657 HW4 Report

Rajeev Kashetti
Sai Karthik Dindi

- we had loaded data into the dataframe using Spark.read.option(“header”, “true”).csv(path_of_the_dataset), as shown in figure below

```
data = spark.read.csv("./ml-20m/ratings.csv",header= True)
data1=spark.read.csv("./ml-20m/movies.csv",header= True)
```

- We then joined both the dataframes as one in “data” has user_id, Movie_id and corresponding rating. While one in the “data1” has movie_id and its information like title and genres. We joined both data sets on movie_id using inner join.

```
data1=data1.withColumnRenamed("movieId", "m_movieId")

final_data=data.join(data1,data.movieId ==  data1.m_movieId,"inner")
```

- The resultant final_data looks as follows:

userId	movieId	rating	title	genres
1	2	3.5	Jumanji (1995)	Adventure|Children|Fantasy
1	29	3.5	City of Lost Children, The (Cité des enfants perdus, La) (1995)	Adventure|Drama|Fantasy|Mystery|Sci-Fi
1	32	3.5	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	Mystery|Sci-Fi|Thriller
1	47	3.5	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
1	50	3.5	Usual Suspects, The (1995)	Crime|Mystery|Thriller
1	112	3.5	Rumble in the Bronx (Hont faan kui) (1995)	Action|Adventure|Comedy|Crime
1	151	4.0	Rob Roy (1995)	Action|Drama|Romance|War
1	223	4.0	Clerks (1994)	Comedy
1	253	4.0	Interview with the Vampire: The Vampire Chronicles (1994)	Drama|Horror
1	260	4.0	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
1	293	4.0	Léon: The Professional (a.k.a. The Professional) (Léon) (1994)	Action|Crime|Drama|Thriller
1	296	4.0	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller
1	318	4.0	Shawshank Redemption, The (1994)	Crime|Drama
1	337	3.5	What's Eating Gilbert Grape (1993)	Drama
1	367	3.5	Mask, The (1994)	Action|Comedy|Crime|Fantasy
1	541	4.0	Blade Runner (1982)	Action|Sci-Fi|Thriller
1	589	3.5	Terminator 2: Judgment Day (1991)	Action|Sci-Fi
1	593	3.5	Silence of the Lambs, The (1991)	Crime|Horror|Thriller
1	653	3.0	Dragonheart (1996)	Action|Adventure|Fantasy
1	919	3.5	Wizard of Oz, The (1939)	Adventure|Children|Fantasy|Musical

- As we can see the genres column is not properly formatted. So we had used regular expressions to replace anything other than text and digits with space. But this made sci-fi as sci fi which will be considered as two words. As it is not ideal we considered ’ -’ also to be in the genres column apart from text and numbers.

- We then tokeneized the title and genres column using `Tokenizer()` method in pyspark. Then we left with following dataframe

userId	movieId	rating	title1	genres1
1	2	3.5	[jumanji, 1995]	[adventure, children, fantasy]
1	29	3.5	[city, of, lost, children, the, cit, des, enfants, perdus, la, 1995]	[adventure, drama, fantasy, mystery, sci-fi]
1	32	3.5	[twelve, monkeys, aka, 12, monkeys, 1995]	[mystery, sci-fi, thriller]
1	47	3.5	[seven, aka, se7en, 1995]	[mystery, thriller]
1	50	3.5	[usual, suspects, the, 1995]	[crime, mystery, thriller]
1	112	3.5	[rumble, in, the, bronx, hont, faan, kui, 1995]	[action, adventure, comedy, crime]
1	151	4.0	[rob, roy, 1995]	[action, drama, romance, war]
1	223	4.0	[clerks, 1994]	[comedy]
1	253	4.0	[interview, with, the, vampire, the, vampire, chronicles, 1994]	[drama, horror]
1	260	4.0	[star, wars, episode, iv, , a, new, hope, 1977]	[action, adventure, sci-fi]
1	293	4.0	[lon, the, professional, aka, the, professional, lon, 1994]	[action, crime, drama, thriller]
1	296	4.0	[pulp, fiction, 1994]	[comedy, crime, drama, thriller]
1	318	4.0	[shawshank, redemption, the, 1994]	[crime, drama]
1	337	3.5	[whats, eating, gilbert, grape, 1993]	[drama]
1	367	3.5	[mask, the, 1994]	[action, comedy, crime, fantasy]
1	541	4.0	[blade, runner, 1982]	[action, sci-fi, thriller]
1	589	3.5	[terminator, 2, judgment, day, 1991]	[action, sci-fi]
1	593	3.5	[silence, of, the, lambs, the, 1991]	[crime, horror, thriller]
1	653	3.0	[dragonheart, 1996]	[action, adventure, fantasy]
1	919	3.5	[wizard, of, oz, the, 1939]	[adventure, children, fantasy, musical]

