# Домашна задача 2

Тијана Атанасовска (196014)

Барање:

Во рамки на втората домашна задача ќе треба да поставите Spark околина (https://spark.apache.org/) за аналитика на големи податоци.

Потребно е да го примените алгоритамот за препораки (alternating least squares, имплементација https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html) за решавање на проблемот за препораки на филмови над податочното множество (https://grouplens.org/datasets/movielens/100k/).

Решение:

Најпрвин се креира Спарк сесија во која ќе се извршуваат манипулациите.

Потоа се вчитуваат податоците со spark.read.text() бидејќи се во формат .data.

Потребно е да се поделат колоните бидејќи не се при читање со горната функција. Овде не ги вчитувам timestamp бидејќи не се потребни.

```
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 data")

df = df.withColumn('user', df['data'][0].cast(IntegerType()))
df = df.withColumn('item', df['data'][1].cast(IntegerType()))
df = df.withColumn('rating', df['data'][2].cast(IntegerType()))

df.show()

ratings = df.rdd.map(lambda l: Rating(user=l['user'],product=l['item'],rating=l['rating']))

ranks = [11, 12, 13]
iterationsnum = [11, 12, 13]
lambdas = [0.001, 0.01]
```

```
+--------------------+----+----+------+
|                data|user|item|rating|
+--------------------+----+----+------+
|[196, 242, 3, 881...| 196| 242|     3|
|[186, 302, 3, 891...| 186| 302|     3|
|[22, 377, 1, 8788...|  22| 377|     1|
|[244, 51, 2, 8806...| 244|  51|     2|
|[166, 346, 1, 886...| 166| 346|     1|
|[298, 474, 4, 884...| 298| 474|     4|
|[115, 265, 2, 881...| 115| 265|     2|
|[253, 465, 5, 891...| 253| 465|     5|
|[305, 451, 3, 886...| 305| 451|     3|
|[6, 86, 3, 883603...|   6|  86|     3|
|[62, 257, 2, 8793...|  62| 257|     2|
|[286, 1014, 5, 87...| 286|1014|     5|
|[200, 222, 5, 876...| 200| 222|     5|
|[210, 40, 3, 8910...| 210|  40|     3|
|[224, 29, 3, 8881...| 224|  29|     3|
|[303, 785, 3, 879...| 303| 785|     3|
|[122, 387, 5, 879...| 122| 387|     5|
|[194, 274, 2, 879...| 194| 274|     2|
|[291, 1042, 4, 87...| 291|1042|     4|
|[234, 1184, 2, 89...| 234|1184|     2|
+--------------------+----+----+------+
only showing top 20 rows
```

За да се тестира алгоритамот со различни параметри, креирам листи за различни ranks, numIterations, lambdas.

Се итерира низ комбинациите и се повикува ALS.train() за да се тренира врз податоците. При тоа, ако следниот модел даде подобри резултати го зачувувам него. Се забележува намалувањето на MCE низ итерациите.

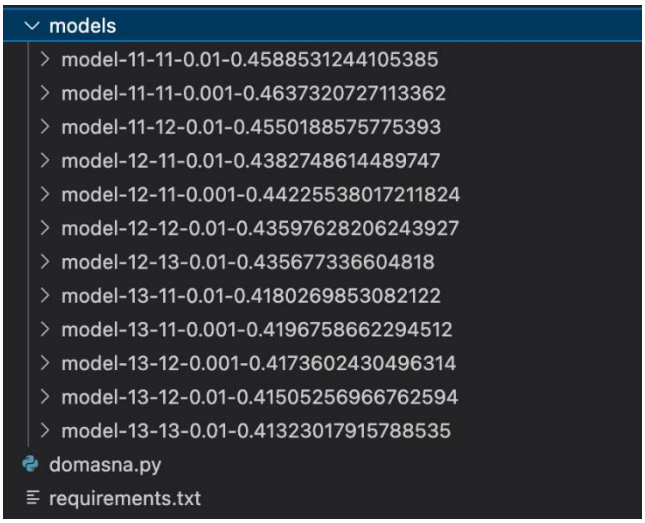
```
mse_best = 100.0
rank_best = ranks[0]
iterations_best = iterationsnum[0]
lambda_best = lambdas[0]
for rank in ranks:
    for iterations in iterationsnum:
        for l in lambdas:
            model = ALS.train(ratings, rank, iterations, l)
            testdata = ratings.map(lambda p: (p[0], p[1]))
            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))
            # save only the best model by far
            if (mse < mse_best):
                mse_best = mse
                rank_best = rank
                iterations_best = iterations
                lambda_best = l
                model.save(sc, "models/model-"+str(rank_best)+"-"+str(iterations_best)+
                                "-"+str(lambda_best)+"-"+str(mse_best))
```

```
11 11 0.01
Mean Squared Error = 0.4588531244105385
11 12 0.001
Mean Squared Error = 0.463217393842734
11 12 0.01
Mean Squared Error = 0.4550188575775393
11 13 0.001
Mean Squared Error = 0.4595969437789822
11 13 0.01
Mean Squared Error = 0.45595209675647674
12 11 0.001
Mean Squared Error = 0.44225538017211824
12 11 0.01
Mean Squared Error = 0.4382748614489747
12 12 0.001
Mean Squared Error = 0.43989851367735405
12 12 0.01
Mean Squared Error = 0.43597628206243927
12 13 0.001
Mean Squared Error = 0.43900469144742726
12 13 0.01
Mean Squared Error = 0.435677336604818
13 11 0.001
Mean Squared Error = 0.4196758662294512
13 11 0.01
Mean Squared Error = 0.4180269853082122
13 12 0.001
Mean Squared Error = 0.4173602430496314
13 12 0.01
Mean Squared Error = 0.41505256966762594
13 13 0.001
Mean Squared Error = 0.4172924742417765
13 13 0.01
Mean Squared Error = 0.41323017915788535
0.41323017915788535
```

На крај, се користи најдобриот модел (со најмал MSE) за да се предвидат резултатите. Се вчитува со MatrixFactorizationModel. Преку имињата на фолдерите во кој се зачувуваат моделите, може да се забележи кој е најдобриот модел поради начинот на запишување на името.

```
print(mse_best)
best_model = MatrixFactorizationModel.load(sc, "models/model-"+str(rank_best)+"-"+str(iterations_best)+
        "-"+str(lambda_best)+"-"+str(mse_best))

# 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))
```




```
Mean Squared Error For The Best Model = 0.41323017915788535
```

Користена литература:

https://medium.com/analytics-vidhya/movie-lens-data-analysis-using-pyspark-for-beginners-9c0f5f21eaf5

https://towardsdatascience.com/collaborative-filtering-in-pyspark-52617dd91194

https://medium.com/geekculture/recommender-system-using-collaborative-filtering-in-pyspark-b98eab2aea75