Business Intelligence Group 20 - Assignment 2

Bauer, Michael 01351159 | Kahsai, Solomon 01326114 | Sederl, Johannes 01225759

188.429 - 2017W

1 MapReduce product review analysis in Java

1.1 General Remarks

We chose the data sets from the categories "Musical Instruments", "Health and Personal Care" and "CDs and Vinyl" for the sentiment analysis. Unfortunately, the Cloudera VM did not run sufficiently on our local machines, so we installed hadoop locally on our machines instead. For this task, the default configuration as single-node installation was used to run hadoop in local (standalone) mode. The correct usage of the MapReduce job is given in the according README file.

1.2 Discussion

The implementation of MapReduce is divided into three Java files. The map part is implemented in the class `Map`, the reduce part is defined in the class `Reduce` and the main class which contains all the job configuration is `SentimentAmazon`. In `Map`, each line of the given JSON files is provided as input and the product id (field "asin") and the review text (field "reviewText") are emitted as output of the `map` method. Therefore the input for the method `reduce` in `Reduce` is the product id as key and a list with all review texts for the according product id as value.

There are 2 nested iterations in `reduce`. The first loop iterates over each review text and the second loop iterates over each word in a given review text. Then, each word is compared to a set of positive and negative words. If the current word is contained in the set of positive words, a counter for positive occurrences is incremented. Respectively, a counter for negative occurrences is incremented if the current word belongs to the set of negative words. Once each word has been checked against these sets, the sentiment score is calculated by using the given formula. The product id and the calculated sentiment score are eventually emitted as the output of `reduce` and written to the output file.

The `map` method is generally invoked for every input block. In this case, an input block is represented by one line in the JSON files, so there are as many invocations of `map` as there are lines in the given input files. The method `reduce` is called for each product id, so if there are `N` unique products in the data set then there are `N` invocations of `reduce`.

Most of the processing part is done in the method `reduce` during the reduce phase of the algorithm. This is due to the 2 loops which iterate over each word in each review text for a given product. In comparison, the map part is fundamentally cheaper because only operations with constant complexity are used.

If the job is distributed among multiple machines linear speed-up can be achieved. If this is the case, then the methods `map` and `reduce` can operate on different chunks of the data.

1.3 Results

The sentiment score is positive for the majority of the products in all chosen categories. This is due to the fact that most of the product reviews have a high rating, which is reflected in the textual reviews, and the basic algorithm used for calculating the sentiment score. The sentiment scores for the first 15 products of each category are listed below.

Category "Musical Instruments"

#	product id	sentiment score
1	1384719342	0.5
2	B00004Y2UT	0.02702702702702703
3	B00005ML71	0.8333333333333334
4	B000068NSX	0.45454545454545453
5	B000068NTU	0.5238095238095238
6	B000068NVI	0.68
7	B000068NW5	0.38016528925619836
8	B000068NZC	0.375
9	B000068NZG	0.36585365853658536
10	B000068O1N	0.6153846153846154
11	B000068O3D	0.36363636363636365
12	B000068O3X	0.5053763440860215
13	B000068O4H	0.5151515151515151
14	B000068O59	0.47619047619047616
15	B00006LVEU	0.4426229508196721

Category "Health and Personal Care"

#	product id	sentiment score
1	159985130X	0.6
2	1933622865	0.16923076923076924
3	1935009656	0.6842105263157895
4	3812028492	0.5757575757575758
5	7884890364	0.52
6	B00000J47L	0.2867132867132867
7	B00000J9DU	0.34782608695652173
8	B00000JHQ2	0.6666666666666666
9	B00000JHQ6	0.3184713375796178
10	B00000JHQC	0.46153846153846156
11	B00002N6SN	0.5267175572519084
12	B00002N88C	0.625
13	B00002N8QW	0.14285714285714285
14	B00002NC1L	0.5111111111111111
15	B00003IE4E	0.3333333333333333

