Zeppelin   Notebook   Interpreter   Connected

EE542 PROJ

default

FINISHED

```
import java.io.File
import scala.io.Source

import org.apache.log4j.Logger
import org.apache.log4j.Level

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.rdd._
import org.apache.spark.mllib.recommendation.{ALS, Rating, MatrixFactorizationModel}
```

```
import java.io.File
import scala.io.Source
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.rdd._
import org.apache.spark.mllib.recommendation.{ALS, Rating, MatrixFactorizationModel}
```

Took 2 seconds.

FINISHED

```
val movieLensHomeDir = "s3://ee542proj/input/"

val movies = sc.textFile(movieLensHomeDir + "movies.dat").map { line =>
  val fields = line.split("::")
  // format: (movieId, movieName)
  (fields(0).toInt, fields(1))
}.collect.toMap

val ratings = sc.textFile(movieLensHomeDir + "ratings.dat").map { line =>
  val fields = line.split("::")
  // format: (timestamp % 10, Rating(userId, movieId, rating))
  (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
}
```

```
movieLensHomeDir: String = s3://ee542proj/
movies: scala.collection.immutable.Map[Int,String] = Map(2163 -> Attack of the Killer Tomat
oes! (1980), 8607 -> Tokyo Godfathers (2003), 645 -> Nelly & Monsieur Arnaud (1995), 42900
-> Cul-de-sac (1966), 892 -> Twelfth Night (1996), 69 -> Friday (1995), 53550 -> Rescue Daw
n (2006), 37830 -> Final Fantasy VII: Advent Children (2004), 5385 -> Last Waltz, The (1978
), 5810 -> 8 Mile (2002), 7375 -> Prince & Me, The (2004), 5659 -> Rocking Horse Winner, Th
e (1950), 2199 -> Phoenix (1998), 8062 -> Dahmer (2002), 3021 -> Funhouse, The (1981), 8536
 -> Intended, The (2002), 5437 -> Manhattan Project, The (1986), 1322 -> Amityville 1992: I
t's About Time (1992), 1665 -> Bean (1997), 5509 -> Biggie and Tupac (2002), 5686 -> Russia
n Ark (Russkiy Kovcheg) (2002), 1036 -> Die Hard (1988), 2822 -> Medi...ratings: org.apache
```

```
.spark.rdd.RDD[(Long, org.apache.spark.mllib.recommendation.Rating)] = MapPartitionsRDD[11]
 at map at <console>:52
```

Took 12 seconds. (outdated)

FINISHED

```
val numRatings = ratings.count
val numUsers = ratings.map(_._2.user).distinct.count
val numMovies = ratings.map(_._2.product).distinct.count

println("Got " + numRatings + " ratings from "
  + numUsers + " users on " + numMovies + " movies.")
```

```
numRatings: Long = 10000054
numUsers: Long = 69878
numMovies: Long = 10677
Got 10000054 ratings from 69878 users on 10677 movies.
```

Took 31 seconds.

FINISHED

```
val training = ratings.filter(x => x._1 < 6)
  .values
  .cache()
val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
  .values
  .cache()
val test = ratings.filter(x => x._1 >= 8).values.cache()

val numTraining = training.count()
val numValidation = validation.count()
val numTest = test.count()

println("Training: " + numTraining + ", validation: " + numValidation + ", test: " +
```

```
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating] = MapParti
tionsRDD[21] at values at <console>:54
validation: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating] = MapPar
titionsRDD[23] at values at <console>:54
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating] = MapPartition
sRDD[25] at values at <console>:53
numTraining: Long = 6002473
numValidation: Long = 1999675
numTest: Long = 1997906
Training: 6002473, validation: 1999675, test: 1997906
```

Took 23 seconds.

FINISHED

```
/** Compute RMSE (Root Mean Squared Error). */
def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating], n: Long): Double = {
    val predictions: RDD[Rating] = model.predict(data.map(x => (x.user, x.product)))
    val predictionsAndRatings = predictions.map(x => ((x.user, x.product), x.rating))
    .join(data.map(x => ((x.user, x.product), x.rating))).values
    math.sqrt(predictionsAndRatings.map(x => (x._1 - x._2) * (x._1 - x._2)).reduce(_ + _)
/ n)
```

```
computeRmse: (model: org.apache.spark.mllib.recommendation.MatrixFactorizationModel, data:
```

```
org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating], n: Long)Double
```

Took 1 seconds.

