Домашна задача 2 – Spark

Милан Тасевски, 196001

```
from dataclasses import dataclass, field
import pyspark
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from pyspark.sql.types import IntegerType

import os
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Domasna2").getOrCreate()

sc = spark.sparkContext

df = spark.read.text("ml-100k/u.data")

df = df.selectExpr("split(value, '\t') as userID_itemID_rating_timestamp")

# print(df.head(3))

# add columns for each element of the array to seperate and cast
df = df.withColumn('user_id', df['userID_itemID_rating_timestamp'][0].cast(IntegerType()))
df = df.withColumn('item_id', df['userID_itemID_rating_timestamp'][1].cast(IntegerType()))
df = df.withColumn('rating', df['userID_itemID_rating_timestamp'][2].cast(IntegerType()))
df = df.withColumn('timestamp', df['userID_itemID_rating_timestamp'][3].cast(IntegerType()))

df = df.drop('userID_itemID_rating_timestamp')

# show the dataframe
df.show()
# df.summary().show()
# print(df.head(3))
```

Првин се креира SparkSession со првата команда и име Domasna2 и се зема контекстот од таа сесија. Потоа, се вчитуваат податоците и се спремаат за обработка во форма на pyspark DataFrame, од една колона се кастираат и мапираат во 4, според тоа каков е форматот (user_id , item_id, rating, timestamp). Се прикажуваат податоците кои изгледаат вака:

```
+-------+-------+------+---------+
|user_id|item_id|rating|timestamp|
+-------+-------+------+---------+
|    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|
+-------+-------+------+---------+
only showing top 20 rows
```

```
# map each row from the df into an objet of type Rating
ratings = df.rdd.map(lambda l: Rating(user=l['user_id'],product=l['item_id'],rating=l['rating']))
```

Потребно е да се мапира секој ред од податоците во објект од тип Rating, кој потоа се предава на алгоритмот како параметар. Ова е така заради начинот на кој е имплементиран ALS.

```
# a class for representing a mean squared error for the model with give parameters
@dataclass(order=True)
class MSE:
    sort_index: int = field(init=False)
    mse: float
    rank: int
    iterations: int
    l: int

    def __post_init__(self):
        self.sort_index = int (self.mse * 10000)
```

Користам една помошна класа MSE за репрезентација на еден експеримент, односно параметрите кои се менуваат при секое ново тренирање на алгоритмот и mean-squared-error-от врз податоците кој се користи како мерка за евалуација. Со помош на ова можам да ги рангирам моделите кои се градат со различни параметри и да го изберам најдобриот.

```
# train an ALS model with the prepared data
ranks = [i for i in range(10,18,2)]
numIterations = [i for i in range (10,18,2)]
lambdas = [0.001, 0.01, 0.1]

mean_squared_errors = []

# iterate for the given parameters, train and save a model based on them, save the parameters wit
for rank in ranks:
    for iterations in numIterations:
        for l in lambdas:
            model = ALS.train(ratings, rank, iterations, l)
            testdata = ratings.map(lambda p: (p[0], p[1]))
            predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
            ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
            mse = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
            print('{0} {1} {2}'.format(rank, iterations, l))
            print("Mean Squared Error = " + str(mse))
            mse_obj = MSE(rank=rank,iterations=iterations,l=l,mse=mse)
            mean_squared_errors.append(mse_obj)
            model.save(sc, "models/model-"+str(rank)+"-"+str(iterations)+"-"+str(l))


# print(mean_squared_errors)
sorted_list = sorted(mean_squared_errors, key=lambda x: x.sort_index)
for element in sorted_list:
    print(element)
```

Ова е главниот дел со експериментите, каде за неколку различни вредности на парамтерите rank, iterations и lambda дефинирани во листите, со итерација за секоја се креира нов модел кој го користи ALS алгоритмот за тренирање. Се пресметува MSE за секој од моделите и се зачувува во листа, со помош на датакласата спомната погоре. Понатаму, оваа листа се сортира според sort_index кој се пресметува врз база на MSE, и првиот елемент на листата е објект кој ги зачувува парамтертите на најдобриот модел, оној со најмал MSE. Самите модели се зачувуваат локално за понатамошно load-ирање.

```python
# load the best model
best_model_mse = sorted_list[0]
print(best_model_mse)
best_model = MatrixFactorizationModel.load(sc, "models/model-"+
            str(best_model_mse.rank)+"-"+str(best_model_mse.iterations)+"-"+str(best_model_mse.l))


# predictions from the loaded model
predictions = best_model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error For The Best Model = " + str(MSE))
```

Се лоадира најдобриот модел и се користи за предикција на множеството. Се печати неговото MSE.

