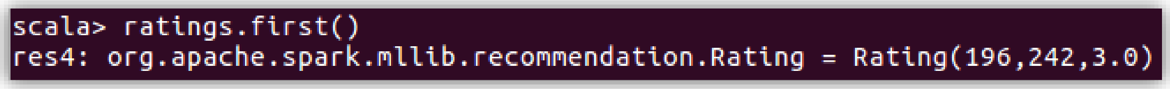
Figure 12. *Inspecting the Rating() class.*

We construct the RDD of Rating objects with **val ratings = rawRatings.map { case Array(user, movie, rating) => Rating(user.toInt, movie.toInt, rating.toDouble)** and verify it with **ratings.first()**.



```
scala> val ratings = rawRatings.map { case Array(user, movie, rating) => Rating(user.toInt, mo
vie.toInt, rating.toDouble) }
ratings: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating] = MapPartition
sRDD[3] at map at <console>:30
```

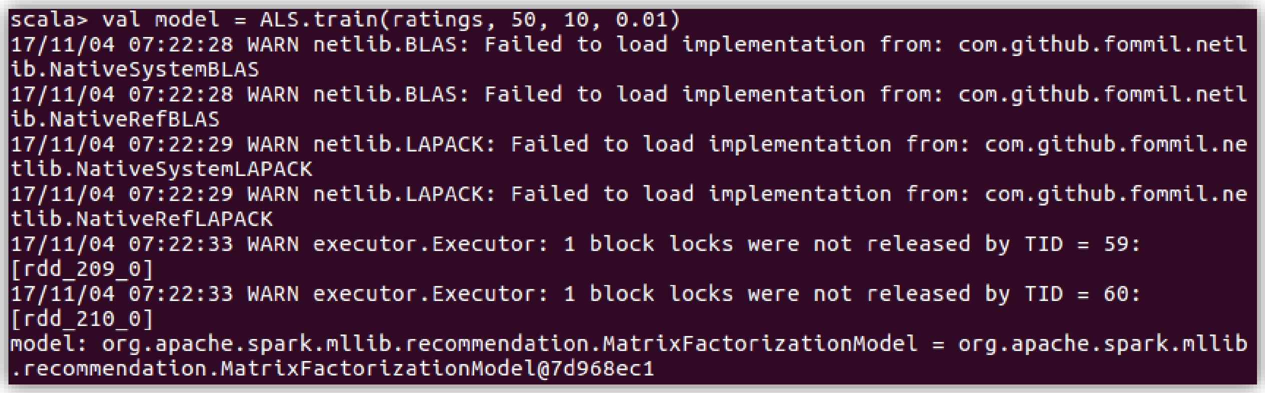
Figure 13. *Constructing the RDD of Rating objects.*



```
scala> ratings.first()
res4: org.apache.spark.mllib.recommendation.Rating = Rating(196,242,3.0)
```

Figure 14. *Verifying our RDD.*

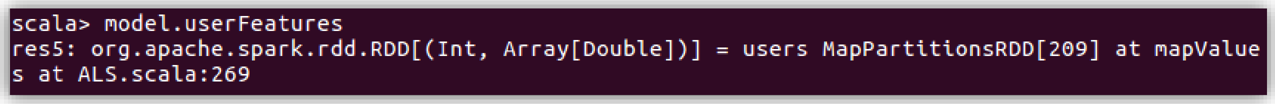
We train the ALS model with rank=50, iterations=10, lambda=0.01 using **val model = ALS.train(ratings, 50, 10, 0.01)**.



```
scala> val model = ALS.train(ratings, 50, 10, 0.01)
17/11/04 07:22:28 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netl
ib.NativeSystemBLAS
17/11/04 07:22:28 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netl
ib.NativeRefBLAS
17/11/04 07:22:29 WARN netlib.LAPACK: Failed to load implementation from: com.github.fommil.ne
tlib.NativeSystemLAPACK
17/11/04 07:22:29 WARN netlib.LAPACK: Failed to load implementation from: com.github.fommil.ne
tlib.NativeRefLAPACK
17/11/04 07:22:33 WARN executor.Executor: 1 block locks were not released by TID = 59:
[rdd_209_0]
17/11/04 07:22:33 WARN executor.Executor: 1 block locks were not released by TID = 60:
[rdd_210_0]
model: org.apache.spark.mllib.recommendation.MatrixFactorizationModel = org.apache.spark.mllib
.recommendation.MatrixFactorizationModel@7d968ec1
```

Figure 15. *Training our ALS model.*

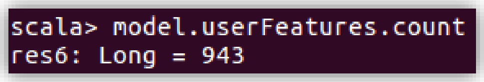
We inspect the user factors by typing **model.userFeatures**.



```
scala> model.userFeatures
res5: org.apache.spark.rdd.RDD[(Int, Array[Double])] = users MapPartitionsRDD[209] at mapValue
s at ALS.scala:269
```

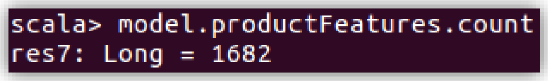
Figure 16. *Inspecting the user factors.*

We count user factors and force computation with **model.userFeatures.count**. We do the same for product factors with **model.productFeatures.count**.



```
scala> model.userFeatures.count
res6: Long = 943
```

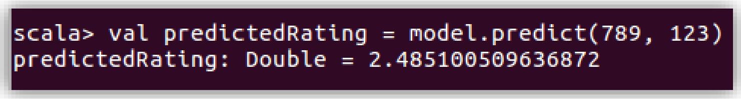
Figure 17. *Counting user features.*



```
scala> model.productFeatures.count
res7: Long = 1682
```

Figure 18. *Counting product features.*

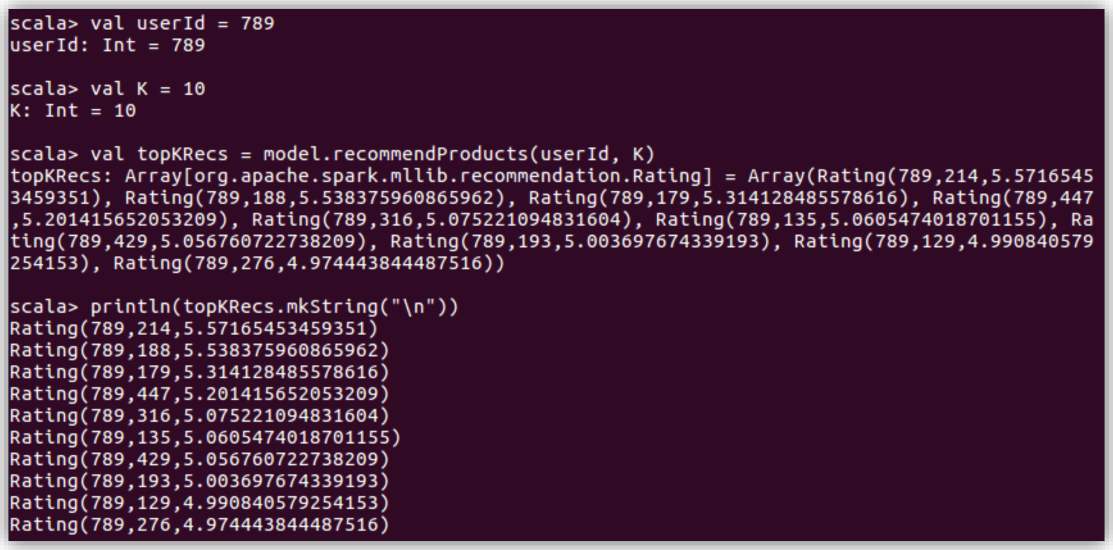
We make a prediction for a single user and movie pair with **val predictedRating = model.predict(789, 123)**.



```
scala> val predictedRating = model.predict(789, 123)
predictedRating: Double = 2.485100509636872
```

Figure 19. *Making a prediction for a single user and movie pair.*

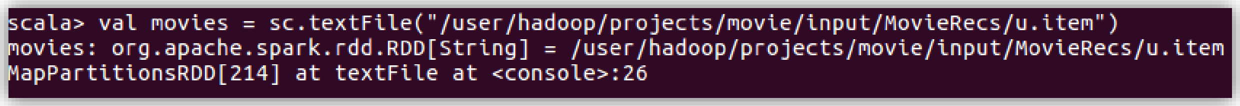
We make predictions for a single user across all movies with the following commands:
val userId = 789
val K = 10
val topKRecs = model.recommendProducts(userId, K)
println(topKRecs.mkString("\n"))



```
scala> val userId = 789
userId: Int = 789

scala> val K = 10
K: Int = 10

scala> val topKRecs = model.recommendProducts(userId, K)
topKRecs: Array[org.apache.spark.mllib.recommendation.Rating] = Array(Rating(789,214,5.5716545
3459351), Rating(789,188,5.538375960865962), Rating(789,179,5.314128485578616), Rating(789,447
,5.201415652053209), Rating(789,316,5.075221094831604), Rating(789,135,5.0605474018701155), Ra
ting(789,429,5.056760722738209), Rating(789,193,5.003697674339193), Rating(789,129,4.990840579
254153), Rating(789,276,4.974443844487516))

scala> println(topKRecs.mkString("\n"))
Rating(789,214,5.57165453459351)
Rating(789,188,5.538375960865962)
Rating(789,179,5.314128485578616)
Rating(789,447,5.201415652053209)
Rating(789,316,5.075221094831604)
Rating(789,135,5.0605474018701155)
Rating(789,429,5.056760722738209)
Rating(789,193,5.003697674339193)
Rating(789,129,4.990840579254153)
Rating(789,276,4.974443844487516)
```

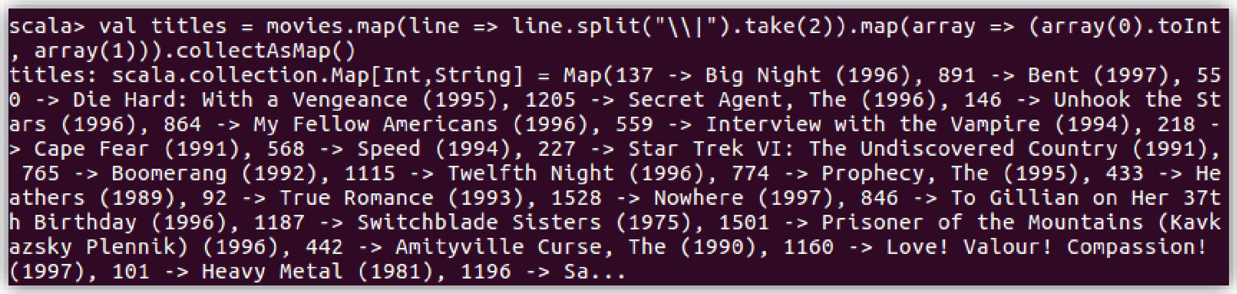
Figure 20. *Making a prediction for a single user across all movies.*

We load the movie titles to inspect the recommendations with **val movies = sc.textFile("/user/hadoop/projects/movie/input/MovieRecs/u.item")**, map the data using **val titles = movies.map(line => line.split("\\|").take(2)).map(array => (array(0).toInt, array(1))).collectAsMap()** and inspect it with **titles(123)**.



