Homework 4

Large-Scale Data Analysis

(individual submission)

Fabian Gieseke
Department of Computer Science
University of Copenhagen

2 June 2020

Submission Deadline:
14 June 2020, 23:59

This assignment is an **individual assignment**, i.e., you are supposed to solve the assignment on your own. In particular, the report and the code must be written completely by yourself!

Virtual Machine: We assume that you make use of the LSDA virtual machine for this assignment, i.e., all instructions provided are tailored to this system. For your submission, please make sure that your code runs on the virtual machine.

Deliverables: Please submit *two separate files* via Absalon: (1) a pdf file `answers.pdf` containing your answers and (2) a `code.zip` file containing the associated code as zip file (do

1 Analyzing Airline Data (50 pts)

We will analyze a dataset containing information about flights in the USA that stems from the *Bureau of Transportation Statistics*.[1]. In the zip file provided, you will find a directory `airline_data` containing the files `2016_1.csv`, ..., `2016_6.csv`. Each line of a file (except the headers) contains the flight date, the airline ID, the flight number, the origin airport, the destination airport, the departure time, the departure delay in minutes, the arrival time, the arrival delay in minutes, the time in air in minutes, and the distance between both airports in miles. Make use of your virtual machine and the Hadoop cluster that is already installed on it. For both subtasks, keep in mind to first activate your Python3 environment via `source .venvs/lsda/bin/activate` (execture from the home directory).

1. *Data Analysis with Hadoop:* You first need to start Hadoop by executing `./.start_dfs_yarn.sh` from your home directory. Note that both the `mapper.py` and the `reducer.py` need to be executable, which you can achieve via `chmod 755 mapper.py` and `chmod 755 reducer.py`. **Also make sure that the first line of your mapper.py/reducer.py files equals "#!/usr/bin/env python3".**
   Hint: You can quickly test the mappers and reducers via Unix Streaming, e.g., via:

   ```
   cat 2016*.csv | ./mapper.py | sort | ./reducer.py
   ```

   (a) *Total Departure Delay (5 pts):* Create a directory `airline_data` on the Hadoop cluster and copy all csv files from the local file system to this HDFS directory. Write a mapper and reducer that computes the total departure delay per airport in minutes (ignore all negative values for the delay, which indicate a departure before the scheduled time). What is the total departure delay for airport LAX?

[1] `https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time`

(b) *Maximal Departure Delay (5 pts):* Write a mapper and reducer that computes the maximal departure delay per airport in minutes. What is the maximal departure delay for airport DTW?

(c) *Mean and Standard Deviation for Departure Delays (10 pts):* Write a mapper and reducer that compute the mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and the standard deviation $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ for the departure delays $x_1, \ldots, x_n$ for each airline ID. Consider all delay values including the negative ones (treat missing values as 0). A reducer should output, for each airline ID, the associated mean and standard deviation. What is the mean and standard deviation for airline ID 20366?

(d) *Top-10 of Departure Delays (10 pts):* Write a mapper and reducer that compute the 10 most delayed flights for each airline ID. The reducer should output, for each airline ID, a list containing the delays for the 10 most delayed flights w.r.t. the departure time. What are the 10 most delayed flights for airline ID 20366?

Note: Write efficient mappers/reducers in the sense that only a constant amount of additional main memory is used by each of the workers. For instance, you can define a couple of "counters" (like for the word count example) or lists of constant size. An example for an inefficient reducer would be a reducer that simply collects all the departure delays per airport in a list to subsequently compute the total departure delay (this list could become very large depending on the amount of incoming data!). In doubt, feel free to get in touch with us!

2. *Data Analysis with Spark (20 pts, 5 pts for each of the subtasks):* Implement the four jobs for the Airline dataset in Apache Spark! As a starting point, make use of the Jupyter notebook `Airline Stats.ipynb`, which already creates an RDD based on the airline dataset. You are supposed to implement the tasks in a "similar" fashion as before, i.e., you are supposed to (only) make use of `map` and `reduceByKey` (do not simply make use of some Spark functions that yield, e.g., the mean and standard deviation). In doubt, feel free to ask via Absalon ;-).

   Note: You have to execute `source .activate_jupyter_pyspark` *before running* `pyspark`.

