Data Engineering Report

Loading and Exploring the Data:

The first step is to load in the data.

```
    # parse---

def parseRating(line):
    """
    Parses a rating record in MovieLens format userId::movieId::rating::timestamp .
    """
    fields = line.strip().split("::")
    return int(fields[3]) % 10, (int(fields[0]), int(fields[1]), float(fields[2]))

def parseMovie(line):
    """
    Parses a movie record in MovieLens format movieId::movieTitle .
    """
    fields = line.strip().split("::")
    return int(fields[0]), fields[1]

    # load---

def loadRatings(ratingsFile):
    """
    Load ratings from file.
    """
    if not isfile(ratingsFile):
        print("File %s does not exist." % ratingsFile)
        sys.exit(1)
    f = open(ratingsFile, 'r')
    ratings = filter(lambda r: r[2] > 0, [parseRating(line)[1] for line in f])
    f.close()
    if not ratings:
        print("No ratings provided.")
        sys.exit(1)
    else:
        return ratings
```

Figure 1. parse and load data

In the code shown in Figure 1, The system first parses and store the existing data sets (movies set and ratings set), and then the program loads the data.

Note: In *parseRating()*, the data is stored in a two-dimensional form, it needs to pay attention to the way of reading the data when loading the data.

```
    # load personal ratings
    myRatings = loadRatings(os.path.abspath('/home/aono/CS4337/Project1/personalRatings.txt')) #Original: '/home/ash
    myRatingsRDD = sc.parallelize(myRatings, 1)
    # (personal comments)easy to see and know what personalRatingsRDD is
    print("personalRatingsRDD:")
    print(myRatingsRDD.take(20))
    print("\n")


    # load ratings and movie titles

    movieLensHomeDir = os.path.abspath('/home/aono/CS4337/Project1/movielens/medium') #Original: '/home/ashish/movie

    # ratings is an RDD of (last digit of timestamp, (userId, movieId, rating))
    ratings = sc.textFile(join(movieLensHomeDir, "ratings.dat")).map(parseRating)
    # (personal comments)easy to see and know what RatingsRDD is
    print("RatingsRDD:")
    print(ratings.take(20))
    print("\n")

    # movies is an RDD of (movieId, movieTitle)
    movie = sc.textFile(join(movieLensHomeDir, "movies.dat")).map(parseMovie)
    # (personal comments)easy to see and know what moviesRDD is
    print("moviesRDD:")
    print(movie.take(20))
    print("\n")
    movies = dict(movie.collect())
    # Original code: movies = dict(sc.textFile(join(movieLensHomeDir, "movies.dat")).map(parseMovie).collect())
```

Figure 2. load data and show

In the code shown in Figure 2, the data in the *personalRatings* set, ratings set, and movies set are loaded. In my project, it uses print() to display part of the data, which helps me visually observe the content and form of the data.

Note: Be careful when using collect()!

Running this line of code can cause the driver to run out of memory.

Using *take()* method is a safer approach

Use *take()* to display only part of the data

```
personalRatingsRDD:
[(0, 1, 1.0), (0, 780, 2.0), (0, 590, 3.0), (0, 1210, 4.0), (0, 648, 5.0), (0, 344, 1.0), (0, 165, 2.0), (0, 153,
3.0), (0, 597, 4.0), (0, 1580, 5.0), (0, 231, 1.0)]


RatingsRDD:
[(0, (1, 1193, 5.0)), (9, (1, 661, 3.0)), (8, (1, 914, 3.0)), (5, (1, 3408, 4.0)), (1, (1, 2355, 5.0)), (8, (1, 119
7, 3.0)), (9, (1, 1287, 5.0)), (9, (1, 2804, 5.0)), (8, (1, 594, 4.0)), (8, (1, 919, 4.0)), (8, (1, 595, 5.0)), (2,
(1, 938, 4.0)), (1, (1, 2398, 4.0)), (4, (1, 2918, 4.0)), (3, (1, 1035, 5.0)), (8, (1, 2791, 4.0)), (8, (1, 2687,
3.0)), (7, (1, 2018, 4.0)), (3, (1, 3105, 5.0)), (9, (1, 2797, 4.0))]


moviesRDD:
[(1, 'Toy Story (1995)'), (2, 'Jumanji (1995)'), (3, 'Grumpier Old Men (1995)'), (4, 'Waiting to Exhale (1995)'),
(5, 'Father of the Bride Part II (1995)'), (6, 'Heat (1995)'), (7, 'Sabrina (1995)'), (8, 'Tom and Huck (1995)'),
(9, 'Sudden Death (1995)'), (10, 'GoldenEye (1995)'), (11, 'American President, The (1995)'), (12, 'Dracula: Dead a
nd Loving It (1995)'), (13, 'Balto (1995)'), (14, 'Nixon (1995)'), (15, 'Cutthroat Island (1995)'), (16, 'Casino (1
995)'), (17, 'Sense and Sensibility (1995)'), (18, 'Four Rooms (1995)'), (19, 'Ace Ventura: When Nature Calls (199
5)'), (20, 'Money Train (1995)')]
```