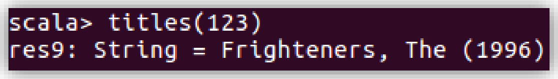
```
scala> val movies = sc.textFile("/user/hadoop/projects/movie/input/MovieRecs/u.item")
movies: org.apache.spark.rdd.RDD[String] = /user/hadoop/projects/movie/input/MovieRecs/u.item
MapPartitionsRDD[214] at textFile at <console>:26
```

Figure 21. *Loading our movie titles.*



```
scala> val titles = movies.map(line => line.split("\\|").take(2)).map(array => (array(0).toInt
, array(1))).collectAsMap()
titles: scala.collection.Map[Int,String] = Map(137 -> Big Night (1996), 891 -> Bent (1997), 55
0 -> Die Hard: With a Vengeance (1995), 1205 -> Secret Agent, The (1996), 146 -> Unhook the St
ars (1996), 864 -> My Fellow Americans (1996), 559 -> Interview with the Vampire (1994), 218 -
> Cape Fear (1991), 568 -> Speed (1994), 227 -> Star Trek VI: The Undiscovered Country (1991),
 765 -> Boomerang (1992), 1115 -> Twelfth Night (1996), 774 -> Prophecy, The (1995), 433 -> He
athers (1989), 92 -> True Romance (1993), 1528 -> Nowhere (1997), 846 -> To Gillian on Her 37t
h Birthday (1996), 1187 -> Switchblade Sisters (1975), 1501 -> Prisoner of the Mountains (Kavk
azsky Plennik) (1996), 442 -> Amityville Curse, The (1990), 1160 -> Love! Valour! Compassion!
(1997), 101 -> Heavy Metal (1981), 1196 -> Sa...
```

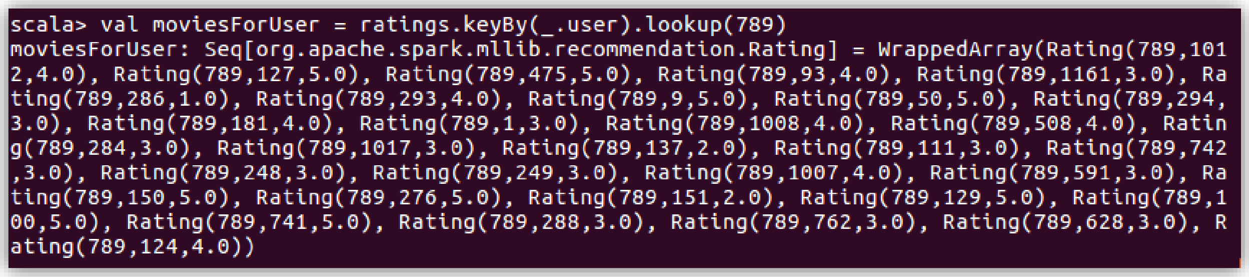
Figure 22. *Mapping our movie titles.*



```
scala> titles(123)
res9: String = Frighteners, The (1996)
```

Figure 23. *Mapping our movie titles.*

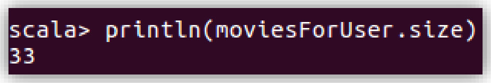
We can inspect the movies rated by user 789 with **val moviesForUser = ratings.keyBy(_.user).lookup(789)**.



```
scala> val moviesForUser = ratings.keyBy(_.user).lookup(789)
moviesForUser: Seq[org.apache.spark.mllib.recommendation.Rating] = WrappedArray(Rating(789,101
2,4.0), Rating(789,127,5.0), Rating(789,475,5.0), Rating(789,93,4.0), Rating(789,1161,3.0), Ra
ting(789,286,1.0), Rating(789,293,4.0), Rating(789,9,5.0), Rating(789,50,5.0), Rating(789,294,
3.0), Rating(789,181,4.0), Rating(789,1,3.0), Rating(789,1008,4.0), Rating(789,508,4.0), Ratin
g(789,284,3.0), Rating(789,1017,3.0), Rating(789,137,2.0), Rating(789,111,3.0), Rating(789,742
,3.0), Rating(789,248,3.0), Rating(789,249,3.0), Rating(789,1007,4.0), Rating(789,591,3.0), Ra
ting(789,150,5.0), Rating(789,276,5.0), Rating(789,151,2.0), Rating(789,129,5.0), Rating(789,1
00,5.0), Rating(789,741,5.0), Rating(789,288,3.0), Rating(789,762,3.0), Rating(789,628,3.0), R
ating(789,124,4.0))
```

Figure 24. *Inspecting the movies rated by user 789.*

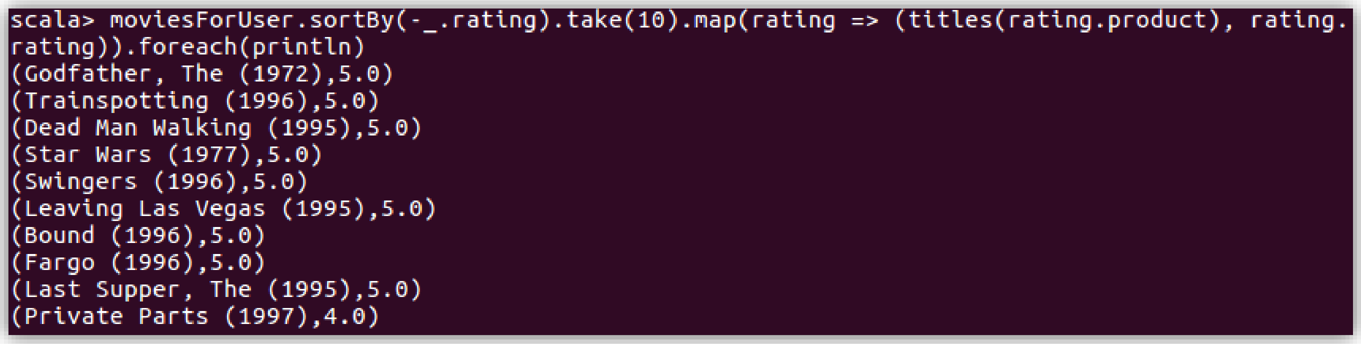
To see how many movies the user has we use **println(moviesForUser.size)**.



```
scala> println(moviesForUser.size)
33
```

Figure 25. *Number of movies reviewed by user 789.*

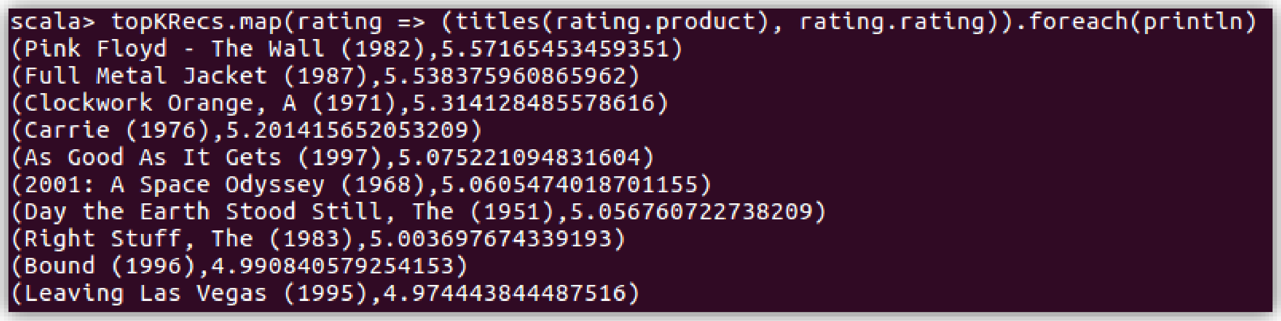
We take the 10 highest rated movies with title from the user with **moviesForUser.sortBy(-_.rating).take(10).map(rating => (titles(rating.product), rating.rating)).foreach(println)**.



```
scala> moviesForUser.sortBy(-_.rating).take(10).map(rating => (titles(rating.product), rating.
rating)).foreach(println)
(Godfather, The (1972),5.0)
(Trainspotting (1996),5.0)
(Dead Man Walking (1995),5.0)
(Star Wars (1977),5.0)
(Swingers (1996),5.0)
(Leaving Las Vegas (1995),5.0)
(Bound (1996),5.0)
(Fargo (1996),5.0)
(Last Supper, The (1995),5.0)
(Private Parts (1997),4.0)
```

Figure 26. *Top 10 rated movies by the user.*

We get the top 10 recommendations for the user with **topKRecs.map(rating => (titles(rating.product), rating.rating)).foreach(println)**.



```
scala> topKRecs.map(rating => (titles(rating.product), rating.rating)).foreach(println)
(Pink Floyd - The Wall (1982),5.57165453459351)
(Full Metal Jacket (1987),5.538375960865962)
(Clockwork Orange, A (1971),5.314128485578616)
(Carrie (1976),5.201415652053209)
(As Good As It Gets (1997),5.075221094831604)
(2001: A Space Odyssey (1968),5.0605474018701155)
(Day the Earth Stood Still, The (1951),5.056760722738209)
(Right Stuff, The (1983),5.003697674339193)
(Bound (1996),4.990840579254153)
(Leaving Las Vegas (1995),4.974443844487516)
```

Figure 27. *Top 10 recommendation for the user.*

PART 2 – MAVEN PROJECT

1. Spark & Scala Installation.
   - We download spark-2.0.0-bin-hadoop2.7.tar.gz from apache mirrors and paste it in /home/hadoop/work folder and extract the file. We download scala-2.11.7.tar.gz from www.scala-lang.org/download/ and paste it in /home/hadoop/work folder and extract the file.

