Assignment 2: Big Data Technologies

Deadline

Please upload all your results **before December 17, 2017**!

Submission guidelines

Upload a single .zip/.tar.gz file named

`VU_BI_2017W_Assignment2_<group no>.zip/.tar.gz`

that contains all your deliverables, i.e.:

1. A single PDF document that describes your solution for all parts of the assignment.

   Filename: `VU_BI_2017W_Assignment2_<group no>.pdf`

2. The solution directories for each of the three exercises
   (see the deliverables outlined for each exercise for details)

   Hence, the directory layout of the zipped solution should be the following (obviously, replace <groupNo> with your group number):

   ```
   1_MapReduce_SentimentAmazon_<groupNo>/
   2_Hive_MovieLensQuerying_<groupNo>/
   3_Spark_MovieRecommendation_<groupNo>/
   VU_BI_2017W_Assignment2_<group no>.pdf
   ```

Please make sure to list all team members on the title page and the page headers of your solution document. It's sufficient to upload the final solution of the group once (i.e., one person for all members of the group).[1]

Questions

Please post general questions in the TUWEL discussion forum of the course. You can also discuss any problems and issues you are facing there. We appreciate if you help other students out with general problems or questions regarding the tools used (and may take that into account in case you are short a few points for a better grade). For obvious reasons, however, please do not post any solutions there.

You can also contact me directly at:

`elmar.kiesling@tuwien.ac.at`

Please use subject line `BI_2017_<your subject>`.

[1] In case of multiple uploads, only the last submission will be considered.

Assignment 2: Big Data Technologies

Learning Objectives

The goal of this hands-on is to acquaint yourself with big data processing environments and experiment with technologies for large-scale data processing.

The first exercise will focus on the "nuts and bolts" of Apache Hadoop programming. In this part, you will learn how to implement and run MapReduce programs using the Hadoop Java framework.

The second exercise will move up the ladder on the Hadoop stack and cover the Hive data warehouse infrastructure, which provides an SQL-like interface for querying large data stored in a distributed filesystem.

The final part of this hands-on will cover Apache Spark and show you how to efficiently implement data-intensive iterative algorithms in parallel programs and execute them on large data sets.

Preliminaries: Lab Setup

You can solve all parts of this assignment on your local machine. For the first two exercises, the easiest way to set up a local Hadoop instance is to download a preconfigured Hadoop stack in a virtual machine image. We tested the exercise on Cloudera's Quickstart VM[2], which you can download from [1]. Note that the Cloudera VM requires a 64-bit host OS (like most virtual machines provided by other Hadoop distributors). Other Hadoop distributions or a local installation on your native OS should work as well.

For the third exercise, it is advisable to run Spark natively on your local machine. Cloudera's Quickstart VM comes with Apache Spark, but according to our experience, it requires a lot of memory and usually does not work out of the box. As we will not be using HDFS in this final exercise, a local standalone Spark setup will be sufficient, dramatically reduce resource requirements (particularly RAM), and still allow you to leverage all the available processing cores on your local machine in parallel. You can download Apache Spark from [2] or install it using your package manager of choice.

Resources

[1] http://go.cloudera.com/vm-download
[2] https://spark.apache.org/downloads.html

[2] After booting into the VM, click "Launch Cloudera Express". In Cloudera Manager, make sure that the services required for the exercieses – i.e., HDFS, YARN, Hive, Hue (if you want to use the graphical interface) - are running.

Assignment 2: Big Data Technologies

I. MapReduce product review analysis in Java

The aim of this exercise is to obtain hands-on MapReduce programming experience in Java. You will implement a simple MapReduce program to efficiently conduct a very basic sentiment analysis on product review texts. You will be using a single machine virtual machine cluster to develop and deploy your Map Reduce programs. Therefore, we'll use "small" subsets of Amazon product review data for testing purposes. However, you should design for near-linear scale-up so that your algorithms can be deployed on a large cluster to process very large amounts of data efficiently (e.g., the > 140 million Amazon reviews in the full data set).

Prerequisites

Before you get started, please choose and download data sets for three product categories from

`http://jmcauley.ucsd.edu/data/amazon/`

We will be using the "5-core" dense subsets in this exercise, please download the "small" subsets available for experimentation.

Once you have downloaded the three json ("5-core") files for the three chosen product categories, copy them into your MapReduce programming environment (e.g., `/home/cloudera/data/`), extract them, and copy them to hdfs, i.e.:

```
hadoop fs -mkdir /amazon
hadoop fs –copyFromLocal <path> /amazon
```

Sentiment analysis

Conduct a simple sentiment analysis by counting up the positive and negative words in all the review texts for each product and calculate the overall share of positive words. Hence, the overall sentiment score you should calculate for each product is defined as

```
sentiment = (positive - negative) / (positive + negative)
```

You will find positive (`pos-words.txt`) and negative (`neg-words.txt`) word lists available for download on TUWEL.

