732A54 - Big Data Analytics

Lab compendium

(Spark and Spark SQL)

Description and Aim

In the lab exercises you will work with the historical meteorological data from the Swedish Meteorological and Hydrological Institute (SMHI). Specifically, you will work with air temperature readings and precipitation readings from 812 meteorological stations in Sweden[1]. In these exercises, you will work with both Spark and Spark SQL.

After completing these two labs you will have basic knowledge of the programing environment, techniques and APIs for running both Spark and Spark SQL. You will work on exercises with Spark and Spark SQL and thus will be able to compare the differences between the two approaches.

Data

The data includes air temperature and precipitation readings from 812 stations in Sweden. The stations include both currently active stations as well readings from historical stations that have been closed down. The latest readings available for active stations are from October 10, 2016.

The air temperature/precipitation readings are hourly readings, however some stations provide only one reading every three hours.

The provided files are prepared csv files with removed headers (zip file available at: https://www.ida.liu.se/~732A54/lab/data.zip[2]). Values are separated with ;. Some files are too big to be read using some text editors. Therefore use either python to read the files or bash commands such as `tail` and `more` to get an overview of a file's content. Provided files:

- *temperature-readings.csv* - ca 2 GB
- *precipitation-readings.csv* - ca 660 MB
- *stations.csv*
- *stations-Ostergotland.csv*
- *temperatures-big.csv* - ca 16 GB
    - already available on `hdfs` under: `/user/common/732A54/temperatures-big.csv`

[1] If interested in other readings please check: http://opendata-catalog.smhi.se/explore/

[2] To unzip the files, use: `unzip -a data.zip`

Headers for *temperature-readings.csv*

Station number	Date	Time	Air temperature (in °C)	Quality[3]

Headers for *precipitation-readings.csv*

Station number	Date	Time	Precipitation (in mm)	Quality[3]

Headers for *stations.csv*

Station number	Station name	Measurement height	Latitude	Longitude	Readings from (date and time)	Readings to (date and time)	Elevation

Headers for *stations-Ostergotland.csv*

These are the same as in stations.csv. The file contains only stations in Östergotland.

Headers for *temperatures-big.csv*

These are the same as in temperature-readings.csv. The file is essentially a concatenation of 8 copies of *temperature-readings.csv* files.

If you notice any mistakes in the dataset/lab compendium or have any comments please contact the course assistants.

[3] G - controlled and confirmed values, Y - suspected or aggregated values

Working on your labs

Cluster setup and logging in

In the labs you will work on the Hadoop cluster set up at the National Supercomputer Centre (NSC). NSC's experimental Heffa lab cluster was built from old nodes from the NSCs 'matter' supercomputer, which was decommissioned. Some details about the nodes are provided below.

System server:	Compute / Login / Analysis nodes:
Hardware: ProLiant DL180 G6 CPU: 2 x 4-core Intel(R) Xeon(R) CPU E5520 @ 2.27GHz Hadoop software: - hadoop hdfs namenode - yarn resource manager - yarn proxyserver - mapreduce historyserver - spark history server	Number of nodes: 11 (of which 2 are login nodes) Hardware: HP SL170z G6 CPUs: 2 x 4-core Intel Xeon E5520 @ 2.2GHz Interconnect: gigabit ethernet Hadoop distributed storage: 9 x 500 GB. Memory: 9 x 4 GB Hadoop software: - hadoop hdfs datanode - hadoop client software (map reduce, etc.) - hadoop yarn nodemanager - spark client software

In the labs you will work with Spark and Spark SQL v. 1.6.0. We will make use of Spark Python API (PySpark) which provides a python programming environment for Spark and Spark SQL. Make use of PySpark's programming guide and API's documentation to get an overview of available functions.[4]

The server is available at `heffa.nsc.liu.se` (log in using your NSC accounts). There are two ways of working on your labs. The first one is by combining `ssh` and `scp`. In this case, you work on your files locally, then using `scp` you copy the files to heffa, and finally using `ssh` you run the jobs. **The first time you log in after receiving your account details, you must log in using ssh.** To do this, use the following command in the terminal:

`ssh` `username@heffa.nsc.liu.se` where username is your NSC username (not the LiU one)

Another easier and recommended approach is to make use of ThinLinc[5] which is a remote desktop solution. NSC has set up a ThinLinc server available at `heffa-thinlinc.nsc.liu.se` . In this way, you can get a graphical environment on the cluster and given that you work directly on the cluster there is no need to use ssh or scp (unless you want to copy files to your local machine). Please remember to log out when done working on the labs so that the server does not keep open sessions.