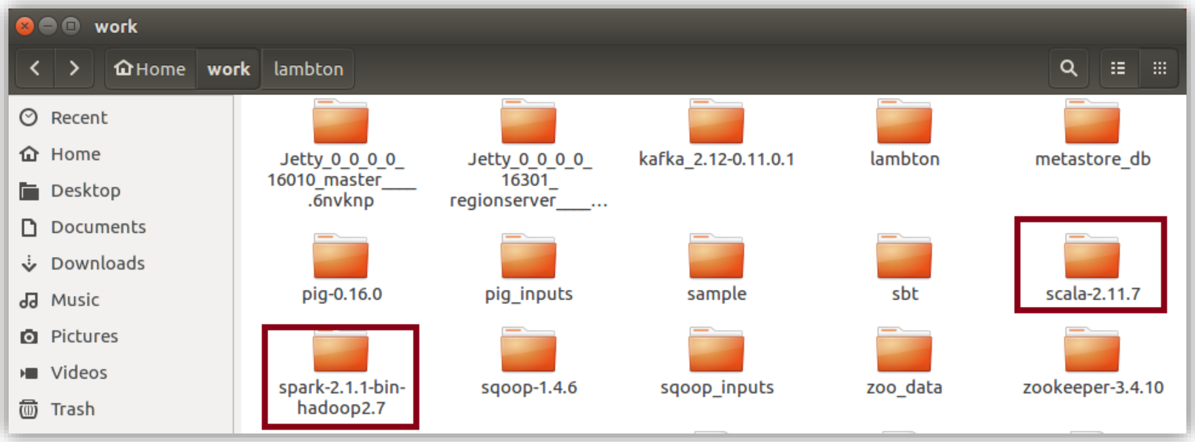
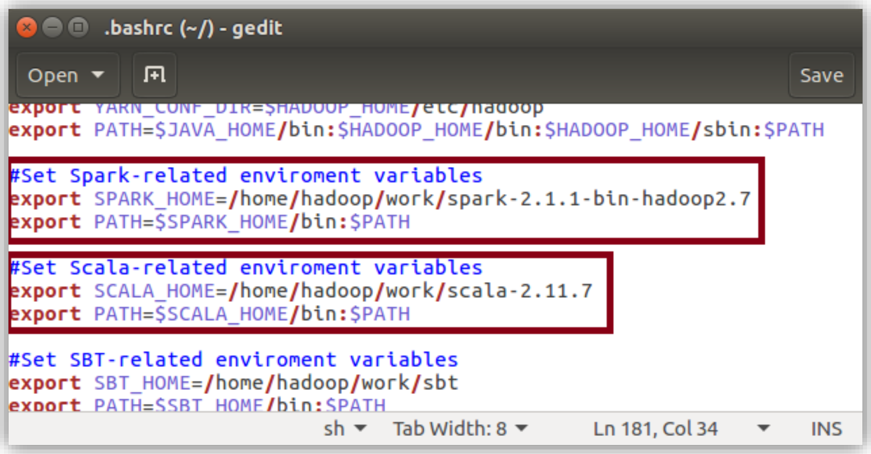


Figure 28. *Extracted Spark and Scala.*

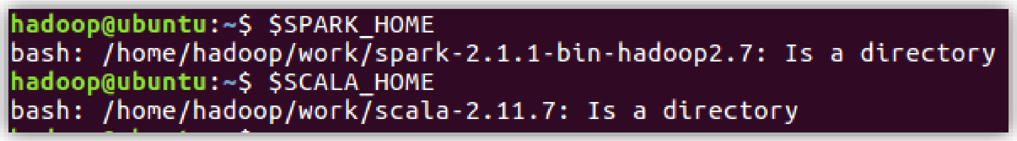
   - We check & set SPARK_HOME and SCALA_HOME in .bashrc file.
     <u>Spark</u>
     export SPARK_HOME=/home/hadoop/work/spark-2.0.0-bin-hadoop2.7
     export PATH=$SPARK_HOME/bin:$PATH
     <u>Scala</u>
     export SCALA_HOME=/home/hadoop/work/scala-2.11.7
     export PATH=$SCALA_HOME/bin:$PATH



```
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH

#Set Spark-related environment variables
export SPARK_HOME=/home/hadoop/work/spark-2.1.1-bin-hadoop2.7
export PATH=$SPARK_HOME/bin:$PATH

#Set Scala-related environment variables
export SCALA_HOME=/home/hadoop/work/scala-2.11.7
export PATH=$SCALA_HOME/bin:$PATH

#Set SBT-related environment variables
export SBT_HOME=/home/hadoop/work/sbt
export PATH=$SBT_HOME/bin:$PATH
```

Figure 29. *Spark and Scala environment variables set.*

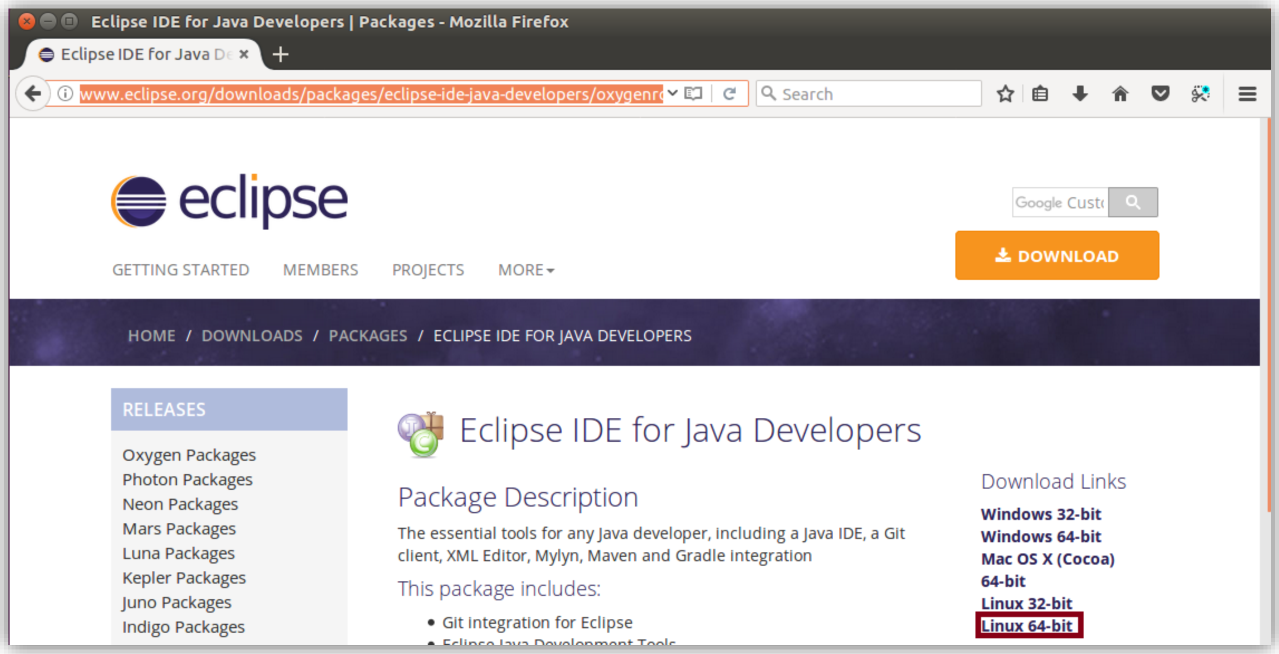
- Type echo $SPARK_HOME and $SCALA_HOME in the terminal to verify.



```
hadoop@ubuntu:~$ $SPARK_HOME
bash: /home/hadoop/work/spark-2.1.1-bin-hadoop2.7: Is a directory
hadoop@ubuntu:~$ $SCALA_HOME
bash: /home/hadoop/work/scala-2.11.7: Is a directory
```

Figure 30. *Verifying installations.*

2. Download Eclipse Oxygen.
We download Eclipse for java developers from http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygenrc3, extract and put in work folder. We run using the eclipse icon.



Eclipse IDE for Java Developers | Packages - Mozilla Firefox
www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygenrc
eclipse
GETTING STARTED
MEMBERS
PROJECTS
MORE
DOWNLOAD
HOME / DOWNLOADS / PACKAGES / ECLIPSE IDE FOR JAVA DEVELOPERS
RELEASES
Oxygen Packages
Photon Packages
Neon Packages
Mars Packages
Luna Packages
Kepler Packages
Juno Packages
Indigo Packages
Eclipse IDE for Java Developers
Package Description
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration
This package includes:
Git integration for Eclipse
Download Links
Windows 32-bit
Windows 64-bit
Mac OS X (Cocoa) 64-bit
Linux 32-bit
Linux 64-bit

Figure 31. *Downloading Eclipse.*

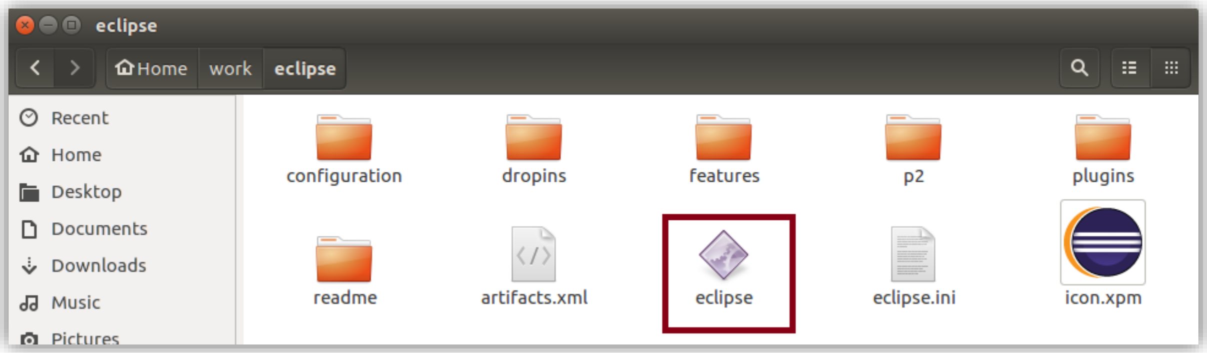


Figure 32. *Eclipse installation.*

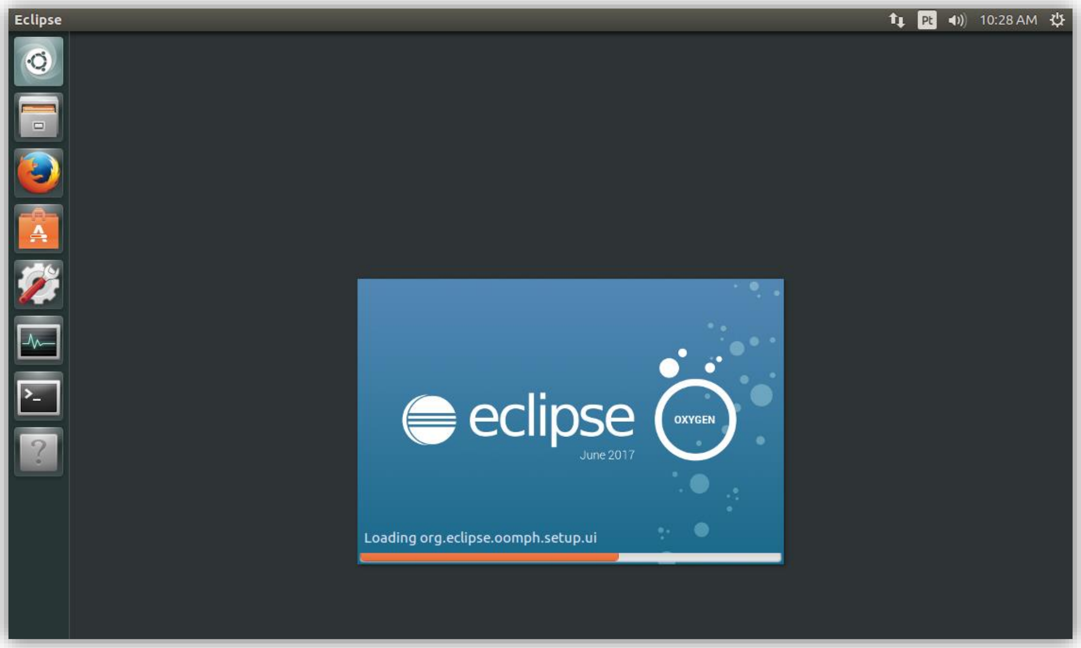


Figure 33. *Running Eclipse.*

3. From Eclipse Marketplace download Scala and Maven and integrate.
   We go to Help, select Eclipse Marketplace and install the following solutions:
   <u>Scala</u>
   - Search: Scala IDE,
   - Install Scala IDE 4.2.x