Ова се резултатите од експериментите:

10 10 0.01

Mean Squared Error = 0.482995806932591

10 10 0.1

Mean Squared Error = 0.5929079351149875

10 12 0.001

Mean Squared Error = 0.4833516818907925

10 12 0.01

Mean Squared Error = 0.4774691477452921

10 12 0.1

Mean Squared Error = 0.5931105825935015

10 14 0.001

Mean Squared Error = 0.4820613093866464

10 14 0.01

Mean Squared Error = 0.4748101581901155

10 14 0.1

Mean Squared Error = 0.5914084946107554

10 16 0.001

Mean Squared Error = 0.4756448452157869

10 16 0.01

Mean Squared Error = 0.47629000425580714

10 16 0.1

Mean Squared Error = 0.5864067535816531

12 10 0.001

Mean Squared Error = 0.4450910559938311

12 10 0.01

Mean Squared Error = 0.440865093655516

12 10 0.1

Mean Squared Error = 0.5720328420910146

12 12 0.001

Mean Squared Error = 0.4416298013461346

12 12 0.01

Mean Squared Error = 0.43711700111160784

12 12 0.1

Mean Squared Error = 0.5706179811768842

12 14 0.001

Mean Squared Error = 0.43564766219977064

12 14 0.01

Mean Squared Error = 0.431857626177579

12 14 0.1

Mean Squared Error = 0.5693704732879901

12 16 0.001

Mean Squared Error = 0.4334601633197006

12 16 0.01

Mean Squared Error = 0.42985038381481566

12 16 0.1

Mean Squared Error = 0.5658497527767702

14 10 0.001

Mean Squared Error = 0.4075822908299367

14 10 0.01

Mean Squared Error = 0.3995242633462642

14 10 0.1

Mean Squared Error = 0.5535634125911839

14 12 0.001

Mean Squared Error = 0.3993869129134475

14 12 0.01

Mean Squared Error = 0.3965564796612056

14 12 0.1

Mean Squared Error = 0.5537019241528686

14 14 0.001

Mean Squared Error = 0.3981445786497867

14 14 0.01

Mean Squared Error = 0.39286097399270475

14 14 0.1

Mean Squared Error = 0.5484013452579674

14 16 0.001

Mean Squared Error = 0.3940975736188304

14 16 0.01

Mean Squared Error = 0.3917527099925932

14 16 0.1

Mean Squared Error = 0.5454038651293973

16 10 0.001

Mean Squared Error = 0.36886036758930174

16 10 0.01

Mean Squared Error = 0.36565249691400925

16 10 0.1

Mean Squared Error = 0.5372541235815056

16 12 0.001

Mean Squared Error = 0.3658744415345983

16 12 0.01

Mean Squared Error = 0.3632368126364752

16 12 0.1

Mean Squared Error = 0.5358576278530123

16 14 0.001

Mean Squared Error = 0.3621277677482654

16 14 0.01

Mean Squared Error = 0.35874145557239184

16 14 0.1

Mean Squared Error = 0.5321094484750732

16 16 0.001

Mean Squared Error = 0.3591184172005424

16 16 0.01

Mean Squared Error = 0.35477596425515445

16 16 0.1

Mean Squared Error = 0.5309724192854823

Листата од MSE објекти испечатена по редослед:

MSE(sort_index=3547, mse=0.35477596425515445, rank=16, iterations=16, l=0.01)

MSE(sort_index=3587, mse=0.35874145557239184, rank=16, iterations=14, l=0.01)

MSE(sort_index=3591, mse=0.3591184172005424, rank=16, iterations=16, l=0.001)

MSE(sort_index=3621, mse=0.3621277677482654, rank=16, iterations=14, l=0.001)

MSE(sort_index=3632, mse=0.3632368126364752, rank=16, iterations=12, l=0.01)

MSE(sort_index=3656, mse=0.36565249691400925, rank=16, iterations=10, l=0.01)

MSE(sort_index=3658, mse=0.3658744415345983, rank=16, iterations=12, l=0.001)

MSE(sort_index=3688, mse=0.36886036758930174, rank=16, iterations=10, l=0.001)

