CS 8001 Big Data

HW #4: Movie Ratings (10 points + 2 bonus points)

Jing Su

js929@mail.missouri.edu

In this exercise, you will write Python code on Spark to find and predict movie ratings. See reference here. The dataset to be used is MovieLens 1M and 20M datasets at http://grouplens.org/datasets/movielens/.

Part I (10 points): Run Spark on MovieLens 1M datasets.

Spark Envoirnment: AWS EMR (4 nodes), Windows 10

1. Basic Recommendations: find the top 20 movies with highest average ratings and more than 500 reviews. Print out the names of the movies, their average ratings, and their numbers of reviews, in descending order of their average ratings.

Code Structure:

1) Parsing the two files yields two RDDS and cache them. For each line in the ratings dataset, I create a tuple of (UserID, MovieID, Rating). For each line in the movies dataset, I create a tuple of (MovieID, Title).
2) From ratingsRDD create an RDD with tuples of the form (MovieID, Python iterable of Ratings for that MovieID).
3) Using movieIDsWithRatingsRDD and getCountsAndAverages() function, compute the number of ratings and average rating for each movie to yield tuples of the form (MovieID, (number of ratings, average rating)).
4) To moviesRDD, apply transformations that use movieIDsWithAvgRatingsRDD to get the movie names for movieIDsWithAvgRatingsRDD, yielding tuples of the form (average rating, movie name, number of ratings).
5) Apply a single RDD transformation to movieNameWithAvgRatingsRDD to limit the results to movies with ratings from more than 500 people. We then use the sortFunction() helper function to sort by the average rating to get the movies in descending order of their rating.

Execution Process:

1) Put the 1M dataset to HDFS.

```
[hadoop@ip-172-31-38-247 ~]$ hadoop fs -put ratings.dat /user/hadoop/ratings.dat

[hadoop@ip-172-31-38-247 ~]$ hadoop fs -put movies.dat /user/hadoop/movies.dat
```

2) Use command: spark-submit RecommendMovie1M.py ratings.dat movies.dat

Result:

```
The top 20 movies with highest average ratings and more than 500 reviews: [(4.56
0509554140127, u'Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1
954)', 628), (4.554557700942973, u'Shawshank Redemption, The (1994)', 2227), (4.
524966261808367, u'Godfather, The (1972)', 2223), (4.52054794520548, u'Close Sha
ve, A (1995)', 657), (4.517106001121705, u'Usual Suspects, The (1995)', 1783), (
4.510416666666667, u"Schindler's List (1993)", 2304), (4.507936507936508, u'Wron
g Trousers, The (1993)', 882), (4.477724741447892, u'Raiders of the Lost Ark (19
81)', 2514), (4.476190476190476, u'Rear Window (1954)', 1050), (4.45369441658308
2, u'Star Wars: Episode IV - A New Hope (1977)', 2991), (4.449890270665691 5, u'D
r. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)', 136
7), (4.425646551724138, u'To Kill a Mockingbird (1962)', 928), (4.41560798548094
4, u'Double Indemnity (1944)', 551), (4.412822049131217, u'Casablanca (1942)', 1
669), (4.406262708418057, u'Sixth Sense, The (1999)', 2459), (4.401925391095066,
 u'Lawrence of Arabia (1962)', 831), (4.395973154362416, u'Maltese Falcon, The (
1941)', 1043), (4.390724637681159, u"One Flew Over the Cuckoo's Nest (1975)", 17
25), (4.388888888888889, u'Citizen Kane (1941)', 1116), (4.386993603411514, u'Br
idge on the River Kwai, The (1957)', 938)]
```