FINISHED

```
val ranks = List(8, 12)
val lambdas = List(0.1, 10.0)
val numIters = List(10, 20)
var bestModel: Option[MatrixFactorizationModel] = None
var bestValidationRmse = Double.MaxValue
var bestRank = 0
var bestLambda = -1.0
var bestNumIter = -1
for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
  val model = ALS.train(training, rank, numIter, lambda)
  val validationRmse = computeRmse(model, validation, numValidation)
  println("RMSE (validation) = " + validationRmse + " for the model trained with rank = "

    + rank + ", lambda = " + lambda + ", and numIter = " + numIter + ".")
  if (validationRmse < bestValidationRmse) {
    bestModel = Some(model)
    bestValidationRmse = validationRmse
    bestRank = rank
    bestLambda = lambda
    bestNumIter = numIter
  }
```

```
ranks: List[Int] = List(8, 12)
lambdas: List[Double] = List(0.1, 10.0)
numIters: List[Int] = List(10, 20)
bestModel: Option[org.apache.spark.mllib.recommendation.MatrixFactorizationModel] = None
bestValidationRmse: Double = 1.7976931348623157E308
bestRank: Int = 0
bestLambda: Double = -1.0
bestNumIter: Int = -1
RMSE (validation) = 0.8236100381645034 for the model trained with rank = 8, lambda = 0.1, a
nd numIter = 10.
RMSE (validation) = 0.8189231812579648 for the model trained with rank = 8, lambda = 0.1, a
nd numIter = 20.
RMSE (validation) = 3.667982949261605 for the model trained with rank = 8, lambda = 10.0, a
nd numIter = 10.
RMSE (validation) = 3.667982949261605 for the model trained with rank = 8, lambda = 10.0, a
nd numIter = 20.
RMSE (validation) = 0.8192026516236711 for the model trained with rank = 12, lambda = 0.1,
and numIter = 10.
RMSE (validation) = 0.8154603688946302 for the model trained with rank = 12, lambda = 0.1,
and numIter = 20.
RMSE (validation) = 3.667982949261605 for the model trained with rank = 12, lambda = 10.0,
and numIter = 10.
RMSE (validation) = 3.667982949261605 for the model trained with rank = 12, lambda = 10.0,
and numIter = 20.
```

Took 459 seconds.

FINISHED

```
// evaluate the best model on the test set
```

```
val testRmse = computeRmse(bestModel.get, test, numTest)
```

```
testRmse: Double = 0.8155943302828965
The best model was trained with rank = 12 and lambda = 0.1, and numIter = 20, and its RMSE on the test set is 0.8155943302828965.
```

Took 16 seconds.

FINISHED

```
// create a naive baseline and compare it with the best model
val meanRating = training.union(validation).map(_.rating).mean
val baselineRmse =
  math.sqrt(test.map(x => (meanRating - x.rating) * (meanRating - x.rating)).mean)
val improvement = (baselineRmse - testRmse) / baselineRmse * 100
println("The best model improves the baseline by " + "%1.2f".format(improvement) + "%.")
```

```
meanRating: Double = 3.5123623057208624
baselineRmse: Double = 1.0597828264660583
improvement: Double = 23.041371315426044
The best model improves the baseline by 23.04%.
```

Took 3 seconds.

FINISHED

```
val candidates = sc.parallelize(movies.keys.toSeq)
val recommendations = bestModel.get
  .predict(candidates.map((100, _)))
  .collect()
  .sortBy(- _.rating)
  .take(10)

var i = 1
println("Movies recommended for you:")
recommendations.foreach { r =>
  println("%2d".format(i) + ": " + movies(r.product))
  i += 1
}
```

```
candidates: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2611] at parallelize at <console>:53
recommendations: Array[org.apache.spark.mllib.recommendation.Rating] = Array(Rating(100,60983,4.321957638468641), Rating(100,61742,3.802899742530066), Rating(100,42783,3.7482030276395695), Rating(100,53883,3.729625669769619), Rating(100,32090,3.6831572695067334), Rating(100,60291,3.5688352736060587), Rating(100,296,3.526813877388885), Rating(100,64280,3.5211789737280452), Rating(100,858,3.5161860620303176), Rating(100,1221,3.5152962996050747))
i: Int = 1
Movies recommended for you:
 1: Eve and the Fire Horse (2005)
 2: Maradona by Kusturica (2008)
 3: Shadows of Forgotten Ancestors (1964)
 4: Power of Nightmares: The Rise of the Politics of Fear, The (2004)
 5: Low Life, The (1995)
 6: Gonzo: The Life and Work of Dr. Hunter S. Thompson (2008)
 7: Pulp Fiction (1994)
 8: Hospital (1970)
 9: Godfather, The (1972)
```

```
10: Godfather: Part II, The (1974)
```

Took 3 seconds.

