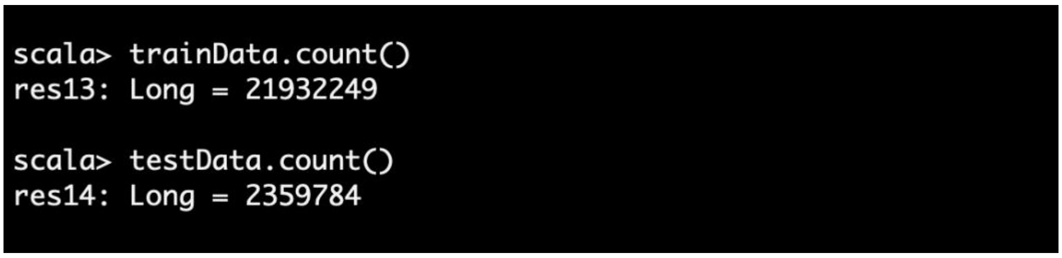
3.Recommender Systems via Matrix Factorization

Problem 3. (a) (i)

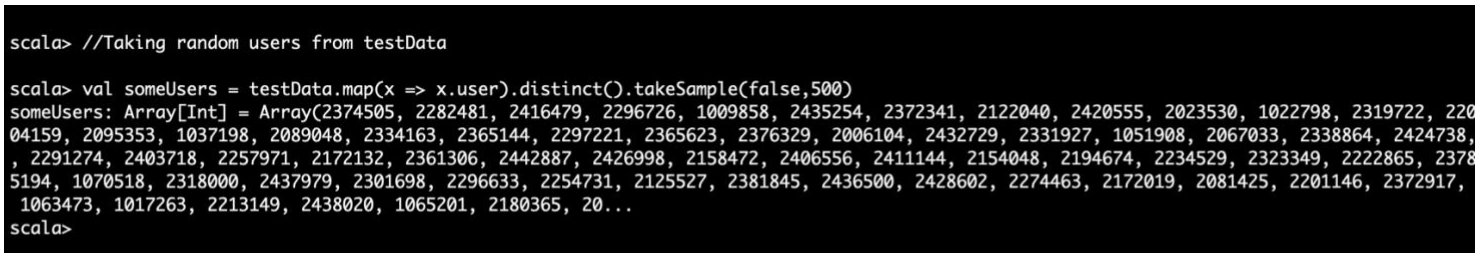
- For each user, 90% of the artists this user has Listened has put into trainData
- Remaining 10% has been put into testData



```
scala> trainData.count()
res13: Long = 21932249

scala> testData.count()
res14: Long = 2359784
```

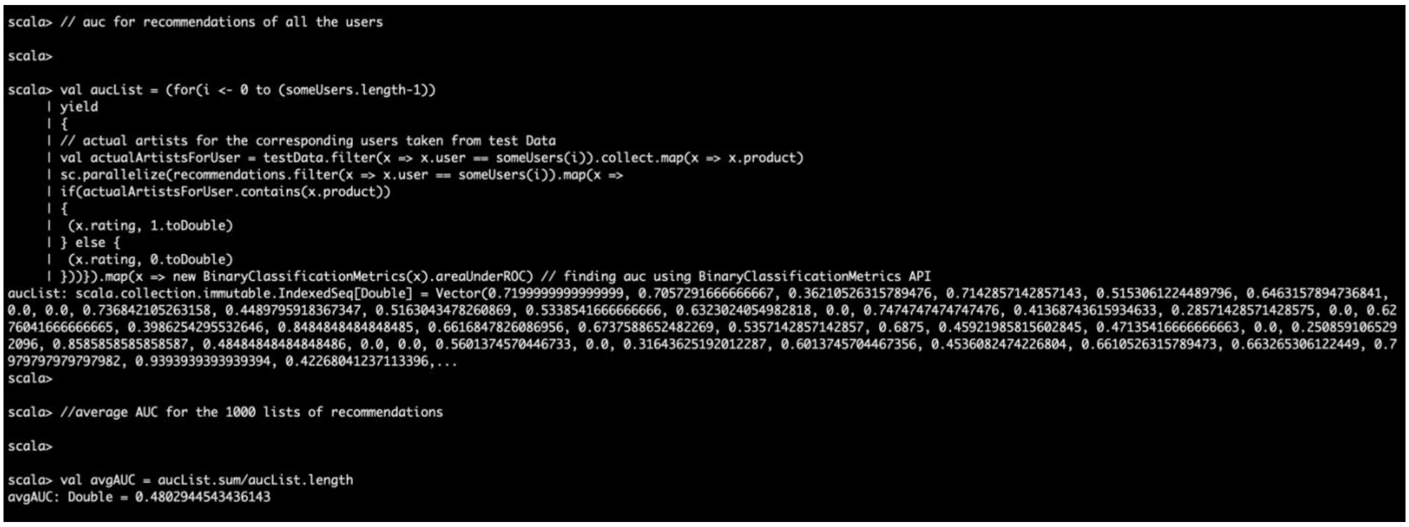
Problem 3(a)(ii)