Python Code:

```python
from __future__ import print_function
import sys
from pyspark import SparkContext
import re
import time
import math
from pyspark.mllib.recommendation import ALS

def get_ratings_tuple(entry):
    """ Parse a line in the ratings dataset
    Args:
        entry (str): a line in the ratings dataset in the form of
        UserID::MovieID::Rating::Timestamp
    Returns:
        tuple: (UserID, MovieID, Rating)
    """
    items = entry.split('::')
    return int(items[0]), int(items[1]), float(items[2])

def get_movie_tuple(entry):
    """ Parse a line in the movies dataset
    Args:
        entry (str): a line in the movies dataset in the form of
        MovieID::Title::Genres
    Returns:
        tuple: (MovieID, Title)
    """
    items = entry.split('::')
    return int(items[0]), items[1]

def sortFunction(tuple):
    """ Construct the sort string (does not perform actual sorting)
    Args:
        tuple: (rating, MovieName)
    Returns:
```

```
        sortString: the value to sort with, 'rating MovieName'
    """
    key = unicode('%.3f' % tuple[0])
    value = tuple[1]
    return (key + ' ' + value)

def getCountsAndAverages(IDandRatingsTuple):
    """ Calculate average rating
    Args:
        IDandRatingsTuple: a single tuple of (MovieID, (Rating1, Rating2, Rating3, ...))
    Returns:
        tuple: a tuple of (MovieID, (number of ratings, averageRating))
    """
    list_tuples = [1.0*rating for rating in IDandRatingsTuple[1]]
    return (IDandRatingsTuple[0], (len(list_tuples), sum(list_tuples)/len(list_tuples)))

if __name__ == "__main__":
    start = time.time()
    #Print error message when the command is wrong.
    if len(sys.argv) != 3:
        print("Usage: RecommendMovie.py ratings.dat movies.dat", file=sys.stderr)
        exit(-1)
    sc = SparkContext(appName="RecommendMovies")
    #Create a SparkContext to store dat files.
    numPartitions = 2
    rawRatings = sc.textFile(sys.argv[1]).repartition(numPartitions)
    rawMovies = sc.textFile(sys.argv[2])

#Part 1: Basic Recommendations
    # Parsing the two files yields two RDDS and cache them.
    ratingsRDD = rawRatings.map(get_ratings_tuple).cache()
    moviesRDD = rawMovies.map(get_movie_tuple).cache()

    # From ratingsRDD with tuples of (UserID, MovieID, Rating) create an RDD with tuples of
    # the (MovieID, iterable of Ratings for that MovieID)
    movieIDsWithRatingsRDD = ratingsRDD.map(lambda (userid, movieid, rating):
    (movieid, rating)).groupByKey()

    # Using `movieIDsWithRatingsRDD`, compute the number of ratings and average rating for
    # each movie to yield tuples of the form (MovieID, (number of ratings, average rating))
    movieIDsWithAvgRatingsRDD = movieIDsWithRatingsRDD.map(lambda rec: getCountsAndAverages(rec))

    # To `movieIDsWithAvgRatingsRDD`, apply RDD transformations that use `moviesRDD` to get the movie
    # names for `movieIDsWithAvgRatingsRDD`, yielding tuples of the form
    # (average rating, movie name, number of ratings)
    movieNameWithAvgRatingsRDD = moviesRDD.join(movieIDsWithAvgRatingsRDD).map(lambda
    (movieID, (movieName, (numRatings, avgRating))): (avgRating, movieName, numRatings))

    # Apply an RDD transformation to `movieNameWithAvgRatingsRDD` to limit the results to movies with
    # ratings from more than 500 people. We then use the `sortFunction()` helper function to sort by
    # the average rating to get the movies in order of their rating (highest rating first)
    movieLimitedAndSortedByRatingRDD = movieNameWithAvgRatingsRDD.filter(lambda (_, __, numRatings):
    numRatings > 500).sortBy(sortFunction, False)
    print ('The top 20 movies with highest average ratings and more than 500 reviews: %s\n' %
    movieLimitedAndSortedByRatingRDD.take(20))
```

2. Collaborative filtering
    a. Break up the ratings dataset into three sets, training, validation, and test set as follows. Report the number of entries in each set.
        1) Sort the ratings data in ascending order of timestamp (past to present)
        2) Divide the data (80%-10%-10%) into
            - training set: the first 4/5 of the sorted data,
            - validation set: the next 1/10 of the sorted data,
            - test set: the last 1/10 of the sorted data.

Result:

```
Training: 800198, validation: 100030, test: 99981
```

    b. Use MLlib's alternating least squares method, ALS.train(), to train using the training set and validate using the validation set. Train 3 models (different ranks) and select the best model based on their validation performance. Report the ranks of your models, their training error (RMSE), training time in seconds, validation error (RMSE), and validation time in seconds, respectively. The model having the lowest validation error is your best model.

Result:

```
For rank 4 :

the training error is 0.865652272897

the training time is 10.5775930882

the validation error is 0.885625491895

the validation time is 1.11262798309

-----------------------------

For rank 8 :

the training error is 0.851508149701

the training time is 8.08406496048

the validation error is 0.884062583335

the validation time is 1.27135682106

-----------------------------

For rank 12 :

the training error is 0.843171246507

the training time is 7.05665397644

the validation error is 0.885762632323

the validation time is 1.05017805099

-----------------------------

The best training model was trained with rank 8
```

c. Test your best model on the test set. Report the test error (RMSE) and test time in seconds.