Your MapReduce job should calculate sentiment scores for all products in a given <category>_5.json file. Once you have implemented and tested your program, calculate the sentiment value for each product in the 5-core data sets of the three categories you have chosen. Include the sentiment scores of the first 15 products in each chosen data set in your solution document.

Assignment 2: Big Data Technologies

Hints:

- You can find a tutorial for sentiment analysis using MapReduce at [1].

Deliverables:

1. `1_MapReduce_SentimentAmazon_<groupNo>/source`
   Maven project with complete source code.

2. `1_MapReduce_SentimentAmazon_<groupNo>/AvgReviewScore.jar`
   Compiled jar file that can be run as a MapReduce job.

3. Your solution document should include:
   - A brief documentation of your implementation and a discussion of its scaling characteristics. In particular, discuss the following questions:
     - How many invocations of the Map and Reduce methods are there?
     - In which phase of the process is most of the runtime spent?
     - What speedup would you expect from distributing the job among more machines?
   - The sentiment scores of the first 10 products in each of your chosen categories.

Resources

[1] *Example: Sentiment Analysis Using MapReduce Custom Counters*
https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_example_4_sentiment_analysis.html

[2] *MapReduce Tutorial*:
https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Dataset citation

Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel, SIGIR, 2015

Inferring networks of substitutable and complementary products
J. McAuley, R. Pandey, J. Leskovec, Knowledge Discovery and Data Mining, 2015

Assignment 2: Big Data Technologies

II. ⏎ MovieLens dataset analysis with Hive

The goal of this exercise is to query a moderately large movie review data set using Hive. This data set includes 26 million ratings and 750,000 tags applied to 45,000 movies by 270,000 users and includes a dense matrix of calculated relevance scores ("genome tags"). For details about the data set, cf. [D1].

Prerequisites

1) Get the MovieLens *full* dataset (224 MB compressed; version updated 8/2017) from

   `http://grouplens.org/datasets/movielens/`

   and extract it into your Hadoop environment.[3]

2) Strip the header line from each file[4]

```
sed -i 1d <files>
```

3) Copy the files into hdfs

```
hadoop fs -mkdir /movieLens
```

4) Create a new database in hive (either by invoking `hive` on the shell or using the web-based Hive query editor[5]):

```
CREATE DATABASE movieLens;
USE movieLens;
```

5) Create the tables:

   - `movies` (for movies.csv)
     Schema: movieId, title, genres
   - `tags` (for tags.csv)
     Schema: userId, movieId, tag, timestamp
   - `ratings` (for ratings.csv)
     Schema: userId, movieId, rating, timestamp
   - `genome_scores` (for genome-scores.csv)
     Schema: movieId, tagId, relevance

[3] You can also download and use the small subset to develop and test your queries. However, make sure to include your final results on the 20M data set in your solution document.
[4] Alternatively, you can skip the line when creating your hive tables using `TBLPROPERTIES ("skip.header.line.count"="1")`
[5] `http://quickstart.cloudera:8888/notebook/editor?type=hive` on Cloudera's Quickstart VM; you may need to start the Hue server in Cloudera manager first.

- `genome_tags` (for genome-tags.csv)
  Schema: tagId, tag

Your table declarations should:

- use appropriate data types
- specify that the actual data will be stored as a text file
- specify that the data is in row format and comma separated

You can test your code in interactive mode, but when you hand in the results, your solution folder should contain your HiveQL `CREATE TABLE` statements in a hql file (`createMovieLensTables.hql`) that can be passed to Hive in batch mode

6) Load data from the text files to populate the respective tables you just created in the previous step. Place your HiveQL `LOAD DATA` statements in a hql file named `loadMovieLens.hql`. In your solution document, also describe your understanding of what happens behind the scenes here.

Queries

Run the following Hive queries:

1) How many **movie ratings** are there in total in the dataset?

2) How many movies in the dataset belong to the "Horror" genre?

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

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)?

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

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

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

8) Which are 10 highest-rated "Drama" movies with more than 10 ratings?

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

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

Assignment 2: Big Data Technologies

Deliverables

Place your HiveQL `SELECT` statements in hql files named `hive-query_<no>.hql`.

- `2_Hive_MovieLensQuerying_<groupNo>/createMovieLensTables.hql`
- `2_Hive_MovieLensQuerying_<groupNo>/loadMovieLens.hql`
- `2_Hive_MovieLensQuerying_<groupNo>/hive-query_<queryNo>.hql`

In the solution document, provide:

- Results for all queries
- Description of your understanding of what happens behind the scenes (depends on what execution engine you use). Finally, comment on what scale-up you would expect when running your queries on a real cluster in parallel.