We have taken 500 users from test Data since srun is taking very long time to run.



```
scala> //Taking random users from testData

scala> val someUsers = testData.map(x => x.user).distinct().takeSample(false,500)
someUsers: Array[Int] = Array(2374505, 2282481, 2416479, 2296726, 1009858, 2435254, 2372341, 2122040, 2420555, 2023530, 1022798, 2319722, 220
04159, 2095353, 1037198, 2089048, 2334163, 2365144, 2297221, 2365623, 2376329, 2006104, 2432729, 2331927, 1051908, 2067033, 2338864, 2424738,
, 2291274, 2403718, 2257971, 2172132, 2361306, 2442887, 2426998, 2158472, 2406556, 2411144, 2154048, 2194674, 2234529, 2323349, 2222865, 2378
5194, 1070518, 2318000, 2437979, 2301698, 2296633, 2254731, 2125527, 2381845, 2436500, 2428602, 2274463, 2172019, 2081425, 2201146, 2372917,
 1063473, 1017263, 2213149, 2438020, 1065201, 2180365, 20...
scala>
```

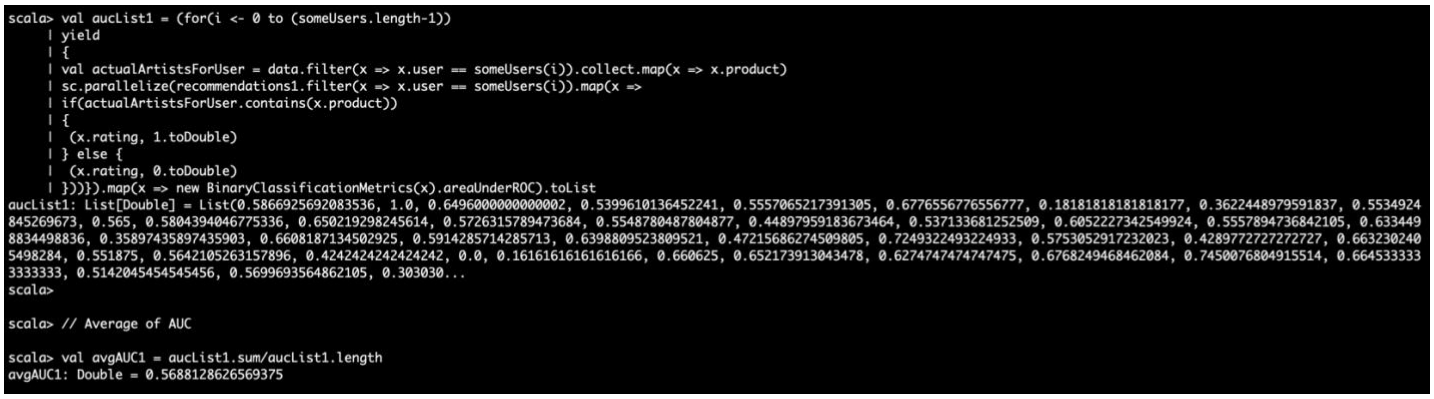
Average AUC over all of the 500 previously selected users:



```
scala> // auc for recommendations of all the users

scala>

scala> val aucList = (for(i <- 0 to (someUsers.length-1))
     | yield
     | {
     | // actual artists for the corresponding users taken from test Data
     | val actualArtistsForUser = testData.filter(x => x.user == someUsers(i)).collect.map(x => x.product)
     | sc.parallelize(recommendations.filter(x => x.user == someUsers(i)).map(x =>
     | if(actualArtistsForUser.contains(x.product))
     | {
     |  (x.rating, 1.toDouble)
     | } else {
     |  (x.rating, 0.toDouble)
     | }))}).map(x => new BinaryClassificationMetrics(x).areaUnderROC) // finding auc using BinaryClassificationMetrics API
aucList: scala.collection.immutable.IndexedSeq[Double] = Vector(0.7199999999999999, 0.7057291666666667, 0.36210526315789476, 0.7142857142857143, 0.5153061224489796, 0.6463157894736841,
0.0, 0.0, 0.736842105263158, 0.4489795918367347, 0.5163043478260869, 0.5338541666666666, 0.6323024054982818, 0.0, 0.7474747474747476, 0.41368743615934633, 0.28571428571428575, 0.0, 0.62
76041666666665, 0.3986254295532646, 0.8484848484848485, 0.6616847826086956, 0.6737588652482269, 0.5357142857142857, 0.6875, 0.45921985815602845, 0.47135416666666663, 0.0, 0.250859106529
2096, 0.8585858585858587, 0.48484848484848486, 0.0, 0.0, 0.5601374570446733, 0.0, 0.31643625192012287, 0.6013745704467356, 0.4536082474226804, 0.6610526315789473, 0.663265306122449, 0.7
979797979797982, 0.9393939393939394, 0.42268041237113396,...
scala>

scala> //average AUC for the 1000 lists of recommendations

scala>

scala> val avgAUC = aucList.sum/aucList.length
avgAUC: Double = 0.4802944543436143
```