Figure 3. print the data

Data Pre-processing:

```
##################################################################################################
# my code here
##################################################################################################
############################## PART2--MACHINE LEARNING ###########################################
##################################################################################################


# (personal comments)
# create RDDs and DataFrames
# DataFrames is helpful to read and operate!
# --RatingsRDD and DataFrame--
r1 = lambda line: Row(userID = line[1][0], movieID = line[1][1], rating = line[1][2]) # This is 2-D. Be careful!
Ratings_df = ratings.map(r1).toDF()
Ratings_df.show(5)
# (personal comments)
# maxIter is the maximum number of iterations to run (defaults to 10).
# regParam specifies the regularization parameter in ALS (defaults to 1.0)



# --personalRatingsRDD and DataFrame--
r2 = lambda line: Row(userID = line[0], movieID = line[1], rating = line[2])
personalRatings_df = myRatingsRDD.map(r2).toDF()
personalRatings_df.show(5)
```

Figure 4. RDD -> DataFrame

In the code shown in Figure 4, the project converted the RDD into a more intuitive and readable form-DateFrame. This not only helps me to visually view the content and form of the data, but the DataFrame is also more in line with the training rules of the ALS algorithm.

```
# --personalRatingsRDD and DataFrame--
r2 = lambda line: Row(userID = line[0], movieID = line[1], rating = line[2])
personalRatings_df = myRatingsRDD.map(r2).toDF()
personalRatings_df.show(5)


# (personal comments) 80% is training sets, 20% is testing sets
(training, test) = Ratings_df.randomSplit([.8,.2], seed = 3500)


# (personal comments) add the personalRatings data (for training)!
training = training.union(personalRatings_df)
```

Figure 5. Divide the data set

In the code shown in Figure 5, the project divided the data set into two parts, 80% of the data set is used for training, and 20% of the data set is used for testing. It is of course possible to divide the data set into other ratios, but 8:2 is more commonly used.
In the last line of code in the figure, the project combined the *personalRatings* data set with the *ratings* data set. This allows the *personalRatings* data set to be added to the training set.

Standardisation:

```
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=float(p[2]), timestamp=int(p[3])))
ratings = spark.createDataFrame(ratingsRDD)
```

```
# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics

# (personal comments)
# use ALS model (important!)
# I tried to keep adjusting maxIter and regParam, and the RMSE was about 0.89 when they were 10 and 0.01,
# respectively, and about 0.87 when they were 15 and 0.05, respectively.
# After many attempts, I used both 20 and 0.06 and the RMSE was about 0.85,
# so maybe there is still a possibility optimization.
als = ALS(maxIter = 20, regParam = 0.06, userCol="userID", itemCol="movieID", ratingCol="rating", coldStartStrateg
model = als.fit(training)


# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
# show the RMSE
print("Root-mean-square error = " + str(rmse))
print("\n")
```

Figure 6. Standardize data and use ALS model to train data & calculate RMSE

To scale project data to be able to use ML library, it needs to use Spark ML to do.
First, the project trains the data, and then it can generate prediction results and calculate RMSE.

```
Root-mean-square error = 0.856073281534929
```

Figure 7. output of the RMSE

The project tried to keep adjusting *maxIter* and *regParam*, and the RMSE was about 0.89 when they were 10 and 0.01, respectively, and about 0.87 when they were 15 and 0.05, respectively. After many attempts, the project used both 20 and 0.06 and the RMSE was about 0.85, so maybe there is still a possibility of optimization.

```
# output the recommented result
print("5 Movies recommended for you:")
users = Ratings_df.select(als.getUserCol()).distinct().limit(1)
userSubsetRecs = model.recommendForUserSubset(users, 5)
userSubsetRecs.show()
```

Figure 8. Organize data and get movie recommendation results

The system organizes the data and gets the user's movie recommendation results.

```
5 Movies recommended for you:
 1: Last Summer in the Hamptons (1995)
 2: Savage Nights (Nuits fauves, Les) (1992)
 3: Guantanamera (1994)
 4: Catfish in Black Bean Sauce (2000)
 5: Source, The (1999)
```

Figure 9. Movie recommendation results

The system outputs the user's movie recommendation results.