Result:

```
The model had a RMSE on the test set of 0.885953442082
```

Code Structure:

1) Break up the ratingsRDD dataset into three pieces:
   A training set (RDD), which we will use to train models;
   A validation set (RDD), which we will use to choose the best model;
   A test set (RDD), which we will use for our experiments.
2) Pick a set of model parameters. The most important parameter to ALS.train() is the rank, which is the number of rows in the Users matrix or the number of columns in the Movies matrix. We will train models with ranks of 4, 8, and 12 using the trainingRDD dataset.
3) Create a model using ALS.train(trainingRDD, rank, seed=seed, iterations=iterations, lambda_=regularizationParameter) with three parameters: an RDD consisting of tuples of the form (UserID, MovieID, rating) used to train the model, an integer rank (4, 8, or 12), a number of iterations to execute (we will use 5 for the iterations parameter), and a regularization coefficient (we will use 0.1 for the regularizationParameter).
4) For the prediction step, create an input RDD, validationForPredictRDD, consisting of (UserID, MovieID) pairs that we extract from validationRDD.
5) Using the model and validationForPredictRDD, we can predict rating values by calling model.predictAll() with the validationForPredictRDD dataset, where model is the model we generated with ALS.train(). predictAll accepts an RDD with each entry in the format (userID, movieID) and outputs an RDD with each entry in the format (userID, movieID, rating).
6) Evaluate the quality of the model by using the computeError() function to compute the error between the predicted ratings and the actual ratings in validationRDD.
7) Use the bestRank=8 to create a model for predicting the ratings for the test dataset and then we will compute the RMSE.

Execution Process:

1) Put the 20M dataset to HDFS.
2) Use command: spark-submit RecommendMovie20M.py ratings.csv