Category "CDs and Vinyl"

#	product id	sentiment score
1	0307141985	0.55
2	073890015X	0.4573643410852713
3	0738900370	0.6052631578947368
4	0738900672	0.43768115942028984
5	0738919039	0.43529411764705883
6	0738920363	0.6470588235294118
7	0738921475	0.5454545454545454
8	0739060287	0.625
9	0760135002	0.1937984496124031
10	0767804341	0.6150943396226415
11	0767816641	0.2786885245901639
12	076783822X	0.5272727272727272
13	0769716903	0.647887323943662
14	0769720226	0.7
15	0769720323	0.10144927536231885

2 MovieLens dataset analysis with Hive

2.1 General Remarks

Running Cloudera VM on any of our personal machines was not possible due to insufficient RAM. We chose to use Amazon's Web Services, specifically Elastic Map Reduce (emr-5.10.0) and created a cluster containing 1x Master ('m4.large') and 2x Core ('m4.large'). Appllications included were the following:

- Core Hadoop: Hadoop 2.7.3 with Ganglia 3.7.2
- Hive 2.3.1
- Hue 4.0.1
- Mahout 0.13.0
- Pig 0.17.0
- Tez 0.8.4

Amazon's emr-5.x uses TEZ as the default execution engine - instead of MapReduce. We connected to the cluster by setting up a SSH-connection through PUTTY and ran queries directly in the shell (i.e. Hue was not used). We uploaded the datasets to Amazon's S3 storage service for loading data into Hive tables.

2.2 Hive Load Statements

After declaring the schema for the five tables, .csv-files are moved to locations corresponding to Hive tables. But: both schema and data are kept separately. Data is not checked during the LOAD-process whether e.g. datatypes match. Only when querying the data (e.g. SELECT * FROM exampleTable) one may observe type mismatch when the result set of a certain column only lists NULL.
For loading data the table 'movies' was special: As the movie title in movies.csv may contain the same delimiters (such as comma) used for field delimiter as well it was necessary to use SerDe CSV. Using SerDe CSV results in columns of type STRING, although types were specified differently in create-statements. Consequently, we created a view from the moviesV1 table and cast movieID to INT and split the Genres-STRING into an ARRAY<STRING>.

2.3 Questions

1. How many movie ratings are there in total in the dataset?
   26,024,289 User ratings.

2. How many movies in the dataset belong to the "Horror" genre?
   There are 4,448 movies with genre "Horror".
   (sidenote: checking the movies.csv file manually = searching for "Horror" yields 4.501 matches. However, in some cases the movie title contains the word "Horror".)

3. Which are the 10 most frequently assigned tags (by users, i.e., from the tags table)?

No	Tag	Frequency
1	sci-fi	8040
2	atmospheric	5240
3	action	5065
4	comedy	4713
5	based on a book	4473
6	surreal	4324
7	twist ending	4073
8	funny	3840
9	BD-Reducer	3420
10	classic	3380

4. Which 10 movies were the most controversial in 2010 (i.e., had the highest variance in ratings between 2010/01/01 and 2010/12/31)?

No	Title	Variance
1	Kids of the Round Table (1995)	5.0625
2	Clifford's Really Big Movie (2004)	5.0625
3	Sudden Fear (1952)	5.0625
4	I Got the Hook Up (1998)	5.0625
5	Lady Death (2004)	5.0625
6	Ethan Mao (2004)	5.0625
7	Ranma $\frac{1}{2}$: Big Trouble in Nekonron, China (Ranma $\frac{1}{2}$: Chûgoku Nekonron daikessen! Okite yaburi no gekitô hen) (1991)	5.0625
8	Charming Mass Suicide, A (Hurmaava joukkoitsemurha) (2000)	5.0625
9	Delgo (2008)	5.0625
10	Life Is Rosy (a.k.a. Life Is Beautiful) (Vie est belle, La) (1987)	4.0