   <u>Maven</u>
   - Search Maven
   - Install the version for Luna and Newer (we have Oxygen)

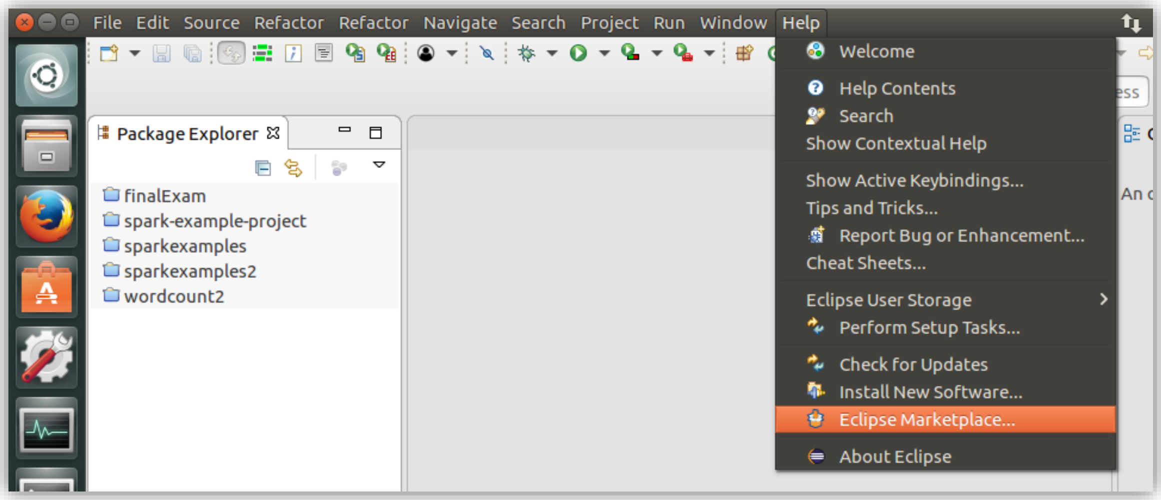
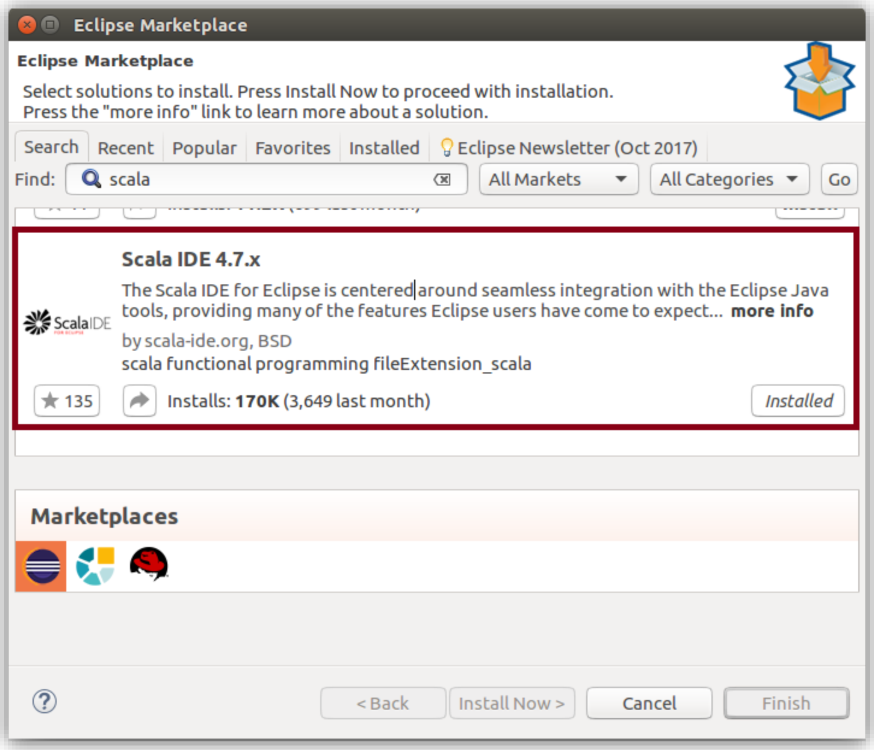


Figure 34. *Accessing Eclipse Marketplace.*


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Find: scala

Scala IDE 4.7.x

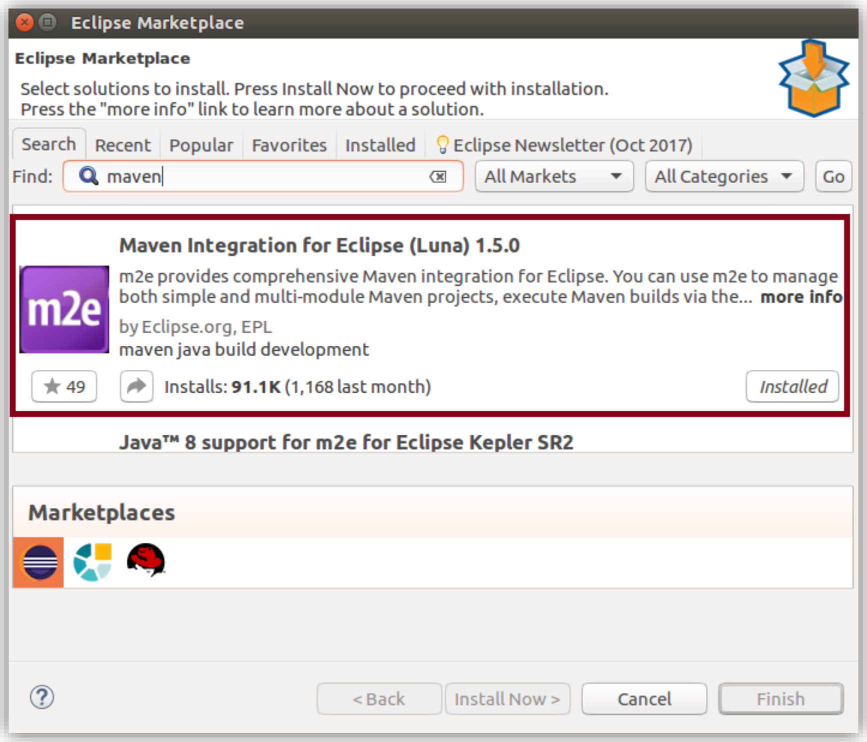
The Scala IDE for Eclipse is centered around seamless integration with the Eclipse Java tools, providing many of the features Eclipse users have come to expect... **more info**

by scala-ide.org, BSD

scala functional programming fileExtension_scala

Installs: **170K** (3,649 last month)

Figure 35. *Scala integration.*


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Find: maven

Maven Integration for Eclipse (Luna) 1.5.0

m2e provides comprehensive Maven integration for Eclipse. You can use m2e to manage both simple and multi-module Maven projects, execute Maven builds via the... **more info**

by Eclipse.org, EPL

maven java build development

Installs: **91.1K** (1,168 last month)

Java™ 8 support for m2e for Eclipse Kepler SR2

Figure 36. *Maven integration.*

4. Create a Maven Project (com.lambton.spark.training)(sparkexamplesmovie).
   Go to File – New – Project
   Select Maven – Maven Project – Next
   Group ID is the package name of Scala one com.lambton.spark.training
   Artifact id Is the project name sparkexamplesmovie