MSE(sort_index=3917, mse=0.3917527099925932, rank=14, iterations=16, l=0.01)

MSE(sort_index=3928, mse=0.39286097399270475, rank=14, iterations=14, l=0.01)

MSE(sort_index=3940, mse=0.3940975736188304, rank=14, iterations=16, l=0.001)

MSE(sort_index=3965, mse=0.3965564796612056, rank=14, iterations=12, l=0.01)

MSE(sort_index=3981, mse=0.3981445786497867, rank=14, iterations=14, l=0.001)

MSE(sort_index=3993, mse=0.3993869129134475, rank=14, iterations=12, l=0.001)

MSE(sort_index=3995, mse=0.3995242633462642, rank=14, iterations=10, l=0.01)

MSE(sort_index=4075, mse=0.4075822908299367, rank=14, iterations=10, l=0.001)

MSE(sort_index=4298, mse=0.42985038381481566, rank=12, iterations=16, l=0.01)

MSE(sort_index=4318, mse=0.431857626177579, rank=12, iterations=14, l=0.01)

MSE(sort_index=4334, mse=0.4334601633197006, rank=12, iterations=16, l=0.001)

MSE(sort_index=4356, mse=0.43564766219977064, rank=12, iterations=14, l=0.001)

MSE(sort_index=4371, mse=0.43711700111160784, rank=12, iterations=12, l=0.01)

MSE(sort_index=4408, mse=0.440865093655516, rank=12, iterations=10, l=0.01)

MSE(sort_index=4416, mse=0.4416298013461346, rank=12, iterations=12, l=0.001)

MSE(sort_index=4450, mse=0.4450910559938311, rank=12, iterations=10, l=0.001)

MSE(sort_index=4748, mse=0.4748101581901155, rank=10, iterations=14, l=0.01)

MSE(sort_index=4756, mse=0.4756448452157869, rank=10, iterations=16, l=0.001)

MSE(sort_index=4762, mse=0.47629000425580714, rank=10, iterations=16, l=0.01)

MSE(sort_index=4774, mse=0.4774691477452921, rank=10, iterations=12, l=0.01)

MSE(sort_index=4820, mse=0.4820613093866464, rank=10, iterations=14, l=0.001)

MSE(sort_index=4829, mse=0.482995806932591, rank=10, iterations=10, l=0.01)

MSE(sort_index=4833, mse=0.4833516818907925, rank=10, iterations=12, l=0.001)

MSE(sort_index=4903, mse=0.49033943905747585, rank=10, iterations=10, l=0.001)

MSE(sort_index=5309, mse=0.5309724192854823, rank=16, iterations=16, l=0.1)

MSE(sort_index=5321, mse=0.5321094484750732, rank=16, iterations=14, l=0.1)

MSE(sort_index=5358, mse=0.5358576278530123, rank=16, iterations=12, l=0.1)

MSE(sort_index=5372, mse=0.5372541235815056, rank=16, iterations=10, l=0.1)

MSE(sort_index=5454, mse=0.5454038651293973, rank=14, iterations=16, l=0.1)

MSE(sort_index=5484, mse=0.5484013452579674, rank=14, iterations=14, l=0.1)

MSE(sort_index=5535, mse=0.5535634125911839, rank=14, iterations=10, l=0.1)

MSE(sort_index=5537, mse=0.5537019241528686, rank=14, iterations=12, l=0.1)

MSE(sort_index=5658, mse=0.5658497527767702, rank=12, iterations=16, l=0.1)

MSE(sort_index=5693, mse=0.5693704732879901, rank=12, iterations=14, l=0.1)

MSE(sort_index=5706, mse=0.5706179811768842, rank=12, iterations=12, l=0.1)

MSE(sort_index=5720, mse=0.5720328420910146, rank=12, iterations=10, l=0.1)

MSE(sort_index=5864, mse=0.5864067535816531, rank=10, iterations=16, l=0.1)

MSE(sort_index=5914, mse=0.5914084946107554, rank=10, iterations=14, l=0.1)

MSE(sort_index=5929, mse=0.5929079351149875, rank=10, iterations=10, l=0.1)

MSE(sort_index=5931, mse=0.5931105825935015, rank=10, iterations=12, l=0.1)

MSE(sort_index=3547, mse=0.35477596425515445, rank=16, iterations=16, l=0.01)

Заклучуваме дека најдобри резултати се постигнуваат со ранк 16, итерации 16 и регуларизациска константа 0.01.