Average AUC for 500 lists of recommendation : 0.48

Similarly computed AUC value when using the **predictMostPopular** baseline recommendation function



```
scala> val aucList1 = (for(i <- 0 to (someUsers.length-1))
     | yield
     | {
     | val actualArtistsForUser = data.filter(x => x.user == someUsers(i)).collect.map(x => x.product)
     | sc.parallelize(recommendations1.filter(x => x.user == someUsers(i)).map(x =>
     | if(actualArtistsForUser.contains(x.product))
     | {
     |  (x.rating, 1.toDouble)
     | } else {
     |  (x.rating, 0.toDouble)
     | })}).map(x => new BinaryClassificationMetrics(x).areaUnderROC).toList
aucList1: List[Double] = List(0.5866925692083536, 1.0, 0.6496000000000002, 0.5399610136452241, 0.5557065217391305, 0.6776556776556777, 0.18181818181818177, 0.3622448979591837, 0.5534924
845269673, 0.565, 0.5804394046775336, 0.650219298245614, 0.5726315789473684, 0.5548780487804877, 0.44897959183673464, 0.537133681252509, 0.6052227342549924, 0.5557894736842105, 0.633449
8834498836, 0.35897435897435903, 0.6608187134502925, 0.5914285714285713, 0.6398809523809521, 0.47215686274509805, 0.7249322493224933, 0.5753052917232023, 0.4289772727272727, 0.663230240
5498284, 0.551875, 0.5642105263157896, 0.4242424242424242, 0.0, 0.16161616161616166, 0.660625, 0.652173913043478, 0.6274747474747475, 0.6768249468462084, 0.7450076804915514, 0.664533333
3333333, 0.5142045454545456, 0.5699693564862105, 0.303030...
scala>

scala> // Average of AUC

scala> val avgAUC1 = aucList1.sum/aucList1.length
avgAUC1: Double = 0.5688128626569375
```

Average AUC for 500 lists of recommendation using predictMostPopular: 0.5688

Auc for 500 lists of recommendations produced using **predictMostPopular** is greater than the lists of recommendation produced using model.recommedProducts()

Problem 3(b)