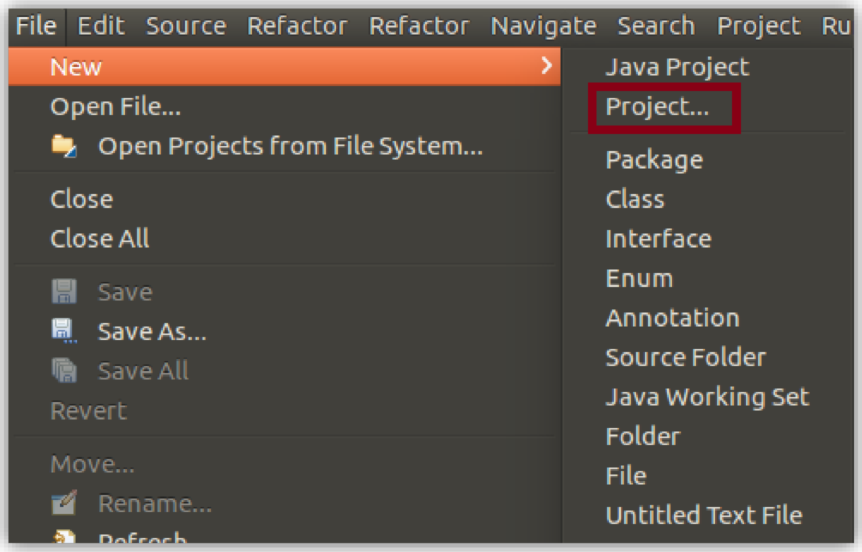


Figure 37. *Creating a new project.*

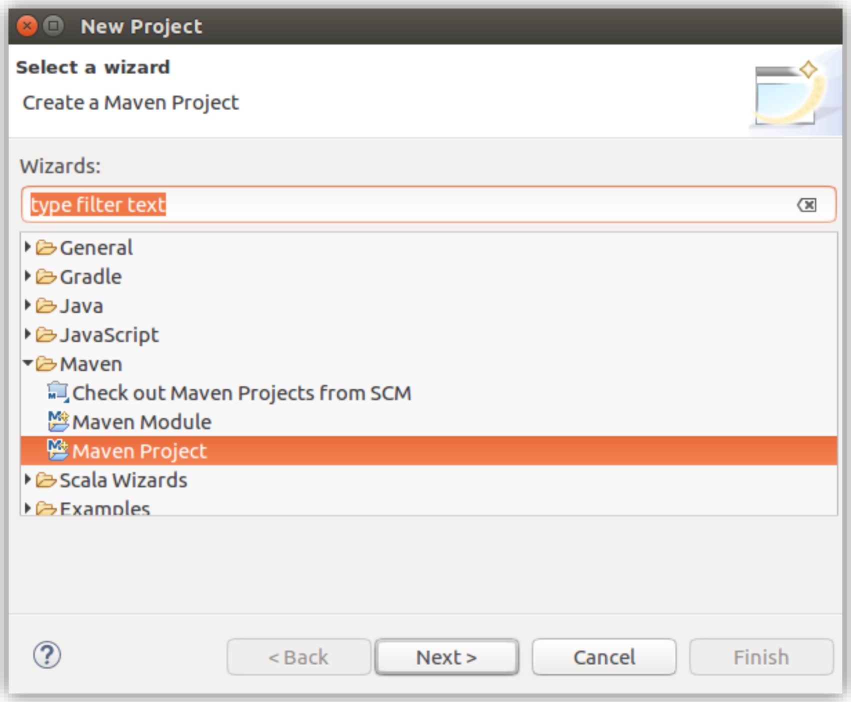
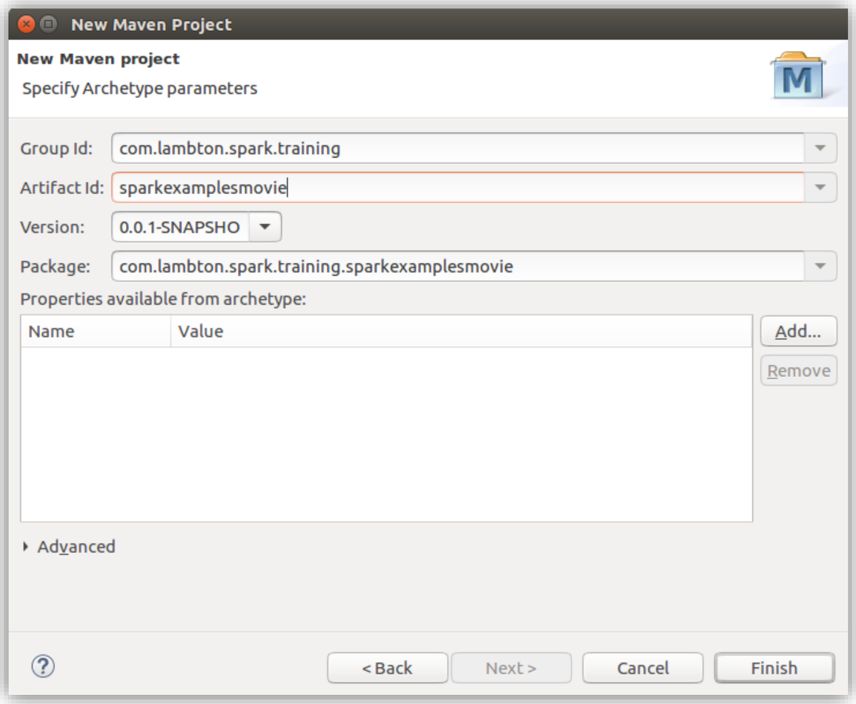


Figure 38. *Selecting type of project.*


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.lambton.spark.training

Artifact Id: sparkexamplesmovie

Version: 0.0.1-SNAPSHO

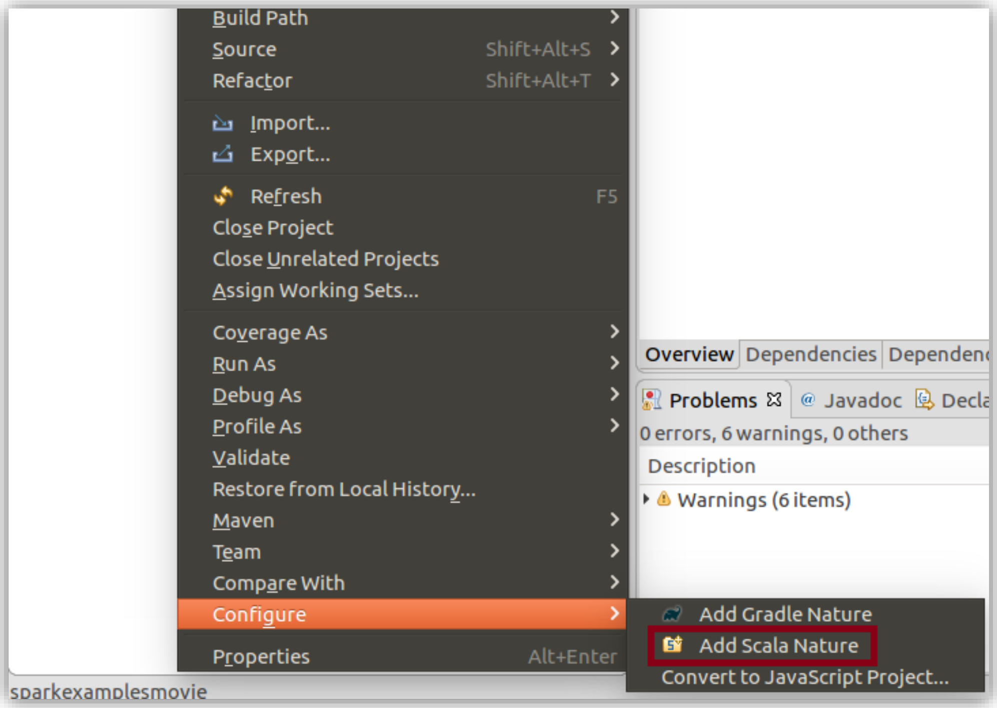
Package: com.lambton.spark.training.sparkexamplesmovie

Properties available from archetype:

Name	Value

Figure 39. *Creating a new Maven Project.*

5. From Configure->Add Scala Nature.

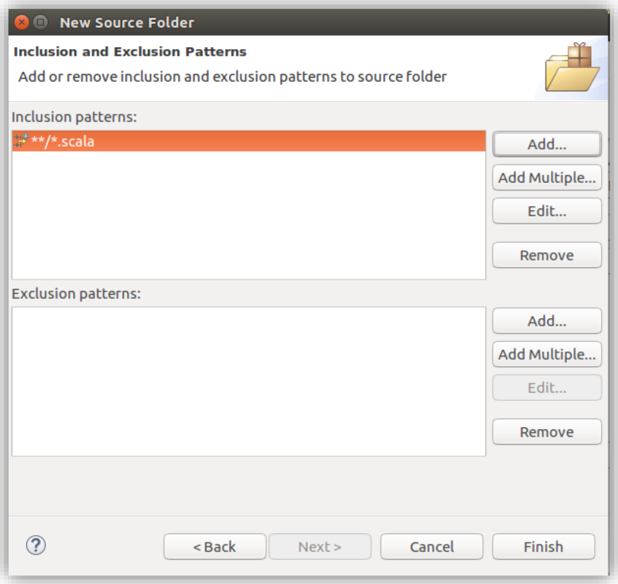


0 errors, 6 warnings, 0 others

Warnings (6 items)

Figure 40. *Adding Scala Nature.*

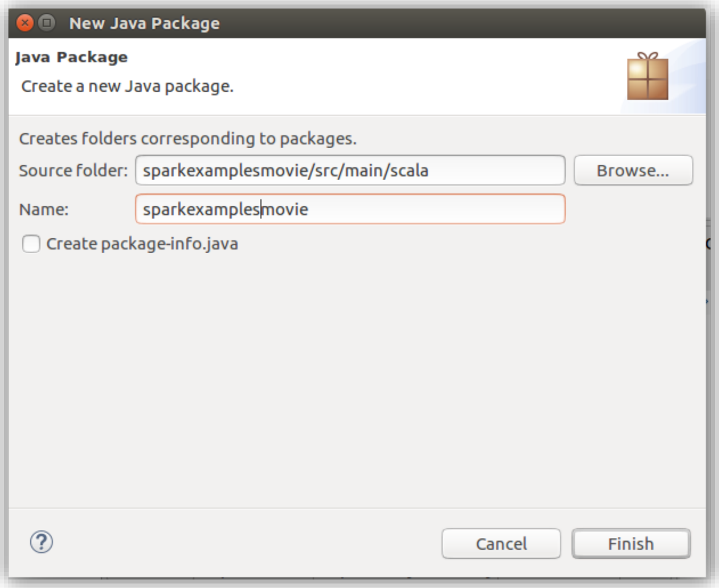
6. From Properties-->java Buildpath-->Add a new folder-->Scala (**/*.scala.
Right click on the project – Build path – Configure build path
In java build path go to source – add folder – add folder – select main and create new folder
Give name scala – In inclusion we have to add the pattern **/*.scala
Click Ok – Apply – Apply and close



**/*.scala

Figure 41. *Adding Scala path.*

7. Create a Package (com.lambton.spark.training.sparkexamples).
Right click on folder scr/main/scala and create new package – to create our scala project



Creates folders corresponding to packages.
Source folder: sparkexamplesmovie/src/main/scala
Name: sparkexamplesmovie
Create package-info.java