5. Which movies (titles) are the 10 most frequently tagged and how often have they been

No	Movie	Frequency (Tags)
1	Star Wars: Episode IV - A new Hope (1977)	9204
2	Pulp Fiction (1994)	4454
3	Inception (2010)	3972
4	Shwashank Redempteion, The (1994)	3355
5	Fight Club (1999)	3331
6	Matrix, The (1999)	3116
7	Interestellar (2014)	2918
8	Forrest Gump (1994)	2754
9	Memento (2000)	2172
10	Eternal Sunshine of the Spotless Mind (2004)	2087

6. Which 15 movies (titles) have been most frequently tagged with the label "sci-fi"?

No	Movie
1	Star Wars: Episode IV - A new Hope (1977)
2	Matrix, The (1999)
3	Interestellar (2014)
4	Inception (2010)
5	Avatar (2009)
6	Blade Runner (1982)
7	Ex Machina (2015)
8	Star Wars: Episode V - The Empire Strikes Back (1980)
9	Alien (1979)
10	2001: A Space Odyssey (1968)
11	Fifth Element, The (1997)
12	District 9 (2009)
13	Edge of Tomorrow (2014)
14	Star Wars: Episode VI - Return of the Jedi (1983)
15	The Martian (2015)

7. Which are the 10 best-rated movies (on average; list titles) with more than 500 ratings?

No	Movie	rating (avg)
1	Planet Earth	4.478
2	Shawshank Redemption, The (1994)	4.429
3	Godfather, The (1972)	4.339
4	Usual Suspects, The (1995)	4.300
5	Schindler's List (1993)	4.266
6	Godfather: Part II, The (1974)	4.263
7	Seven Samurai (Schchinin no samurai) (1954)	4.255
8	Rear Window (1954)	4.232
9	12 Angry Men (1957)	4.231
10	Fight Club (1999)	4.230

8. Which are 10 highest-rated "Drama" movies with more than 10 ratings?
totalRatings = movies with '5.0' ratings

No	Movie	totalRatings
1	Shawshank Redemption, The (1994)	45,511
2	Pulp Fiction (1994)	35.718
3	Forrest Gump (1994)	30,280
4	Schindler's List (1993)	28,934
5	Godfather, The (1972)	27,288
6	Fight Club (1999)	22,200
7	Braveheart (1995)	19,453
8	American Beauty (1999)	19,453
9	Fargo (1996)	18,036
10	Lord of the Rings: The Return of the King, The (2003)	17,387

9. What are the 15 most relevant genome tags for the movie "Four rooms" (movieId=18)?

No	GenomeTag
1	Off-beat comedy
2	hotel
3	storytelling
4	weird
5	multiple storylines
6	stylish
7	tarantino
8	original
9	great ending
10	dark humor
11	twists & turns
12	dialogue
13	great acting
14	absurd
15	comedy

10. Which are the 10 most relevant movies for Vienna (i.e., with the highest genome tag relevance rating for the tag "vienna")?

No	Movie	totalRatings
1	Third Man, The (1949)	0.98
2	Johnny Guitar (1954)	0.96
3	Before Sunrise (1995)	0.95
4	Before Sunset (2004)	0.95
5	Before Midnight (2013)	0.91
6	Woman in Gold (2015)	0.89
7	Night Porter, The (Protiere di notte, Il) (1974)	0.88
8	Amadeus (1984)	0.84
9	Illusionist, The (2006)	0.83
10	Foreign Affair, A (1948)	0.70

2.4 Behind the scenes & Expectations

As described above TEZ execution engine was used. In essence, contrarily to Map reduce jobs TEZ firstly executes the plan, but does not yet read actual data from disk. Secondly, depending on the query once the plan is prepared, necessary data is gathered (types from Hive's metastore usually stored in a relational database, data from HDFS) and requried steps are performed. Finally, the output is written to a temporray HDFS file and presented to the user. In more detail: based on metadata (=schema) of Hive the query compiler sets up a (logical) plan of execution. Here Hive parses the query, checks types and creates a Directed Acyclic Graph (DAG). Important steps of optimization are to only use specific columns needed for the query and the reordering of joins, thus streaming larger tables and keeping smaller tables in memory.
The logical plan is then partitioned into several map-, reduce- and HFDS tasks.
(main source: A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive-a petabyte scale data warehouse using hadoop," in Data Engineering

(ICDE), 2010 IEEE 26th International Conference on, 2010, pp. 996–1005.)