[4] http://spark.apache.org/docs/1.6.0/programming-guide.html
[5] https://www.cendio.com/thinlinc/what-is-thinlinc

ThinLinc is available on machines in the lab rooms. If you want to work on another machine, download the client from: https://www.cendio.com/thinlinc/download.

It is always a good practice to verify that one has kerberos tickets before starting to work with Hadoop, and if not, obtain them. You list kerberos tickets by running `klist` in the terminal, and get new ones with `kinit`. An example of a ticket is given below:

```
Default principal: zladr41@HEFFA.NSC.LIU.SE

Valid starting       Expires              Service principal
11/21/2016 13:56:46  11/22/2016 13:56:46  krbtgt/HEFFA.NSC.LIU.SE@HEFFA.NSC.LIU.SE
        renew until 11/28/2016 13:56:46
```

You should get the ticket automatically at login, but, if one uses ssh public key login, one may not get it. Check that you have acquired kerberos tickets **every time before starting your work** with the Hadoop server.

You can use Geany as text editor for writing your python scripts.

Running your scripts

To submit the jobs to the cluster using pyspark use:

```
spark-submit --deploy-mode cluster --master yarn --num-executors 9
--driver-memory 2g --executor-memory 2g --executor-cores 4 job.py
```

where `job.py` is your python script in your current folder. In this command, we use Yarn for resource management and use the cluster deploy mode. We have 9 worker nodes with 4 cores each with allocated 2GB of memory each.

To make the calling of your python scripts easier, you can download a bash script which includes all the settings (https://www.ida.liu.se/~732A54/lab/scripts/runYarn.sh). In this case, to run your `job.py` you will need to run:

```
./runYarn.sh job.py
```

You can change the settings by editing the `runYarn.sh` file. You might need to add the execute permissions to the script before you run it. To do this run:

```
chmod u+x runYarn.sh
```

During the execution of the job Spark starts SparkUI which is a web user interface for monitoring the job execution (more information available at: http://spark.apache.org/docs/latest/monitoring.html). However, the monitoring will only be available during the execution. In order to be able to access the logs after the execution you will need to set the spark.eventLog.enabled flag when running your job:

```
spark-submit --conf spark.eventLog.enabled=true --deploy-mode cluster
--master yarn --num-executors 9 --driver-memory 2g --executor-memory
2g --executor-cores 4 job.py
```

The script which includes the configuration for running the history server is provided here https://www.ida.liu.se/~732A54/lab/scripts/runYarn-withHistory.sh. To run your jobs use:

```
./runYarn-withHistory.sh job.py
```

To access the logs visit http://heffa-head.local:18088 with a web browser (only if using the ThinLinc approach). Similar as with runYarn.sh you might need to add the execute permissions.

Scheduling

Given the number of course participants and limited resources it may happen that you experience delays in executing your programs using Yarn. More specifically, you will notice that in some cases your application will be in the ACCEPTED state for few minutes until it reaches the RUNNING state. The reason for this is that there are already running tasks on the cluster which were submitted before. To check the up-to-date information about running/scheduled tasks visit:
http://heffa-head.local:8088/cluster

The exercises should not require a lot of time to run, and long running times might imply that there is something wrong with your code. So if you experience long run-times and you do not see other more running jobs please terminate your application (Ctrl-C) to save the resources.

SparkContext

When working with pyspark you will first need to acquire a SparkContext. SparkContext is the entry point to all functionality in Spark. Do this by including the following:

```
from pyspark import SparkContext
sc = SparkContext()
```

`SparkContext` accepts a number of parameters, such as the application name, number of executors, etc. For more information, check the documentation. When working with Spark SQL (for BDA2), in addition to SparkContext you will also need to acquire the SQLContext by:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

Where `sc` is your `SparkContext`.