- We then used CountVectorizer() method in pyspark to get vocab length for title as well as genres column
- we created Tf-Idf representation of those columns with number of features= vocab_length of that column we got from countVectorizer

```
#TF-idf

hashingTF = HashingTF(inputCol="title1", outputCol="title3", numFeatures=title_vocab_length)
featurizedData = hashingTF.transform(final_data)
# alternatively, CountVectorizer can also be used to get term frequency vectors

idf = IDF(inputCol="title3", outputCol="title")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)
```

- Using a vector assembler we had created a features column for Linear regression model which gives the following result

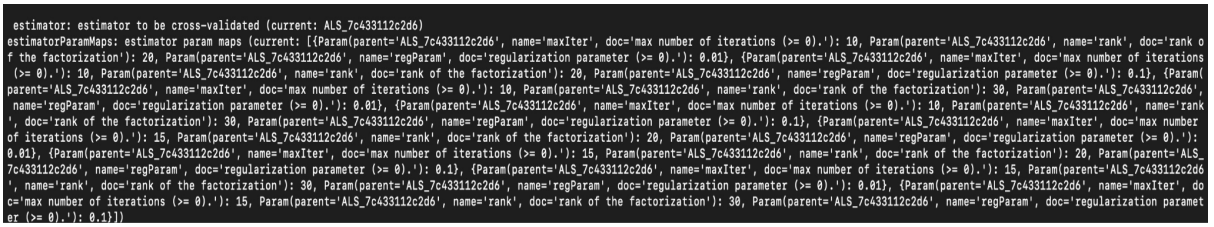
userId	movieId	rating	title	genres	features
1	2	3	(22293,[1339,1514...]	(23,[2,5,18],[1.5...]	(22318,[0,1,1341,...]
1	29	3	(22293,[1339,1438...]	(23,[2,5,16,20,21...]	(22318,[0,1,1341,...]
1	32	3	(22293,[378,1339,...]	(23,[9,20,21],[1....]	(22318,[0,1,380,1...]
1	47	3	(22293,[1339,1293...]	(23,[9,20],[1.325...]	(22318,[0,1,1341,...]
1	50	3	(22293,[1339,1915...]	(23,[5,9,20],[1.3...]	(22318,[0,1,1341,...]
1	112	3	(22293,[1339,4842...]	(23,[2,5,7,8],[1....]	(22318,[0,1,1341,...]
1	151	4	(22293,[732,1339,...]	(23,[7,12,13,16],...]	(22318,[0,1,734,1...]
1	223	4	(22293,[6364,1667...]	(23,[8],[0.980544...]	(22318,[0,1,6366,...]
1	253	4	(22293,[915,1373,...]	(23,[10,16],[2.60...]	(22318,[0,1,917,1...]
1	260	4	(22293,[85,1024,4...]	(23,[2,7,21],[1.5...]	(22318,[0,1,87,10...]
1	293	4	(22293,[10206,164...]	(23,[5,7,9,16],[1...]	(22318,[0,1,10208...]
1	296	4	(22293,[2624,1449...]	(23,[5,8,9,16],[1...]	(22318,[0,1,2626,...]
1	318	4	(22293,[12106,166...]	(23,[5,16],[1.324...]	(22318,[0,1,12108...]
1	337	3	(22293,[4082,5295...]	(23,[16],[0.81444...]	(22318,[0,1,4084,...]
1	367	3	(22293,[12647,166...]	(23,[5,7,8],[2.64...]	(22318,[0,1,12649...]
1	541	4	(22293,[1621,2012...]	(23,[7,9,21],[1.2...]	(22318,[0,1,1623,...]
1	589	3	(22293,[4236,5076...]	(23,[7,21],[1.270...]	(22318,[0,1,4238,...]
1	593	3	(22293,[4236,4764...]	(23,[5,9,10],[1.3...]	(22318,[0,1,4238,...]
1	653	3	(22293,[11260,154...]	(23,[2,5,7],[1.51...]	(22318,[0,1,11262...]
1	919	3	(22293,[7359,1002...]	(23,[2,5,6,18],[1...]	(22318,[0,1,7361,...]