Figure 42. *Creating a package.*

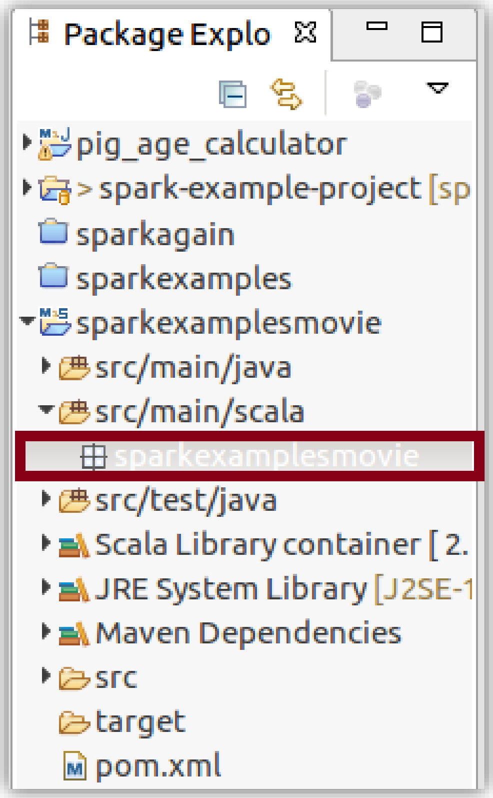


Figure 43. *Package created.*

8. Create a Scala Object (MovieRec).

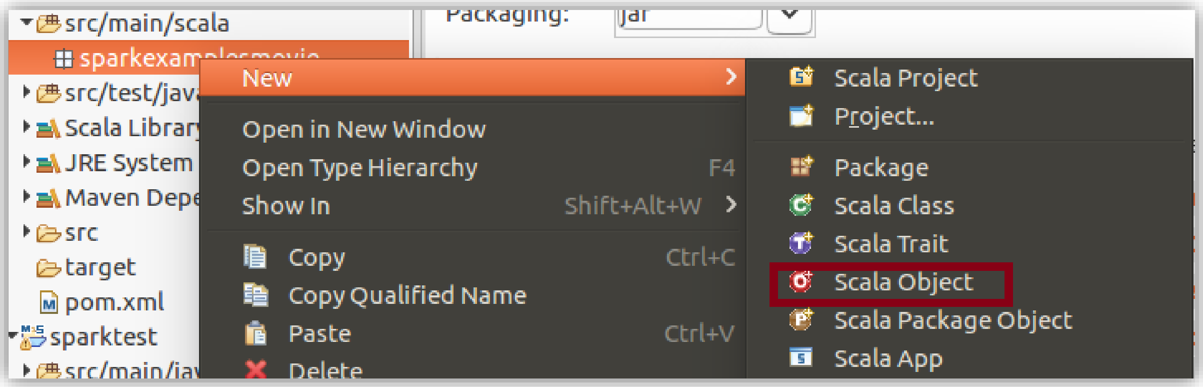
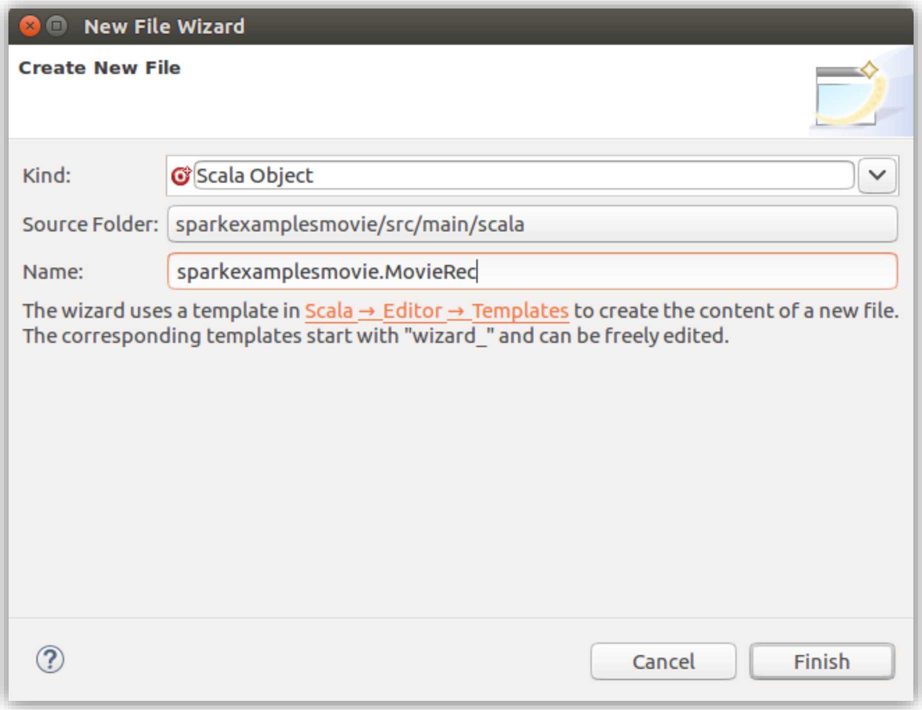


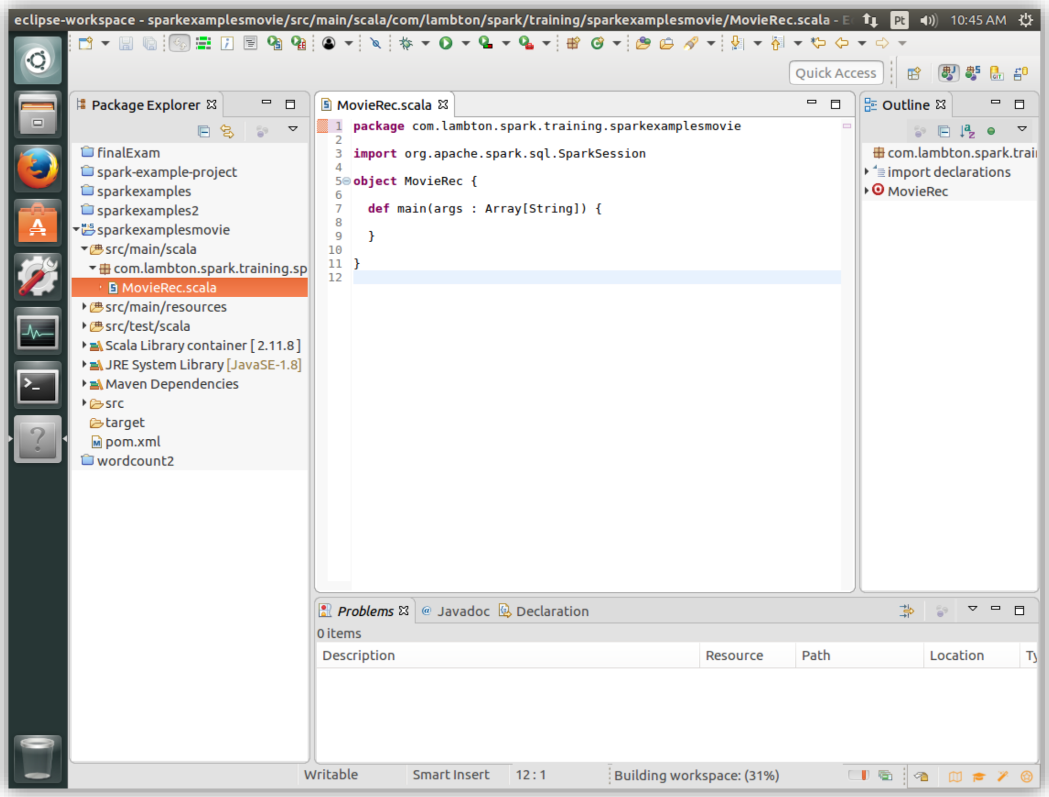
Figure 44. *Selecting Scala Object.*



Kind: Scala Object
Source Folder: sparkexamplesmovie/src/main/scala
Name: sparkexamplesmovie.MovieRec

The wizard uses a template in Scala → Editor → Templates to create the content of a new file. The corresponding templates start with "wizard_" and can be freely edited.

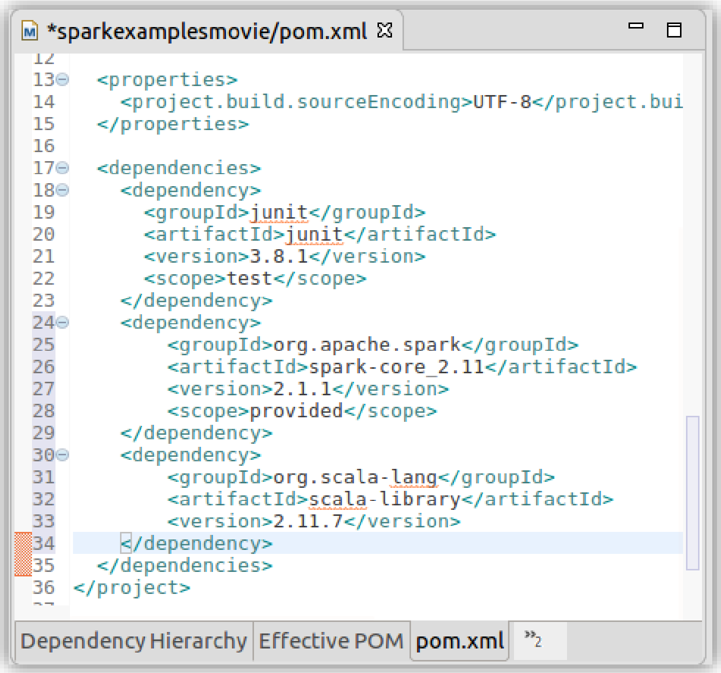
Figure 45. *Creating Scala Object.*



```
package com.lambton.spark.training.sparkexamplesmovie

import org.apache.spark.sql.SparkSession

object MovieRec {

  def main(args : Array[String]) {

  }

}
```

Figure 46. *Scala Object created.*

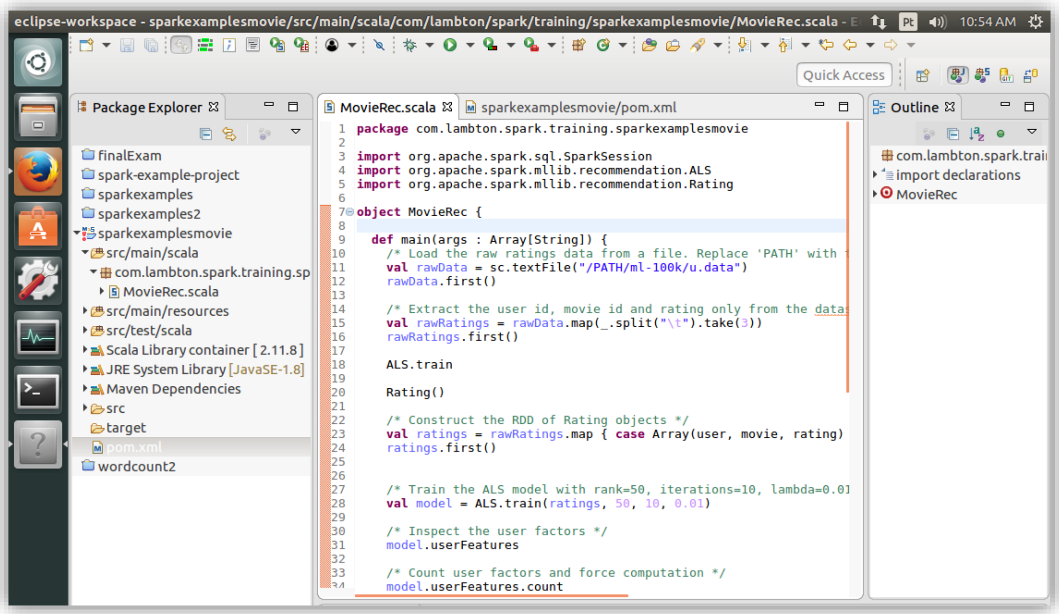
9. Add Pom dependencies for Scala and Spark.
We go to pom.xml and we need to add spark and scala dependencies
We searh for maven dependency spark core 2.1.1 - we copy the code from internet and add dependency
We search for maven scala dependency 2.11.7 - we copy the code from internet and add dependency



```xml
  <properties>
    <project.build.sourceEncoding>UTF-8</project.bui
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.11</artifactId>
        <version>2.1.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>2.11.7</version>
    </dependency>
  </dependencies>
</project>
```