In some exercises you will be required to copy files from/to hdfs. In these cases you will need to make use of hdfs commands. Check available commands by running `hdfs dfs` in the terminal. Some useful commands:

`hdfs dfs -copyFromLocal file.txt data/` - copies local file file.txt to folder data on hdfs

`hdfs dfs -mkdir data` - make a folder called data

`hdfs dfs -rm file.txt` - remove the file file.txt

`hdfs dfs -ls` - check the content of the folder

`hdfs dfs -rm -r folder` - remove the folder and its content

`hdfs dfs -copyToLocal results/ .` - copy the results/ folder to the current folder

When referencing files on hdfs (e.g. with `sc.textFile(path)`) you will need to provide the full path on hdfs. For example, if you created a file `file.txt` under folder `data` in your home directory on hdfs, the full path will be:

```
/user/{username}/data/file.txt
```

where `{username}` is your username.

Reports

For each lab hand in a lab-report that includes the name and LiU-id for each group-member. For each exercise provide your program, results from the program execution (a snippet of the results is enough if the results contain many rows) and written answers to questions in exercises. In cases where a plot of your results is asked, you can include the figure directly in the report. You can use a tool of your preference to produce the plots (e.g. R, Excel, matplotlib in Python, etc.). Comment each step in your program to provide a clear picture of your reasoning when solving the problem.

BDA1 - Spark - Exercises

In this set of exercises you will work exclusively with Spark. This means that in your programs, you only need to create the `SparkContext`.
In a number of exercises you will be asked to calculated temperature averages (daily and monthly). These are not always computed according to the standard definition of 'average'. In this domain the daily average temperature is calculated by averaging the daily measured maximum and the daily measured minimum temperatures. The monthly average is calculated by averaging the daily maximums and minimums for that month. For example, to get the monthly average for October, take maximums and minimums for each day, sum them up and divide by 62 (which is the same as taking the daily averages, summing them up and divide by the number of days).[6]

Assignments

1) What are the lowest and highest temperatures measured each year for the period 1950-2014. Provide the lists sorted in the descending order with respect to the maximum temperature. In this exercise you will use the *temperature-readings.csv* file.
    a) Extend the program to include the station number (**not the station name**) where the maximum/minimum temperature was measured.
    b) (not for the SparkSQL lab) Write the non-parallelized program in Python to find the maximum temperatures for each year without using Spark. In this case you will run the program using:
    `python script.py`
    This program will read the local file (not from HDFS) so you will need to copy the `temperatures-big.csv` to the local drive.
    How does the runtime compare to the Spark version? Use logging (add the --conf spark.eventLog.enabled=true flag) to check the execution of the Spark program. Repeat the exercise, this time using *temperatures-big.csv* file available on hdfs. Explain the differences and try to reason why such runtimes were observed.

2) Count the number of readings for each month in the period of 1950-2014 which are higher than 10 degrees. Repeat the exercise, this time taking only distinct readings from each station. That is, if a station reported a reading above 10 degrees in some month, then it appears only once in the count for that month.
    In this exercise you will use the *temperature-readings.csv* file.
    The output should contain the following information:
    Year, month, count

[6] Note: In many countries in the world, the averages are calculated as discussed. However, in Sweden, daily and monthly averages are calculated using Ekholm-Modéns formula which in addition to minimum and maximum daily temperature also takes into account readings at specific timepoints, the month as well as the longitude of the station. For more information check (in Swedish):
http://www.smhi.se/kunskapsbanken/meteorologi/hur-beraknas-medeltemperatur-1.3923

3) Find the average monthly temperature for each available station in Sweden. Your result should include average temperature for each station for each month in the period of 1960-2014. Bear in mind that not every station has the readings for each month in this timeframe.
   In this exercise you will use the *temperature-readings.csv* file.
   The output should contain the following information:
   Year, month, station number, average monthly temperature

4) Provide a list of stations with their associated maximum measured temperatures and maximum measured daily precipitation. Show only those stations where the maximum temperature is between 25 and 30 degrees and maximum daily precipitation is between 100 mm and 200 mm.
   In this exercise you will use the *temperature-readings.csv* and *precipitation-readings.csv* file.
   The output should contain the following information:
   Station number, maximum measured temperature, maximum daily precipitation