For running our queries on a real cluster in parallel we expect that a linear speed-up may be reached by scaling out more nodes. As we were already working on a 'real cluster' - Amazon's Elastic Map Reduce resizing the cluster by increasing the number of task nodes would have been fairly simple.

3 Spark movie recommender

Objective of task: to obtain personal movie recommendations for each member in your group by training a collaborative filtering model. This works because of the preference information in the 20 million ratings in the MovieLens data set and a few of the personal movie ratings.

It will provide personal movie ratings that you are likely to find interesting too based on the ratings of users with similar preferences in the past.

In order to solve the collaborative filtering problem it implements an Alternating Least Squares (ALS) algorithm that can be run in parallel and keep factor matrices data in memory.

Implementation (Please extract the data into the datasets folder before running)

Implementation language: Python Data Structure: RDD

1. Load the full and small movie lens data sets into RDDs or DataFrames

Code snippet used:

```
# not usefull since its already downloaded and extracted

# let's load the raw ratings data. We need to filter out the header, included in each file
############
############
print('----------------------------------------')
print('filtering out the header, included in each file...')
print('----------------------------------------')
small_ratings_file = os.path.join(datasets_path, 'ml-latest-small', 'ratings.csv')
small_ratings_raw_data = sc.textFile(small_ratings_file)
small_ratings_raw_data_header = small_ratings_raw_data.take(1)[0]
############
############
```

2. Parse the data into a suitable format for the ALS algorithm via Spark transformations.

```
# Now we can parse the raw data into a new RDD
############
############
print('----------------------------------------')
print('parsing the raw data into a new RDD...')
print('----------------------------------------')
small_ratings_data = small_ratings_raw_data.filter(lambda line: line != small_ratings_raw_data_header) \
    .map(lambda line: line.split(",")).map(lambda tokens: (tokens[0], tokens[1], tokens[2])).cache()

############
############
```

- Load and parse for the small movies data set as well

```
# We proceed in a similar way with the movies.csv file
###########
###########
print('---------------------------------------')
print('now all the above for movies...')
print('---------------------------------------')
small_movies_file = os.path.join(datasets_path, 'ml-latest-small', 'movies.csv')

small_movies_raw_data = sc.textFile(small_movies_file)
small_movies_raw_data_header = small_movies_raw_data.take(1)[0]

small_movies_data = small_movies_raw_data.filter(lambda line: line != small_movies_raw_data_header) \
    .map(lambda line: line.split(",")).map(lambda tokens: (tokens[0], tokens[1])).cache()
###########
###########
```

3. Split the data into a training and a testing set via a Spark transformations.

```
# We need first to split it into
# -------train-------------------
# -------validation--------------
# -------and test datasets-------
###########
###########
training_RDD, validation_RDD, test_RDD = small_ratings_data.randomSplit([6, 2, 2], seed=0)
validation_for_predict_RDD = validation_RDD.map(lambda x: (x[0], x[1]))
test_for_predict_RDD = test_RDD.map(lambda x: (x[0], x[1]))
###########
###########
```

4. Use the small data set to determine the best ALS parameters (optional).

```
from pyspark.mllib.recommendation import ALS
import math

seed = 5
iterations = 10
regularization_parameter = 0.1
ranks = [4, 8, 12]
errors = [0, 0, 0]
err = 0
tolerance = 0.02

min_error = float('inf')
best_rank = -1
best_iteration = -1
for rank in ranks:
    model = ALS.train(training_RDD, rank, seed=seed, iterations=iterations,
                      lambda_=regularization_parameter)
    predictions = model.predictAll(validation_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
    rates_and_preds = validation_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
    error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1]) ** 2).mean())
    errors[err] = error
    err += 1
    print('For rank %s the RMSE is %s' % (rank, error))
    if error < min_error:
        min_error = error
        best_rank = rank
print('---------------------------------------')
print('The best model was trained with rank %s' % best_rank)
```

We got the the parameters from the tutorial.

