Assignment: Data Analysis with Spark

TDT4305 Big Data, Spring 2016

The practical assignment consists two tasks that we describe in this document. It is focused on learning how to perform data analysis for large datasets. You will work with the Apache Spark framework[1] and you can choose freely which of the Spark-compatible languages you prefer: Python, Java or Scala. You can work in groups of two people.

1 Exploratory Analysis of Foursquare Dataset

Foursquare[2] is a location-based social network whose main objective is recommendation of places of interest (e.g., restaurants, airports, historic sights) and sharing of user's current and past locations with 'friends'. When user marks his/her actual position in the Foursquare application, his/her geographical coordinates are recorded together with information about the place of interest (POI) he or she is currently at. This activity is called "check-in" to POI.

In this task you'll work with the Foursquare dataset that contains check-ins of many users to many POIs and your goal is to analyze the dataset using the Spark framework. Try to understand differences and best practices in usage of various Spark functions.

1.1 Data

dataset_TIST2015.tsv

Download dataset: `http://tiny.cc/bigdata-foursquare` (354 MB)

We provide you with a dataset of Foursquare check-ins (1.8GB). It comes in *Tab Separated Values* format (TSV), with one check-in record per line and contains the following columns:

[1] `http://spark.apache.org/`
[2] `http://www.foursquare.com`

1. `checkin_id` - unique identifier for each check-in.
2. `user_id` - unique identifier for each user; no other user info is available in this dataset.
3. `session_id` - series of check-ins that belong to one user and where no pair of consecutive check-ins is more than 6 hours apart is called a session of check-ins. Each session is given a unique id.
4. `utc_time` - check-in time on the server represented by a UNIX timestamp.
5. `timezone_offset` - timezone offset of the check-in in minutes with respect to UTC time.
6. `lat` - geographical latitude.
7. `lon` - geographical longitude.
8. `category` - category of the check-in (e.g., Food).
9. `subcategory` - subcategory of the check-in (e.g., Sushi Restaurant).

dataset_TIST2015_Cities.txt

Each check-in contains exact geographical coordinates, however, to find out in which city it was performed, you need to have coordinates of the cities. This dataset contains the following fields:

1. `City name`
2. `Latitude (of City center)`
3. `Longitude (of City center)`
4. `Country code (ISO 3166-1 alpha-2 two-letter country codes)`
5. `Country name`
6. `City type (e.g., national capital, provincial capital)`

1.2 Task

On your local machine install Spark, download above mentioned datasets and using suitable Spark functions (where appropriate) perform these tasks:

1. Load the Foursquare dataset.
2. Calculate local time for each check-in (UTC time + timezone offset).
3. Assign a city and country to each check-in (use Haversine formula described below).
4. Answer the following questions:
   (a) How many unique users are represented in the dataset?
   (b) How many times did they check-in in total?
   (c) How many check-in sessions are there in the dataset?
   (d) How many countries are represented in the dataset?
   (e) How many cities are represented in the dataset?
5. Calculate lengths of sessions as number of check-ins and provide a histogram of these lengths.
6. For sessions with 4 and more check-ins, calculate their distance in kilometers (use Haversine formula to compute distance between two pairs of geo. coordinates).
7. Find 100 longest sessions (in terms of check-in counts) that cover distance of at least 50Km.
   (a) For these 100 sessions, output data about their check-ins into a CSV or TSV file. Use all available data fields such as checkin_id, session_id, etc. and also add check-in date in `` `YYYY-MM-DD HH:MM:SS' `` format.
   (b) Visualize these sessions in CartoDB (more info below).
8. Feel free to explore the data also in other ways. Extensions to the analysis and further insights are welcome!

1.2.1 Haversine formula

The Haversine formula[3] is used to calculate the distance between two points on a sphere (e.g., the Earth), where the two points are represented by their longitudes a latitudes (in radians).

$$d = 2r \arcsin(\sqrt{\sin^2(\frac{\varphi_2 - \varphi_1}{2}) + \cos(\varphi_1)\cos(\varphi_2)\sin^2(\frac{\lambda_2 - \lambda_1}{2})}) \quad (1)$$

, where d is the calculated distance, r is the radius of sphere (6378Km for the Earth), φ_1, φ_2 are the latitudes of point 1 and 2 and λ_1, λ_2 are the longitudes of point 1 and 2.