Resources

[1] Hive Wiki:
https://cwiki.apache.org/confluence/display/Hive/Home

[2] Hive Tutorial:
http://thinkbig-academy.s3.amazonaws.com/Strata2013/HiveTutorial/index.html

[4] Hive Language Manual:
https://cwiki.apache.org/confluence/display/Hive/LanguageManual

Dataset citation

[D1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), http://dx.doi.org/10.1145/2827872

III. ⏎ Spark movie recommender

In this exercise, you will get hands-on experience with Apache Spark by implementing a recommender model that, based on the movie lens data set used in the previous exercise, provides personalized movie recommendations. The focus of this exercise is not on MLlib and the ALS algorithm applied for collaborative filtering, but on getting to know Apache Spark as a programming environment for scalable, highly parallel data processing. After completing this exercise, you will have the basic skills needed to implement scalable data processing workflows. Data mining aspects will then be covered in the last part of the course.

Prerequisites

1) Lab Setup:
   The Cloudera Quickstart VM used in the previous exercises comes with Apache Spark. However, our experience is that it tends to not work (well) out-of-the-box and that it requires considerable configuration and resources. We therefore recommend a local Apache Spark installation. As we will not be using HDFS in this exercise, but load the data sets into RDDs (or data frames) from disk, a local standalone Spark setup will dramatically reduce the overhead and still allow you to leverage all the available processing cores of your local machine.
   You can download Apache Spark from `https://spark.apache.org/downloads.html` or install it using your package manager of choice[6].

2) Extract the latest MovieLens dataset used in the previous exercise (`http://files.grouplens.org/datasets/movielens/ml-latest.zip`) as well as the MovieLens 100.000 data set (`http://files.grouplens.org/datasets/movielens/ml-latest-small.zip`) and place them in a `datasets` folder on the same level as your solution folder on your local file system.

Objective

Building on the data used in the previous exercise, the goal in this final part is to obtain personal movie recommendations for each member in your group. To this end, you will train a collaborative filtering model that, based on the preference information in the 20 million ratings in the MovieLens data set and a few of your personal movie ratings finds movies that you are likely to find interesting based on the ratings of users with similar preferences in the past. Collaborative filtering using Alternating Least Squares (ALS) is an example of an iterative problem that usually cannot be implemented efficiently as a MapReduce job. Apache Spark allows you to efficiently run the iterative algorithm in parallel and keep the important data (the factor matrices) in memory.

[6] We use a brew installation of Apache Spark 2.2.0 and Python 2.7 on a mac for testing.

Assignment 2: Big Data Technologies

Implementation

To complete this exercise, you should implement the following steps[7]:

1. Load the full and small movie lens data sets into RDDs or DataFrames.
2. Parse the data into a suitable format for the ALS algorithm via Spark transformations.
3. Split the data into a training and a testing set via a Spark transformations.
4. Use the small data set to determine the best ALS parameters (optional).
5. Run the ALS algorithm with the identified parameters on the full dataset to train the final model.
6. For each group members, use at least 15 ratings of frequently rated movies to obtain the top 15 recommended movies.

You may choose your implementation language (Python, Scala) and data structure/API (RDDs, DataFrames) to your liking. You can find detailed tutorials for this exercise on the Web, including:

- a Python-based tutorial using RDDs, available as a Jupyter notebook [1]
- a Python-based tutorial using DataFrames [2]
- a Scala–based tutorial [3]

Deliverables

In your `3_movieRecommendation_<groupNo>` directory, submit your solution code (Jupypter notebook, Python script, or Scala program).

To allow us to replicate your solution, please make sure of the following:

- the script/notebook should assume that `sc` is the Spark context variable
- We assume that the movie lens small and complete datasets are loaded in the script from a `datasets` folder that sits on the same level as your `solution` folder on the local filesystem. Place the datasets in `datasets/ml-latest` and `datasets/ml-latest-small`, respectively.
- Do not rename the files (i.e., `movies.csv`, `ratings.csv` etc.) in the dataset folders

In the solution document, describe your solution approach and include the 15 movie ratings and the resulting top 15 recommended movies for each of your group members[8].

Resources

[1] GitHub: Jadianes/spark-movie-lens
https://github.com/jadianes/spark-movie-lens/blob/master/notebooks/building-recommender.ipynb

[2] MapR: Building a Recommendation Engine with Spark
https://mapr.com/ebooks/spark/08-recommendation-engine-spark.html

[3] Databricks Tutorial: Movie recommendation with Spark MLlib
http://bit.ly/2zGt8q9

[7] You can execute the steps interactively using one of the Spark shells, but make sure to hand in a complete implementation as a python script, Jupyter notebook, or Scala program.

[8] You may freely pick the movies you want to rate (obviously, movies that have been frequently rated will work better) and you may submit feigned ratings if you are embarrassed by your taste in movies ;).