Python Code:

```python
from __future__ import print_function
import sys
from pyspark import SparkContext
import re
import time
import math
from pyspark.mllib.recommendation import ALS

def get_ratings_tuple(entry):
    """ Parse a line in the ratings dataset
    Args:
        entry (str): a line in the ratings dataset in the f
        orm of UserID::MovieID::Rating::Timestamp
    Returns:
        tuple: (UserID, MovieID, Rating, Timestamp)
    """
    items = entry.split(',')
    return int(items[0]), int(items[1]), float(items[2]),long(items[3])


def computeError(predictedRDD, actualRDD):
    """ Compute the root mean squared error between predicted and actual
    Args:
        predictedRDD: predicted ratings for each movie and each user
        where each entry is in the form (UserID, MovieID, Rating)
        actualRDD: actual ratings where each entry is in the form
        (UserID, MovieID, Rating)
    Returns:
        RSME (float): computed RSME value
    """
    # Transform predictedRDD into the tuples of the form ((UserID, MovieID), Rating)
    predictedReformattedRDD = predictedRDD.map(lambda (UserID, MovieID, Rating):
    ((UserID, MovieID), Rating))

    # Transform actualRDD into the tuples of the form ((UserID, MovieID), Rating)
    actualReformattedRDD = actualRDD.map(lambda (UserID, MovieID, Rating):
    ((UserID, MovieID), Rating))

    # Compute the squared error for each matching entry (i.e., the same (User ID,
    # Movie ID) in each RDD) in the reformatted RDDs using RDD transformtions
    # do not use collect()
    squaredErrorsRDD = (predictedReformattedRDD
                         .join(actualReformattedRDD)
                        .map(lambda ((UserID, MovieID), (PredRating, ActRating)):
                        (PredRating - ActRating)**2))

    # Compute the total squared error - do not use collect()
    totalError = squaredErrorsRDD.reduce(lambda a,b: a+b)

    # Count the number of entries for which you computed the total squared error
    numRatings = squaredErrorsRDD.count()

    # Using the total squared error and the number of entries, compute the RSME
    return math.sqrt(1.0*totalError/numRatings)


if __name__ == "__main__":
    #Print error message when the command is wrong.
    if len(sys.argv) != 2:
        print("Usage: RecommendMovie20M.py ratings.csv", file=sys.stderr)
        exit(-1)
    sc = SparkContext(appName="RecommendMovies")
    #Create a SparkContext to store dat files.
    numPartitions = 2
    rawRatings = sc.textFile(sys.argv[1])
    #extract header from rating.csv
    header = rawRatings.first()
    print ("-----------------------------")
    print ('\nHeader: %s\n' % header)
    #filter out header
```

```
    rawRatings = rawRatings.filter(lambda x:x !=header).repartition(numPartitions)

    ratingsRDD = rawRatings.map(get_ratings_tuple).cache()
    print ("----------------------------")
    print ('\nRatings: %s\n' % ratingsRDD.take(3))

#Part 2: Collaborative Filtering
    #Sort the ratings data in ascending order of timestamp (past to present)
    sortedRatingsRDD = ratingsRDD.sortBy(lambda r: r[3])
    print ("----------------------------")
    print ('\nSorted Ratings: %s\n' % sortedRatingsRDD.take(3))
    noTimeSortedRatingsRDD = sortedRatingsRDD.map(lambda (UserID, MovieID,
    Rating, Timestamp): (UserID, MovieID, Rating))

    #The training set contains the first 80% of the original dataset.
    #The validation set contains the next 10% of the original dataset.
    #The test set contains the last 10% of the original dataset.
    #To randomly split the dataset into the multiple groups, we can use the pySpark
    #randomSplit() transformation.
    #randomSplit() takes a set of splits and seed and returns multiple RDDs.
    trainingRDD, validationRDD, testRDD = noTimeSortedRatingsRDD
                                .randomSplit([8, 1, 1], seed=0L)
    print ('\nTraining: %s, validation: %s, test: %s\n' %
    (trainingRDD.count(),validationRDD.count(),testRDD.count()))

    # For the prediction step, create an input RDD, validationForPredictRDD,
    # consisting of (UserID, MovieID) pairs that we extract from validationRDD.
    validationForPredictRDD = validationRDD.map(lambda (UserID, MovieID, Rating)
    : (UserID, MovieID))
    trainingForPredictRDD = trainingRDD.map(lambda (UserID, MovieID, Rating)
    : (UserID, MovieID))

    seed = 5L
    iterations = 5
    regularizationParameter = 0.1
    ranks = [4, 8, 12]
    training_errors = [0, 0, 0] #training error
    validation_errors = [0, 0, 0]
    err = 0
    tolerance = 0.03
    training_time = [0,0,0];
    validation_time = [0,0,0];

    minError = float('inf')
    bestRank = -1
    bestIteration = -1

    for rank in ranks:
        train_start = time.time();
        #Training
        model = ALS.train(trainingRDD, rank, seed=seed, iterations=iterations,
        lambda_=regularizationParameter)
        training_time[err] = time.time()-train_start
        #Training RMSE
        tra_predictedRatingsRDD = model.predictAll(trainingForPredictRDD)
        #Use the Root Mean Square Error (RMSE) or Root Mean Square Deviation
        #(RMSD) to compute the error of each model.
        tra_error = computeError(tra_predictedRatingsRDD, trainingRDD)
        training_errors[err] = tra_error

        validation_start = time.time()
        #Validation
        val_predictedRatingsRDD = model.predictAll(validationForPredictRDD)
        validation_time[err] = time.time()-validation_start
        #Validation RMSE
        val_error = computeError(val_predictedRatingsRDD, validationRDD)
        validation_errors[err] = val_error
        if val_error < minError:
            minError = val_error
            bestRank = rank
```

```
        err += 1

# Use the bestRank=8 to create a model for predicting the ratings
# for the test dataset and then we will compute the RMSE.
myModel = ALS.train(trainingRDD, 8, seed=seed, iterations=iterations,
lambda_=regularizationParameter)
testForPredictingRDD = testRDD.map(lambda (UserID, MovieID, Rating)
: (UserID, MovieID))
test_start = time.time()
predictedTestRDD = myModel.predictAll(testForPredictingRDD)
# Test error
testRMSE = computeError(testRDD, predictedTestRDD)
# Test time
test_time = time.time() - test_start

i=0;
for rank in ranks:
    print ("-----------------------------")
    print ('\nFor rank %s : \n' % rank)
    print ('the training error is %s\n' % training_errors[i])
    print ('the training time is %s\n' % training_time[i])
    print ('the validation error is %s\n' % validation_errors[i])
    print ('the validation time is %s\n' % validation_time[i])
    i += 1

print ("-----------------------------")
print ('\nThe best training model was trained with rank %s\n' % bestRank)
print ('\nThe model had a RMSE on the test set of %s\n' % testRMSE)
print ('\nThe validation time is %s\n' % test_time)
```

Part II (2 bonus points): Run Spark on MovieLens 20M Dataset.

Perform the same tasks as in Part I. Compare the results with those in Part I. Fill in the following table.

Best model on 1M dataset			
	training	validation	test
RMSE	0.8515	0.8841	0.8860
Time (sec)	8.0841	1.2714	1.2673
Best model on 20 M dataset			
	training	validation	test
RMSE	0.7821	0.8130	0.8208
Time (sec)	76.4633	25.3092	26.5362