Implementation in Python:

```
def haversine(lat1, lon1, lat2, lon2):

    # 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))
   r = 6371 # Radius of earth in kilometers. Use 3956 for miles
   return c * r
```

1.2.2 Session visualization in CartoDB

CartoDB is an in-browser mapping service, which you can use to visualize check-ins or sessions. How to visualize the check-ins:

1. Create a free account at `https://cartodb.com/signup`.
2. Upload your results in CSV/TSV format with 1 check-in per line.
3. 'DATA VIEW' tab should open once you choose to connect the dataset.
4. Select which of your columns refer to coordinates (lat, lon). If you name your columns 'lat'/'lon', CartoDB will recognize them automatically.

[3] `https://en.wikipedia.org/wiki/Haversine_formula`

5. Check if column with check-in date was recognized as a date data type, otherwise change the data type.

6. Switch to 'MAP VIEW' tab.

7. In the right panel, choose 'wizard' and change map type to 'CATEGORY'; change 'Column' drop-down menu to the column in your data that represents category of check-ins.

1.3 Delivery

You should deliver a written report (PDF), short oral presentation and source code of your program. In your report, provide answers to the (sub)tasks and describe which Spark functions you used and why. In the presentation, you should quickly guide us through your code, explain used methods and demonstrate results. It shouldn't take more than 10 minutes.

2 Sentiment Analysis on Twitter

Sentiment analysis is a very common technique for extraction of general opinion on some topic. It is used in many real-world scenarios such as analyzing reviews on a product, estimating election preferences, etc. In this task, you will use this technique in its simple form to describe sentiment of geo-located tweets. The main emphasis of this second part of the assignment is put on efficiency, we'd like you to come up with fastest possible solution using Apache Spark and your selected language. Your code will be run on much larger dataset and execution time will be measured.

2.1 Data

geotweets.tsv

Download dataset: `http://tiny.cc/bigdata-twitter` (222 MB)

We pre-processed a dataset obtained from publicly available Twitter API; it contains the following columns and is represented in *TSV* format.

1. `utc_time` - check-in time on the server represented by a UNIX timestamp.
2. `country name`
3. `country code`
4. `place type`
5. `city name`
6. `language`
7. `username`
8. `user screen name` (displayed on Twitter account)
9. `user timezone offset` (minutes)
10. `number of friends`
11. `text of tweet`
12. `latitude`
13. `longitude`

positive-words.txt, negative-words.txt

These two files are part of the same archive as `geotweets.tsv` and contain lists of positive and negative words - one per line.

2.2 Task

On your local machine install Spark, download above mentioned datasets and using suitable Spark functions (where appropriate) perform this task in the most efficient way:

1. Load the Twitter dataset.
2. **Find aggregated polarity (sentiment) of all English tweets (`lang = 'en'`) for each city in the United States (`place type = 'city'`, `country = 'US'`) for each day of week.**
3. Output your result into a file in predefined format.

Important notes:

- Work with tweet's local time (you need to calculate the time from UTC time and timezone offset).
- Keep in mind that words in the lexicons are lowercased (you need to lowercase the tweet texts or use case-insensitive comparison).
- You can ignore non-ascii characters in the tweets if it causes troubles in your program.

2.2.1 Finding polarity of a tweet

Given the lexicons of positive and negative words (i.e., `positive-words.txt`, `negative-words.txt`), look for words from these lists in each tweet. Count '+1' for each positive and '-1' for each negative word within a tweet and calculate the aggregated polarity $Tweet_{ap}$ of a tweet by summing up these values. Assign the final polarity $Tweet_{fp}$ to each tweet according to its aggregated polarity as positive (+1), negative (-1) or neutral (0):

$$Tweet_{fp} = \begin{cases} 1, & \text{if } Tweet_{ap} > 0 \\ 0, & \text{if } Tweet_{ap} = 0 \\ -1, & \text{if } Tweet_{ap} < 0 \end{cases} \quad (2)$$

To find the overall polarity for a set of tweets you only need to sum up their final polarities. For instance, if you have 7 tweets with the following polarities: $\{-1, 1, 1, 1, 0, -1, 1\}$, the overall polarity of the set is 2.

2.2.2 Input and output format

Arguments
Your program should accept two arguments from the command line: 1) path to the input dataset, 2) path to the output file.

Output format
Your program should output the results in *TSV* (tab-separated values) format with the following structure:

```
city<tab>day_of_week<tab>overall_polarity
```

Example:

```
New York    Monday     -3440
New York    Tuesday    14777
New York    Wednesday  1777
...
Chicago     Monday     4409
...
```

2.3 Delivery

You should deliver a source code of your program. If you deliver reasonably ahead of deadline (only 1 program will run on the server at a time), we'll test your code and give you feedback.