FINISHED

```
val moviesWithGenres = sc.textFile(movieLensHomeDir + "movies.dat").map { line =>
  val fields = line.split("::")
  // format: (movieId, movieName, genre information)
  (fields(0).toInt, fields(2))
}.collect.toMap
```

```
moviesWithGenres: scala.collection.immutable.Map[Int,String] = Map(2163 -> Comedy|Horror, 8
607 -> Adventure|Animation|Drama, 645 -> Drama, 42900 -> Comedy|Crime|Drama|Thriller, 892 -
> Comedy|Drama|Romance, 69 -> Comedy, 53550 -> Action|Adventure|Drama|War, 37830 -> Action|
Adventure|Animation|Fantasy|Sci-Fi, 5385 -> Documentary, 5810 -> Drama, 7375 -> Comedy|Roma
nce, 5659 -> Drama|Horror, 2199 -> Crime|Drama, 8062 -> Drama|Horror|Thriller, 3021 -> Horr
or, 8536 -> Drama|Thriller, 5437 -> Comedy|Thriller, 1322 -> Horror, 1665 -> Comedy, 5509 -
> Documentary, 5686 -> Drama|Fantasy|War, 1036 -> Action|Crime|Thriller, 2822 -> Adventure|
Romance, 7304 -> Animation|Comedy|Fantasy|Musical, 54999 -> Action|Adventure|Thriller, 2630
 -> Drama, 6085 -> Comedy|Drama, 3873 -> Comedy|Western, 4188 -> Chil...
```

Took 1 seconds.

FINISHED

```
val comedyMovies = moviesWithGenres.filter(_._2.matches(".*Comedy.*")).keys
val candidates = sc.parallelize(comedyMovies.toSeq)
val recommendations = bestModel.get
  .predict(candidates.map((100, _)))
  .collect()
  .sortBy(- _.rating)
  .take(5)

var i = 1
println("Comedy Movies recommended for you:")
recommendations.foreach { r =>
  println("%2d".format(i) + ": " + movies(r.product))
  i += 1
}
```

```
comedyMovies: Iterable[Int] = Set(2163, 42900, 892, 69, 7375, 5437, 1665, 7304, 6085, 3873,
 26413, 4201, 4447, 33004, 3962, 5422, 5469, 3944, 6387, 3883, 62851, 5116, 4094, 6167, 508
8, 2889, 59858, 2295, 2306, 4571, 5857, 4464, 101, 2109, 1454, 4909, 2031, 5896, 59625, 207
2, 8663, 4062, 3399, 54256, 33675, 6544, 4169, 4899, 53578, 6712, 55020, 5950, 3167, 31160,
 4183, 909, 4290, 3477, 333, 3979, 2463, 3397, 49110, 3581, 8784, 3830, 6317, 518, 7990, 24
99, 8843, 1083, 468, 54193, 5205, 6172, 4015, 26842, 234, 6690, 2331, 3566, 4728, 6954, 487
7, 6014, 5582, 4992, 5131, 6374, 88, 50354, 47047, 32289, 352, 53993, 33145, 1855, 45722, 5
454, 56176, 1211, 3990, 7888, 4714, 1158, 582, 762, 3072, 8883, 1005, 5141, 115, 6944, 3317
, 5168, 4500, 65027, 7409, 5718, 34018, 37384, 46976, 276, 2622, 4402...candidates: org.apa
che.spark.rdd.RDD[Int] = ParallelCollectionRDD[2715] at parallelize at <console>:55
recommendations: Array[org.apache.spark.mllib.recommendation.Rating] = Array(Rating(100,296
,3.526813877388885), Rating(100,3030,3.331894391300624), Rating(100,750,3.2978257339389936)
, Rating(100,50949,3.248493364773723), Rating(100,6119,3.2032888900258034))
i: Int = 1
Comedy Movies recommended for you:
 1: Pulp Fiction (1994)
 2: Yojimbo (1961)
 3: Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
```

```
 4: Mafioso (1962)
 5: Père Noël est une Ordure, Le (1982)
```
Took 2 seconds.