5. Run the ALS algorithm with the identified parameters on the full dataset to train the final model.

After loading and parsing the complete dataset

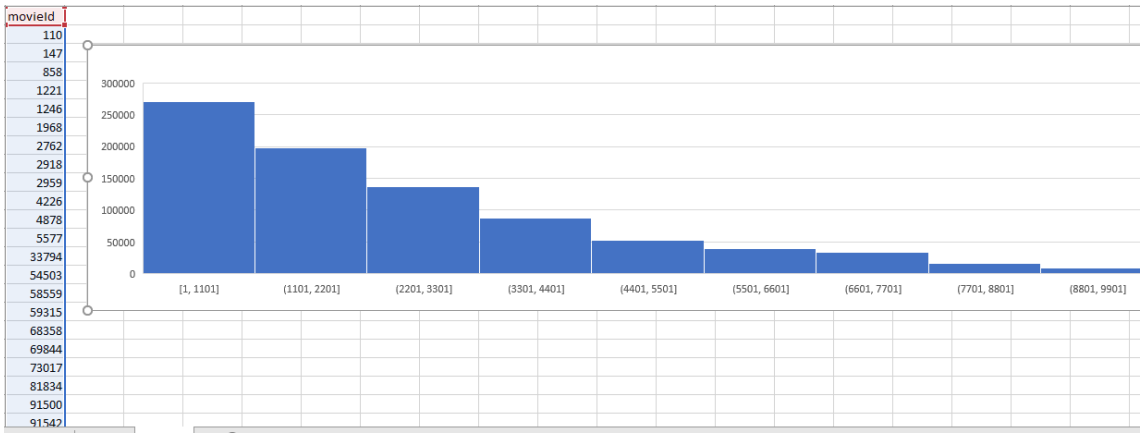
```
print('-----------------------------------')
print('training the ALS model using all the paramers selected in the small dataset...')
print('-----------------------------------')
from time import time

t0 = time()
new_ratings_model = ALS.train(complete_data_with_new_ratings_RDD, best_rank, seed=seed,
                              iterations=iterations, lambda_=regularization_parameter)
tt = time() - t0

print('-----------------------------------')
print("New model trained in %s seconds" % round(tt, 3))
print('-----------------------------------')
# It took some time. We will need to repeat that every time a user
# adds new ratings. Ideally we will do this in batches, and not for every
# single rating that comes into the system for every user.
###########
###########
```

6. For each group members, use at least 15 ratings of frequently rated movies to obtain the top 15 recommended movies.

- Checking for frequency of the movies revealed that movies with ID 1 till 1101 had the highest level of occurring. Hence why each group member choose to pick 15 movies from within those.



1) Then group member 1 is assigned the user ID 99999991, 2 is assigned 99999992, and 3 is assigned 99999993

2) Then each member gets to choose frequently rated movies. 15 each.

3) Then we add these choices for each user in its own RDD

4) Then we train the ALS with ALS.train with each user's RDD

5) Then we use the input RDD of unrated movies of the user and input RDD to predict new ratings for the movies that user rated

6) Then we take the recommendation for a the user and transfer mit so it contains pairs of the form (Movie ID, Predicted Rating)

7) Then we do flattening which basically means the result will contain (Title, Rating, Ratings Count)

8) Finally we get the highest rated recommendations for that user.

We repeated the process from 2-8 for the three group members

Code snippet for steps 2-8