- We had split the data into training and test set with seed=0 in the ratio of 0.8 and 0.2 respectively using rescaledData.randomsplit() method in pyspark
- Using ALS() method in pyspark we had built as ALS model using userid,movieid and rating and with coldstartStrategy='drop' using trainset crossvalidated on following parameters and got best results for maxIter=10,rank=20,regParam=0.01

```
paramGrid = ParamGridBuilder()\
            .addGrid(als.maxIter,[10,15,20])\
            .addGrid(als.rank,[10,20,30])\
            .addGrid(als.regParam,[0.01,0.1])\
            .build()

crossval = CrossValidator( estimator = als,
                           estimatorParamMaps = paramGrid,
                           evaluator = RegressionEvaluator(),
                           numFolds = 5)
```

- We then validated our model performance using testset and results are as follows
  ALS Results:
  RMSE: 0.8341343886952859
  MSE: 0.6957801784040583
  MAP: 29
  The best parameters are given below:



- **Item based collaborative filtering:**
  For this part we had used the code form the class activity to get movie pair similarity. We then joined rating data that has user_id, movie_id and ratings with test data using inner join to get all possible pairs of user_id and movie_id in the test dataset. Then cartesian product was applied on movie pair similarity data with the output of above join operation. Which is then filtered to only have user mapped to the movies he rated along with the similarity scores. Then this data is grouped by user_id,movie_id to obtain something like this : (userid,movieid) => ((m1,m2),(sm1,c1)), ((m1,m2),(sm1,c1)),..... This is then used to compute the required prediction. We have already computed mean_user_ratings, movie_mean_ratings, all_movies_mean_rating which are required in the formula to compute the prediction.

```
def getpredictions(x):
    sim_sum = 0
    rating_sum = 0
    user_mean = 0
    for i in range(10):
        (userid,movieid,rating),(_,sm)=x[0]
        sim_sum += sm[0]
        rating_sum += rating
        user_mean += user_mean_ratings(userid)
    fraction = (sim_sum+rating_sum - user_mean)/sim_sum
    prediction = fraction + user_mean_ratings(userid)+ movie_mean_ratings[movieid] - all_movies_mean
    return ((userid,movieid,rating),prediction)
```

Item based collaborative filtering Results:

RMSE: 0.8143195829568349
MSE: 0.6631163831869936
MAP: 29

Hybrid Model combining Item based CF and ALS models:

We used the weights **0.3,0.7** for **ALS, Item-based-collaborative filtering** respectively. The final metrics are as follows

RMSE: 0.8138911689383143
MSE: 0.6624188348757756
MAP: 28

We noticed that overall error got lesser using hybrid model

- **Linear regression:**
  Using linear regression() in pyspark we had built a Linear Regression model using features we got from vectorassembler and evaluated on test set.

```
paramGrid = ParamGridBuilder() \
.addGrid(lr.maxIter,[10,15,20])\
.addGrid(lr.regParam, [0.1,0.001]) \
.build()

crossval = CrossValidator(estimator=lr,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(labelCol = "rating"),
                          numFolds=5)
```

Linear Regression Results:
RMSE: 1.0061642586637913
MSE: 1.0123665154
MAP: 32

Hybrid Model combining all three models:

We used the weights **0.8,0.2** for **ALS, previous Hybrid model and Linear regression** respectively. The final metrics are as follows

RMSE: 0.809394013541
MSE: 0.6551186691560086
MAP: 28

We noticed that overall error decreased with current hybrid model