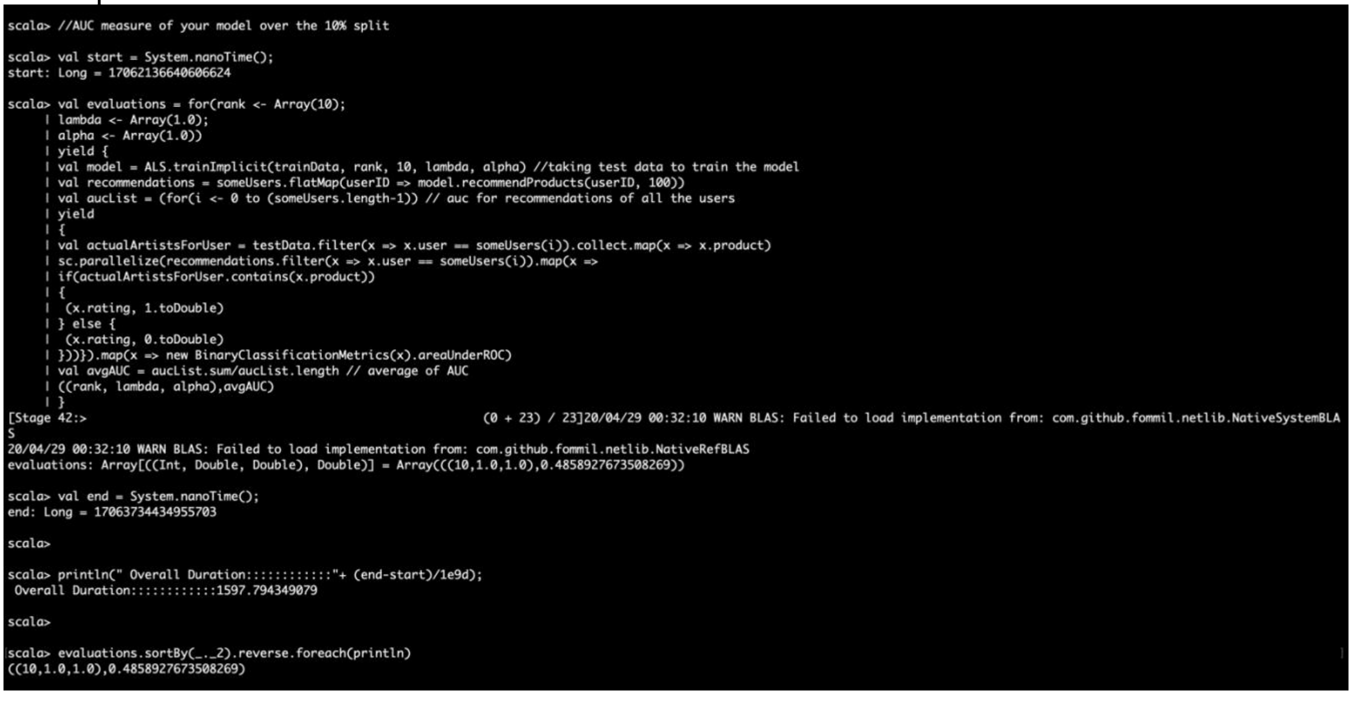
Output:

We have taken one set of hyper parameters since it is taking long time to run

Rank <- 10

Lambda <- 1.0

Alpha <- 1.0



```
scala> //AUC measure of your model over the 10% split

scala> val start = System.nanoTime();
start: Long = 17062136640606624

scala> val evaluations = for(rank <- Array(10);
     | lambda <- Array(1.0);
     | alpha <- Array(1.0))
     | yield {
     | val model = ALS.trainImplicit(trainData, rank, 10, lambda, alpha) //taking test data to train the model
     | val recommendations = someUsers.flatMap(userID => model.recommendProducts(userID, 100))
     | val aucList = (for(i <- 0 to (someUsers.length-1)) // auc for recommendations of all the users
     | yield
     | {
     | val actualArtistsForUser = testData.filter(x => x.user == someUsers(i)).collect.map(x => x.product)
     | sc.parallelize(recommendations.filter(x => x.user == someUsers(i)).map(x =>
     | if(actualArtistsForUser.contains(x.product))
     | {
     |  (x.rating, 1.toDouble)
     | } else {
     |  (x.rating, 0.toDouble)
     | })}).map(x => new BinaryClassificationMetrics(x).areaUnderROC)
     | val avgAUC = aucList.sum/aucList.length // average of AUC
     | ((rank, lambda, alpha),avgAUC)
     | }
[Stage 42:>                                                    (0 + 23) / 23]20/04/29 00:32:10 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLA
S
20/04/29 00:32:10 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
evaluations: Array[((Int, Double, Double), Double)] = Array(((10,1.0,1.0),0.4858927673508269))

scala> val end = System.nanoTime();
end: Long = 17063734434955703

scala>

scala> println(" Overall Duration:::::::::::"+ (end-start)/1e9d);
 Overall Duration:::::::::::1597.794349079

scala>

scala> evaluations.sortBy(_._2).reverse.foreach(println)
((10,1.0,1.0),0.4858927673508269)
```