FINISHED

```
val comedyMovies = moviesWithGenres.filter(_._2.matches(".*Action.*")).keys
val candidates = sc.parallelize(comedyMovies.toSeq)
val recommendations = bestModel.get
  .predict(candidates.map((100, _)))
  .collect()
  .sortBy(- _.rating)
  .take(5)

var i = 1
println("Action Movies recommended for you:")
recommendations.foreach { r =>
  println("%2d".format(i) + ": " + movies(r.product))
  i += 1
}
```

```
comedyMovies: Iterable[Int] = Set(53550, 37830, 1036, 54999, 1586, 26413, 809, 7373, 7766,
58627, 2094, 5469, 6387, 7272, 1168, 4005, 63433, 7569, 4262, 7445, 54256, 479, 51077, 3434
, 2412, 6283, 3698, 54686, 7143, 6317, 2427, 50147, 36519, 34645, 26842, 6448, 5999, 27611,
 7072, 8646, 4682, 56167, 555, 6566, 6014, 1110, 4166, 6808, 2363, 1200, 45722, 6057, 170,
5898, 3681, 6587, 7040, 26746, 46335, 6576, 6177, 7164, 7409, 4339, 1882, 2808, 31553, 5069
, 6219, 2527, 3153, 8811, 27828, 6900, 2947, 3959, 8118, 33672, 49651, 1544, 48319, 51935,
1591, 379, 511, 5691, 4614, 64508, 8045, 26152, 861, 1497, 10, 4543, 1788, 61132, 8733, 560
67, 59840, 1608, 3439, 6764, 56921, 384, 3745, 64231, 26950, 7573, 533, 4011, 4026, 1867, 4
528, 1275, 4638, 7345, 1233, 3781, 4440, 4564, 6078, 8370, 2476, 6996...candidates: org.apa
che.spark.rdd.RDD[Int] = ParallelCollectionRDD[2775] at parallelize at <console>:55
recommendations: Array[org.apache.spark.mllib.recommendation.Rating] = Array(Rating(100,575
1,3.482584374718392), Rating(100,27376,3.4506631353117534), Rating(100,1208,3.3443345063216
37), Rating(100,3030,3.331894391300624), Rating(100,2019,3.3181241628444287))
i: Int = 1
Action Movies recommended for you:
 1: Goodbye Pork Pie (1981)
 2: Tunnel, The (Der Tunnel) (2001)
 3: Apocalypse Now (1979)
 4: Yojimbo (1961)
 5: Seven Samurai (Shichinin no samurai) (1954)
```
Took 3 seconds.

FINISHED

```
val comedyMovies = moviesWithGenres.filter(_._2.matches(".*Adventure.*")).keys
val candidates = sc.parallelize(comedyMovies.toSeq)
val recommendations = bestModel.get
  .predict(candidates.map((100, _)))
  .collect()
  .sortBy(- _.rating)
  .take(5)

var i = 1
println("Adventure Movies recommended for you:")
recommendations.foreach { r =>
  println("%2d".format(i) + ": " + movies(r.product))
```

```
  i += 1
}
```

```
comedyMovies: Iterable[Int] = Set(8607, 53550, 37830, 2822, 54999, 809, 7373, 33004, 2094,
6405, 1168, 4571, 3345, 4005, 101, 63433, 7569, 3930, 4899, 26085, 3698, 1031, 54686, 1899,
 7143, 6317, 6512, 6448, 27611, 941, 3927, 6986, 6566, 31934, 6527, 5582, 3172, 6808, 45722
, 2077, 1750, 170, 5898, 52328, 8883, 8450, 8723, 7164, 4339, 6162, 44022, 6106, 31553, 506
9, 50601, 40815, 3417, 3285, 3153, 8811, 6458, 8682, 2947, 3959, 1544, 48319, 2099, 1591, 7
332, 58105, 26152, 10, 4543, 5967, 61132, 6764, 56, 5361, 6401, 3745, 8635, 4575, 7573, 533
, 3332, 1867, 3175, 47124, 1275, 2141, 4638, 7345, 26483, 51575, 26792, 40339, 31617, 340,
153, 4980, 5192, 4941, 1196, 52287, 3629, 5227, 2723, 4142, 2046, 8898, 1127, 54001, 45431,
 709, 33558, 34520, 1359, 2173, 8678, 1967, 4467, 2405, 63853, 1254, ...candidates: org.apa
che.spark.rdd.RDD[Int] = ParallelCollectionRDD[2765] at parallelize at <console>:55
recommendations: Array[org.apache.spark.mllib.recommendation.Rating] = Array(Rating(100,575
1,3.482584374718392), Rating(100,1201,3.2594467304885155), Rating(100,25798,3.2484933647737
23), Rating(100,4993,3.2462897675533324), Rating(100,7153,3.239427986275163))
i: Int = 1
Adventure Movies recommended for you:
 1: Goodbye Pork Pie (1981)
 2: Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)
 3: Island of Lost Souls (1932)
 4: Lord of the Rings: The Fellowship of the Ring, The (2001)
 5: Lord of the Rings: The Return of the King, The (2003)
```