5) Calculate the average monthly precipitation for the Östergotland region (list of stations is provided in the separate file) for the period 1993-2016. In order to do this, you will first need to calculate the total monthly precipitation for each station before calculating the monthly average (by averaging over stations).
   In this exercise you will use the *precipitation-readings.csv* and *stations-Ostergotland.csv* files.
   HINT (not for the SparkSQL lab): Avoid using joins here! stations-Ostergotland.csv is small and if distributed will cause a number of unnecessary shuffles when joined with precipitation RDD. If you distribute *precipitation-readings.csv* then either repartition your stations RDD to 1 partition or make use of the collect to acquire a python list and broadcast function to broadcast the list to all nodes.
   The output should contain the following information:
   Year, month, average monthly precipitation

6) Compare the average monthly temperature (find the difference) in the period 1950-2014 for all stations in Östergotland with long-term monthly averages in the period of 1950-1980. Make a plot of your results.
   HINT: The first step is to find the monthly averages for each station. Then, you can average over all stations to acquire the average temperature for a specific year and month. This RDD/Data Frame can be used to compute the long-term average by averaging over all the years in the interval.
   The output should contain the following information:
   Year, month, difference

BDA2 - Spark SQL - Exercises

Assignments

Redo the above exercises using Spark SQL whenever possible. The initial processing of csv files (such as splitting on ;) can be done using Spark's map.

There are two ways to write queries in SparkSQL - using built-in API functions or running SQL-like queries. You can choose which method to use, however **to pass this lab you need to use the other way for at least one of the exercises**.

For each exercise include the following data in the report and sort it as shown:

1.
year, station with the max, maxValue ORDER BY maxValue DESC
year, station with the min, minValue ORDER BY minValue DESC

2.
year, month, value ORDER BY value DESC
year, month, value ORDER BY value DESC

3.
year, month, station, avgMonthlyTemperature ORDER BY avgMonthlyTemperature DESC

4.
station, maxTemp, maxDailyPrecipitation ORDER BY station DESC

5.
year, month, avgMonthlyPrecipitation ORDER BY year DESC, month DESC

6.
Year, month, difference ORDER BY year DESC, month DESC

BDA3 - Machine Learning with Spark - Exercises

Implement in Spark[7] (PySpark) a kernel method to predict the hourly temperatures for a date and place in Sweden. To do so, you should use the files *temperature-readings.csv* and *stations.csv*. Specifically, the forecast should consist of the predicted temperatures from 4 am (04:00) to 12 am (00:00) in an interval of 2 hours for a date and place in Sweden. Use a kernel that is the sum of three Gaussian kernels:

- The first to account for the distance from a station to the point of interest.
- The second to account for the distance between the day a temperature measurement was made and the day of interest.
- The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Choose an appropriate smoothing coefficient or width for each of the three kernels above. You do not need to use cross-validation.

Questions

1) Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.
2) Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.

Help

- Note that the file *temperature-readings.csv* may contain temperature measurements that are posterior to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot be used to compute the forecast.
- Cache the data you will reuse by using `rdd.cache()`. Check the course slides.
- Avoid joining two RDDs. Instead, broadcast the smallest, if small enough. Check the course slides.
- My program takes 5-6 minutes (wallclock) on the whole *temperature-readings.csv*. However, you may want to use a sample when implementing and testing different settings. Then, do `rdd.sample(False, 0.1)` to obtain a sample without replacement of size 10 %.
- Feel free to use the template below to solve the assignment.

[7] Do not use SparkSQL

Template

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext

sc = SparkContext(appName="lab_kernel")

def haversine(lon1, lat1, lon2, lat2):
      """
      Calculate the great circle distance between two points
      on the earth (specified in decimal degrees)
      """
      # convert decimal degrees to radians
      lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
      # haversine formula
      dlon = lon2 - lon1
      dlat = lat2 - lat1
      a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
      c = 2 * asin(sqrt(a))
      km = 6367 * c
      return km

h_distance = # Up to you
h_date = # Up to you
h_time = # Up to you
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-07-04" # Up to you

stations = sc.textFile("data/stations.csv")
temps = sc.textFile("data/temps.csv")

# Your code here
```