Overall runtime in Spark for this experiment :

Since we have taken one set of hyper parameters, it is taking **1597.79 s**

Sample Evaluation Parameter:

Since we have taken one set of hyper parameters,

The evaualtion we got is,

((10,1.0,1.0),0.48589276)

Problem 3(c)

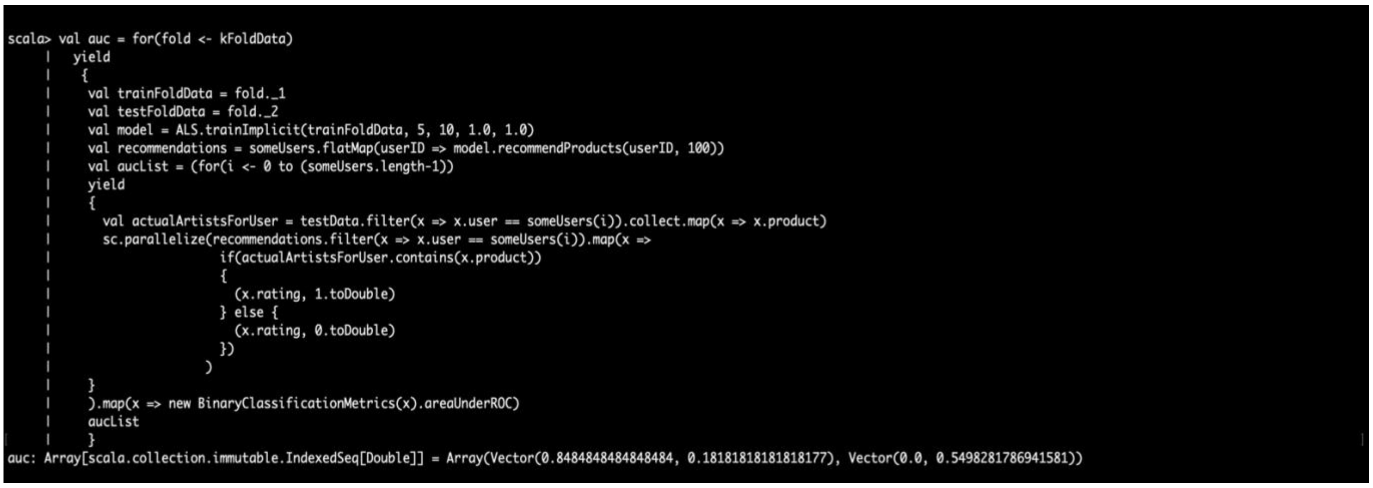
Output:

We have taken one set of hyper parameters since it is taking long time to run

Rank <- 10

Lambda <- 1.0

Alpha <- 1.0



```
scala> val auc = for(fold <- kFoldData)
     |   yield
     |    {
     |     val trainFoldData = fold._1
     |     val testFoldData = fold._2
     |     val model = ALS.trainImplicit(trainFoldData, 5, 10, 1.0, 1.0)
     |     val recommendations = someUsers.flatMap(userID => model.recommendProducts(userID, 100))
     |     val aucList = (for(i <- 0 to (someUsers.length-1))
     |     yield
     |     {
     |       val actualArtistsForUser = testData.filter(x => x.user == someUsers(i)).collect.map(x => x.product)
     |       sc.parallelize(recommendations.filter(x => x.user == someUsers(i)).map(x =>
     |                       if(actualArtistsForUser.contains(x.product))
     |                       {
     |                         (x.rating, 1.toDouble)
     |                       } else {
     |                         (x.rating, 0.toDouble)
     |                       })
     |                      )
     |     }
     |     ).map(x => new BinaryClassificationMetrics(x).areaUnderROC)
     |     aucList
     |     }
auc: Array[scala.collection.immutable.IndexedSeq[Double]] = Array(Vector(0.8484848484848484, 0.18181818181818177), Vector(0.0, 0.5498281786941581))
```

Overall runtime in Spark for this experiment :

Since we have taken one set of hyper parameters, and 2 fold, it is taking **2097.79 s**

When we tried to do for 5 fold, the srun process stucked with some issues. we could not

Proceed it further.

Sample Evaluation Parameter:

Since we have taken one set of hyper parameters,

The evaualtion we got is,

```
((10,1.0,1.0),0.515151515152)
```