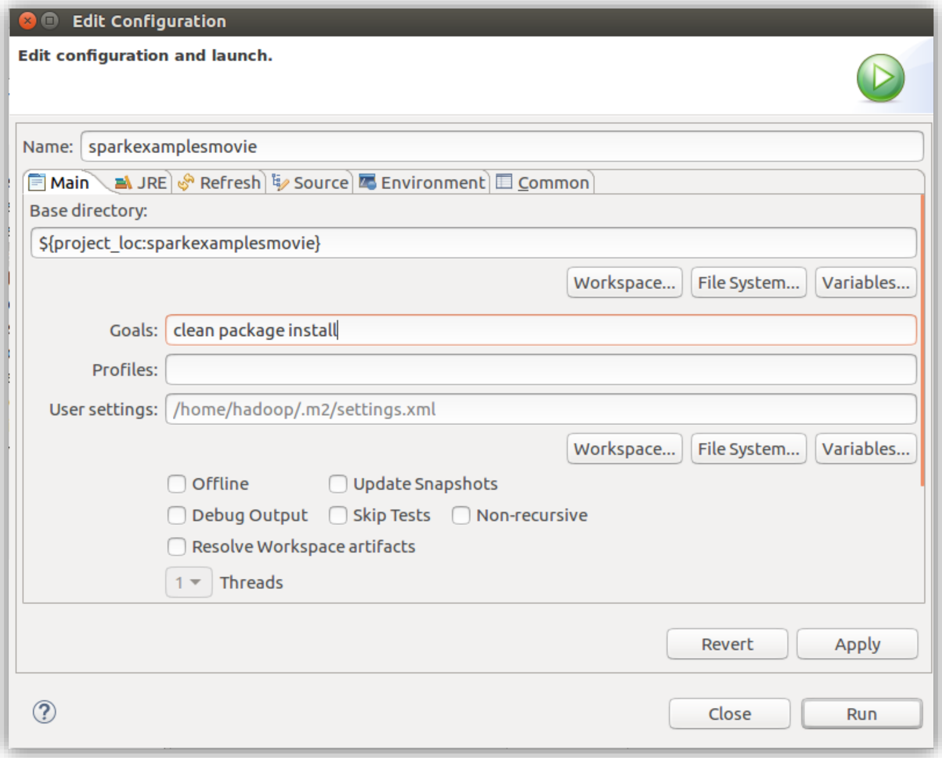
Figure 47. *POM dependencies added for Scala and Spark.*

10. Write a MovieRec Scala Program.



```scala
package com.lambton.spark.training.sparkexamplesmovie

import org.apache.spark.sql.SparkSession
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating

object MovieRec {

  def main(args : Array[String]) {
    /* Load the raw ratings data from a file. Replace 'PATH' with
    val rawData = sc.textFile("/PATH/ml-100k/u.data")
    rawData.first()

    /* Extract the user id, movie id and rating only from the data
    val rawRatings = rawData.map(_.split("\t").take(3))
    rawRatings.first()

    ALS.train

    Rating()

    /* Construct the RDD of Rating objects */
    val ratings = rawRatings.map { case Array(user, movie, rating)
    ratings.first()


    /* Train the ALS model with rank=50, iterations=10, lambda=0.01
    val model = ALS.train(ratings, 50, 10, 0.01)

    /* Inspect the user factors */
    model.userFeatures

    /* Count user factors and force computation */
    model.userFeatures.count
```

Figure 48. *Our MovieRec Scala Program.*


Edit Configuration

Edit configuration and launch.

Name: sparkexamplesmovie

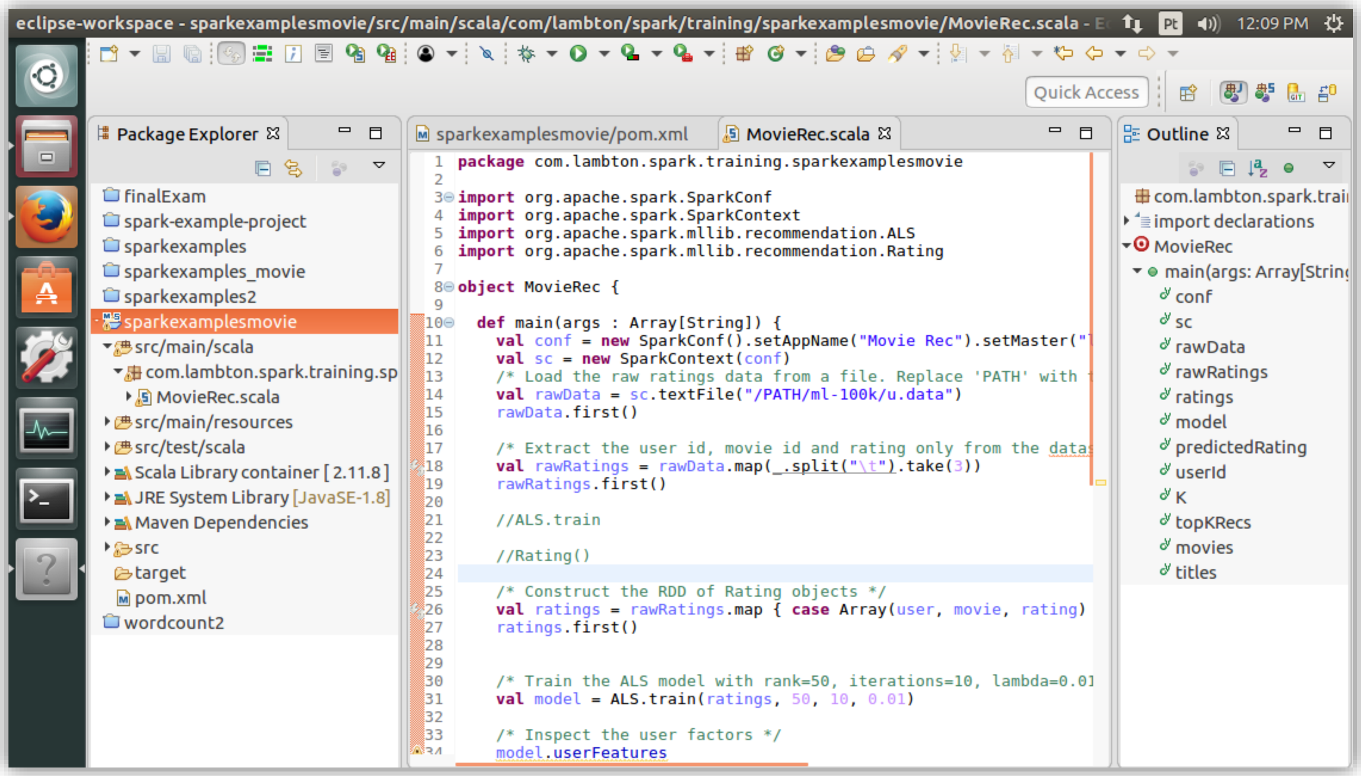
Base directory: ${project_loc:sparkexamplesmovie}

Goals: clean package install

Profiles:

User settings: /home/hadoop/.m2/settings.xml

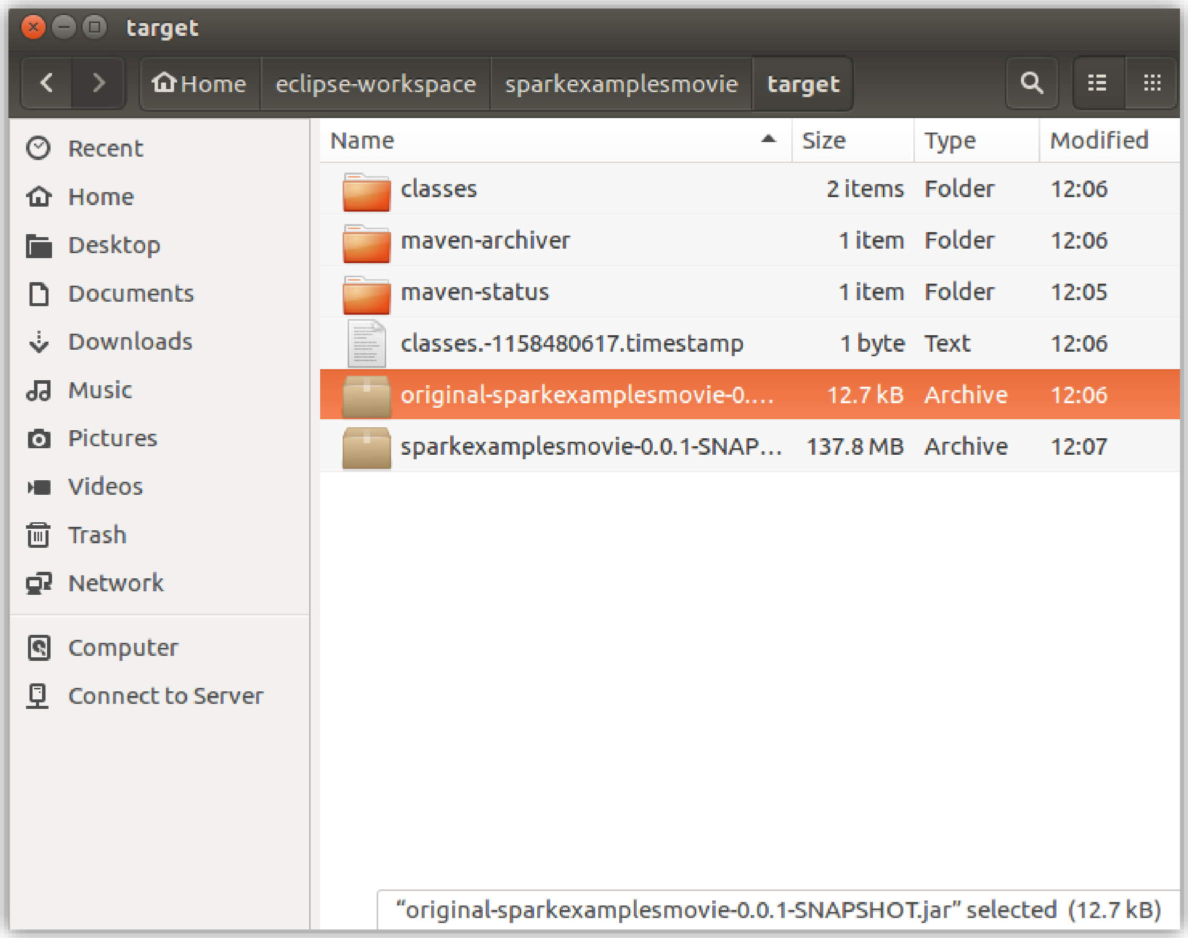
Figure 49. *We launch to test our program.*