Deliverables

Include answers to the questions raised above to your write-up (`answers.pdf`). Provide, for each subtask, the mapper.py/reducer.py files in a subdirectory (e.g., `1a_total_pos_delay_departure`) of your overall source code archive. We recommend to user Unix streaming for testing your mappers and reducers. **Your are also supposed to run all jobs on the Hadoop cluster.** Please add the text output produced by each Hadoop job (e.g., "copy and paste" from the terminal) as well as the generated output files (e.g., `part-00000`) to your `code.zip`. For the Spark task, please add the modified `Airline Stats.ipynb` notebook to your code.

2 Landsat Data and Apache Spark (25 pts)

Next, we will make use of Apache Spark to analyze some data. Make sure that you have copied all the necessary files to your Hadoop distributed filesystem.

1. *Training Set Reduction (12.5 pts):* The first notebook `Landsat Training Set Reduction.ipynb` parses a relatively large training set (based on the Landsat satellite mission) containing 5 million training points (`landsat_train_small.csv`; make sure to copy this file to your HDFS into the newly created directory `landsat`). The training set is very unbalanced, i.e., most of the instances belong to one of the classes. Follow the instructions provided in the notebook to extract a stratified subsample of the data that you can then use to train a extra trees classifier on the driver node. Report the training accuracy that is produced at the end of the notebook. Note that the notebook generates a `model.save` file at the end, which you are supposed to use for the next subtask.

2. *Distributed Model Application (12.5 pts):* We have had a quick look at "broadcasting" machine learning models to the worker nodes via Spark. The notebook `Landsat Distributed Model Application.ipynb` makes use of this mechanism to apply the previously generated extra trees model generated in part 1. to a validation set (`landsat_validation`; must be made available in the directory `landsat` on your HDFS). In its current form, it resorts to the `map` transformation, which basically processes the incoming data line by line. Your task is to speed up the computations by making use of the `mapPartitions` transformation. Have a look at the documentation of this transformation.[2] Afterwards, define a new function `compute_predictions_chunk` to be used in combination with the `mapPartitions` transformation. At the moment, only 1% of the validation data is considered. Can you process the whole RDD? What is the induced classification accuracy when considering the whole validation RDD? Can you briefly explain why the new code is faster (3-4 lines)?

Deliverables

Include answers to the questions raised above to your write-up (`answers.pdf`). Add the modified notebooks to your code.zip.

3 Distributed Linear Regression with Apache Spark (25 pts)

We have briefly discussed how to conduct gradient descent for fitting a logistic regression model in a distributed fashion via Spark. You can find the correponding notebook on Absalon (`Logistic Regression Spark.ipynb`). This part of the exercise is about immplementing linear least-squares regression in a similar fashion!

1. Warm-Up: Follow the code given in the `Logistic Regression Spark.ipynb` notebook.

2. Next, have a look at the first part of the `Distributed_Linear_Least_Squares.ipynb` notebook. The class `LinearLeastSquaresRegression` computes a standard linear regression model (no distributed computations) and resorts to the "black box" optimizer `fmin_l_bfgs_b` of the `scipy.optimize` package. Note that it is also possible to only provide a function that computes the function values (i.e., the gradient is not necessarily needed). Thus, it is enough to provide a function to the optimizer that computes

$$F(\mathbf{w}) = \frac{1}{n}(\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{1}$$

   for a given $\mathbf{w} \in \mathbb{R}^d$ and data $\mathbf{X}$ (matrix containing the data points) and $\mathbf{y}$ (vector containing the labels).

3. Implement this approach such that the linear least-squares models are computed in a distributed fashion via `map` and `reduce`. More precisely, finalize the implementation of `_function` of the class `LinearLeastSquaresRegressionSpark`, which should return, for a given $\mathbf{w} \in \mathbb{R}^d$, the function value $F(\mathbf{w})$. Note that you can assume that the single vector $\mathbf{w}$ is small enough to be handled by the driver node that executes the optimizer. However, you implementation should be able to deal with a very large number n of data points stored in matrix $\mathbf{X}$ and $\mathbf{y}$, respectively. Briefly describe the individial steps of your distributed function evaluation (5-10 lines). Add the final figure that is generated to you write-up.

Deliverables

Include answers to the questions raised above to your write-up (`answers.pdf`). Add the modified notebook to your code.zip.

[2] See `https://spark.apache.org/docs/latest/rdd-programming-guide.html`.