Took 2 seconds.

```
val comedyMovies = moviesWithGenres.filter(_._2.matches(".*Action|Romance.*")).keys
val candidates = sc.parallelize(comedyMovies.toSeq)
val recommendations = bestModel.get
  .predict(candidates.map((100, _)))
  .collect()
  .sortBy(- _.rating)
  .take(5)

var i = 1
println("Action and Romance Movies recommended for you:")
recommendations.foreach { r =>
  println("%2d".format(i) + ": " + movies(r.product))
  i += 1
}
```

FINISHED

```
comedyMovies: Iterable[Int] = Set(58627, 1110, 6177, 8118, 4614, 1497, 1137, 5316, 3414, 88
16, 1475, 4637, 3442, 4866, 6095, 6723, 2708, 4568, 6130, 638, 1514, 1398, 34538, 26696, 63
276, 7268, 1714, 5084, 4001, 623, 7376, 5156, 821, 7899, 2833, 64999, 2965, 4892, 4853, 419
9, 4531, 6588, 2737, 894, 1666, 5826, 6163, 4947, 6556, 1159, 43987, 2157, 980, 4636, 9, 88
00, 1493, 1424, 2196, 3541, 2497, 1599, 2756, 1434, 1170, 1071, 6192, 36392, 4438, 4503, 32
83, 7704, 1520, 2817, 204, 71, 3796, 1669, 4099, 4764, 5580, 5212, 1574, 64368, 64997, 6280
3, 4569, 7892, 251, 964, 932, 983, 8131, 32617, 1477, 1102, 3444, 4441, 6417, 3206, 64030,
145, 3376, 4035, 6472, 4950, 4200, 4630, 4092, 1658, 3769, 4106, 1749, 7192, 4651, 976, 593
4, 4738, 4542, 62334, 3368, 5922, 63647, 667, 5409, 4387, 2258, 2534,...candidates: org.apa
che.spark.rdd.RDD[Int] = ParallelCollectionRDD[2785] at parallelize at <console>:55
recommendations: Array[org.apache.spark.mllib.recommendation.Rating] = Array(Rating(100,636
47,3.309619538780511), Rating(100,62803,3.106653546147294), Rating(100,6082,2.7761122259552
766), Rating(100,4438,2.7225535165937966), Rating(100,6192,2.7211811068702954))
i: Int = 1
```

```
Action and Romance Movies recommended for you:
 1: Hanzo the Razor: Sword of Justice (Goyôkiba) (1972)
 2: Lone Wolf and Cub: Baby Cart in Peril (Kozure Ôkami: Oya no kokoro ko no kokoro) (1972)
 3: Grey Fox, The (1982)
 4: Chinese Connection, The (a.k.a. Fist of Fury) (Jing wu men) (1972)
 5: Open Hearts (Elsker dig for evigt) (2002)
```

Took 1 seconds.

FINISHED

```
// Save and load model
bestModel.get.save(sc, "s3://ee542proj/movieLens/model/recommendation")
val sameModel = MatrixFactorizationModel.load(sc,  "s3://ee542proj/movieLens/model
```

```
sameModel: org.apache.spark.mllib.recommendation.MatrixFactorizationModel = org.apache.spar
k.mllib.recommendation.MatrixFactorizationModel@4a3c98d1
```

Took 14 seconds.

READY