eclipse-workspace - sparkexamplesmovie/src/main/scala/com/lambton/spark/training/sparkexamplesmovie/MovieRec.scala

```
package com.lambton.spark.training.sparkexamplesmovie

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating

object MovieRec {

  def main(args : Array[String]) {
    val conf = new SparkConf().setAppName("Movie Rec").setMaster("
    val sc = new SparkContext(conf)
    /* Load the raw ratings data from a file. Replace 'PATH' with 
    val rawData = sc.textFile("/PATH/ml-100k/u.data")
    rawData.first()

    /* Extract the user id, movie id and rating only from the datas
    val rawRatings = rawData.map(_.split("\t").take(3))
    rawRatings.first()

    //ALS.train

    //Rating()

    /* Construct the RDD of Rating objects */
    val ratings = rawRatings.map { case Array(user, movie, rating)
    ratings.first()


    /* Train the ALS model with rank=50, iterations=10, lambda=0.01
    val model = ALS.train(ratings, 50, 10, 0.01)

    /* Inspect the user factors */
    model.userFeatures
```

Figure 50. *We make some modifications to our program.*

11. Create a jar to use on Spark.
Right click on project – Export – click on Java, jar file – Next
We select destination – point it to work folder and give a name for it – Next
Next – Finish – We have built a jar file

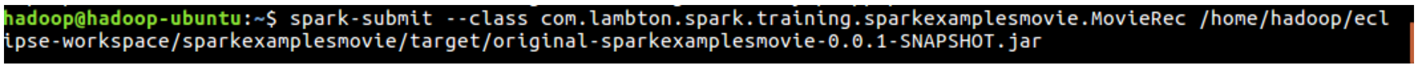


Name	Size	Type	Modified
classes	2 items	Folder	12:06
maven-archiver	1 item	Folder	12:06
maven-status	1 item	Folder	12:05
classes.-1158480617.timestamp	1 byte	Text	12:06
original-sparkexamplesmovie-0.…	12.7 kB	Archive	12:06
sparkexamplesmovie-0.0.1-SNAP…	137.8 MB	Archive	12:07

“original-sparkexamplesmovie-0.0.1-SNAPSHOT.jar” selected (12.7 kB)

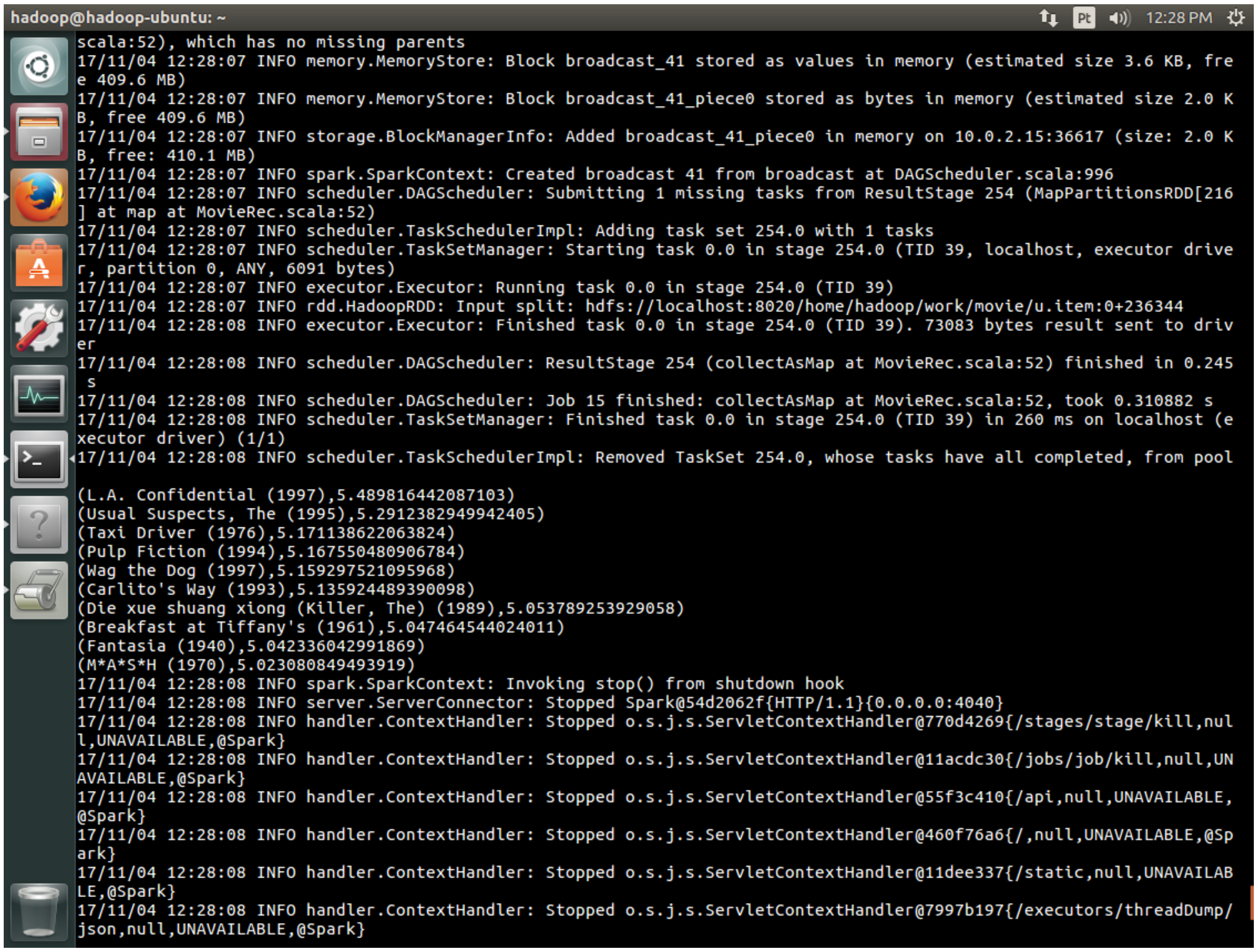
Figure 51. *Jar created.*

12. We run the program in Spark.



```
hadoop@hadoop-ubuntu:~$ spark-submit --class com.lambton.spark.training.sparkexamplesmovie.MovieRec /home/hadoop/ecl
ipse-workspace/sparkexamplesmovie/target/original-sparkexamplesmovie-0.0.1-SNAPSHOT.jar
```

Figure 52. *Submitting our program to Spark.*



```
scala:52), which has no missing parents
17/11/04 12:28:07 INFO memory.MemoryStore: Block broadcast_41 stored as values in memory (estimated size 3.6 KB, fre
e 409.6 MB)
17/11/04 12:28:07 INFO memory.MemoryStore: Block broadcast_41_piece0 stored as bytes in memory (estimated size 2.0 K
B, free 409.6 MB)
17/11/04 12:28:07 INFO storage.BlockManagerInfo: Added broadcast_41_piece0 in memory on 10.0.2.15:36617 (size: 2.0 K
B, free: 410.1 MB)
17/11/04 12:28:07 INFO spark.SparkContext: Created broadcast 41 from broadcast at DAGScheduler.scala:996
17/11/04 12:28:07 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 254 (MapPartitionsRDD[216
] at map at MovieRec.scala:52)
17/11/04 12:28:07 INFO scheduler.TaskSchedulerImpl: Adding task set 254.0 with 1 tasks
17/11/04 12:28:07 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 254.0 (TID 39, localhost, executor drive
r, partition 0, ANY, 6091 bytes)
17/11/04 12:28:07 INFO executor.Executor: Running task 0.0 in stage 254.0 (TID 39)
17/11/04 12:28:07 INFO rdd.HadoopRDD: Input split: hdfs://localhost:8020/home/hadoop/work/movie/u.item:0+236344
17/11/04 12:28:08 INFO executor.Executor: Finished task 0.0 in stage 254.0 (TID 39). 73083 bytes result sent to driv
er
17/11/04 12:28:08 INFO scheduler.DAGScheduler: ResultStage 254 (collectAsMap at MovieRec.scala:52) finished in 0.245
 s
17/11/04 12:28:08 INFO scheduler.DAGScheduler: Job 15 finished: collectAsMap at MovieRec.scala:52, took 0.310882 s
17/11/04 12:28:08 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 254.0 (TID 39) in 260 ms on localhost (e
xecutor driver) (1/1)
17/11/04 12:28:08 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 254.0, whose tasks have all completed, from pool

(L.A. Confidential (1997),5.489816442087103)
(Usual Suspects, The (1995),5.2912382949942405)
(Taxi Driver (1976),5.171138622063824)
(Pulp Fiction (1994),5.167550480906784)
(Wag the Dog (1997),5.159297521095968)
(Carlito's Way (1993),5.135924489390098)
(Die xue shuang xiong (Killer, The) (1989),5.053789253929058)
(Breakfast at Tiffany's (1961),5.047464544024011)
(Fantasia (1940),5.042336042991869)
(M*A*S*H (1970),5.023080849493919)
17/11/04 12:28:08 INFO spark.SparkContext: Invoking stop() from shutdown hook
17/11/04 12:28:08 INFO server.ServerConnector: Stopped Spark@54d2062f{HTTP/1.1}{0.0.0.0:4040}
17/11/04 12:28:08 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@770d4269{/stages/stage/kill,nul
l,UNAVAILABLE,@Spark}
17/11/04 12:28:08 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@11acdc30{/jobs/job/kill,null,UN
AVAILABLE,@Spark}
17/11/04 12:28:08 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@55f3c410{/api,null,UNAVAILABLE,
@Spark}
17/11/04 12:28:08 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@460f76a6{/,null,UNAVAILABLE,@Sp
ark}
17/11/04 12:28:08 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@11dee337{/static,null,UNAVAILAB
LE,@Spark}
17/11/04 12:28:08 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@7997b197{/executors/threadDump/
json,null,UNAVAILABLE,@Spark}
```

Figure 53. *Program's result.*