UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Big Data Management (BDMA & MIRI Masters)
Session: Spark Streaming

Sergi Nadal

Instructions

In this session, as Spark jobs will run locally, you do not need to start the Spark service and everything can be run from eclipse. Thus, you can dismiss the virtual machine and directly use the eclipse version provided in the lab session computers. Notice that as lab sessions are performed in couples, only one computer will be used during the session.

Once you have fetched the code, it can be imported as a *Maven Project* into intelliJ or eclipse. All executions can be run locally, and thus you do not need to be concerned with the host connection. In the *Main* class you can see that different arguments are expected, and will drive the behaviour of the program. The simplest way to define such arguments is by creating a *Run Configuration* of type *Java Application* from the *Run* menu. In the tab menu called *Arguments* you will be able to edit such arguments, in each exercise you will be guided with the required arguments.

From now on, we assume that Twitter's API credentials have been properly created as described in the training. If you do not have such credentials, please refer to the training to see how to create them.

Delivery

By the end of the session hand in your answers. Coding exercises must be delivered via LearnSQL. Prepare a ZIP file and upload it to LearnSQL, with the following content: (1) `Exercise_1.java`, (2) `Exercise_2.java`. **The system will close the submission five minutes before the deadline, make sure to submit your file before!**

Exercise 1: Sentiment analysis of Twitter for COVID-19 (50% weight)

Sentiment analysis (SA), sometimes referred as opinion mining, describes the process of using NLP, statistics, or machine learning methods to extract, identify, or otherwise characterize the sentiment content of a text unit. In this exercise we will perform real-time SA to the stream of tweets related to COVID-19, in order to distinguish if the tweets are expressing negative feelings/reactions or positive stories related to the pandemics. The implementation of the exercise can be found in class `Exercise_1.java`, where some helper functions are provided. In the next steps, you will be guided on how to complete each subtask.

Preprocessing

A sentiment analysis process is dependant of the language, therefore first we need to filter the stream in order to retrieve only tweets written in English. In order to help you, we provide the method `LanguageDetector.isEnglish(String)` which will return `true` or `false` whether the provided text is written in English or not. For further reference on how this is done, you can check the library `Apache Tika`.

Next, given that we consider tweets talking about COVID-19, you should further filter those containing specific keywords: "COVID" or "coronavirus".

Once your stream has been filtered, it is recommended to do further preprocessing by mapping only the pairs (id, text) and making sure you are not working with any `null` text.

Applying text functions

It is necessary to put all text to analyze to a common standard form, this can be done by converting the tweet using the following code.

```
1 text.replaceAll("[^a-zA-Z\\s]", "").trim().toLowerCase();
```

After, it is also necessary to perform *stemming*. Stemming techniques can be very complex, in here we will just get rid of stop words (those words that do not provide any value for our purpose). The method `StopWords.getWords()` is provided, which returns a list of stop words.

Scoring tweets

Now it's time to get the tweets and decide whether they express positive opinion or not. For both positive and negative, the methods `PositiveWords.getWords()` and `NegativeWords.getWords()` are provided, which return a list with the corresponding words. For the case of positive words, in order to score each word, you should check how many words in the tweet are positive (by checking containment in the positive words list). Finally, the tweet's positive score is calculated by means of $\frac{p}{n}$, where p is the number of positive words and n is the total number of words in the tweet. For negative words, the process is likewise but using the proper list of negative words.

Classifying tweets

Once tweets have been scored and two sets (positive and negative) of triples (id, text, score) have been obtained, they should be merged. This can be easily achieved using the method `join`, as the following code shows:

```
1 JavaPairDStream<Tuple2<Long, String>, Tuple2<Float, Float>> joined = positiveTweets.join(
    negativeTweets);
```

The previous code joins the sets `positiveTweets` and `negativeTweets`, using the keys in the first parameter `Tuple2<Long, String>` (the id and text) and adding the two scores in `Tuple2<Float, Float>`. Afterwards, the two sets of `Tuple2` should be mapped to a single `Tuple4`, representing the structure (id, text, positive score, negative score). Finally, in order to classify the tweet we just need to map a new attribute describing the sentiment of the tweet:

- **Positive** when $pos\_score > neg\_score$.
- **Neutral** when $pos\_score = neg\_score$.
- **Negative** when $pos\_score < neg\_score$.

This will construct a final structure `Tuple5<Long, String, Float, Float, String>`. If you print the contents of this final variable, you should see something like this:

```
-------------------------------------------
Time: 1428243800000 ms
-------------------------------------------
(584722988152401921,rt larry baylor called protesters today faith miracle temple
    httptcopqqcrdw,0.15,0.05,positive)
(584723441149935616,morrbecks yeah i bothered good  a waste space,0.0625,0.125,negative)
(584723520833261569, life a book chapters sad happy exciting turn page
    ,0.08695652,0.04347826,positive)
```

Exercise 2: Optimizing the historical analysis of hashtags (50% weight)

Acknowledgement. The content of this exercise is based on Section 4.7.2 *Definition of the Decaying Window* from the following book: *Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). Mining of massive datasets. Cambridge university press.* A recommended reading, available online at `http://infolab.stanford.edu/~ullman/mmds/book.pdf`.

After implementing the third exercise in the training, you should have realized that this implementation is not scalable for massive data streams and/or long time computations. Indeed, we are storing the full set of hashtags independently of their time of occurrence. A more scalable alternative is to redefine the question so that we compute a smooth aggregation of the hashtags seen in the stream, with decaying weights, so the further back in the stream, the less weight is given. This approach is known as *exponentially decaying window*. Formally, let a stream currently consist of the elements $a_1, a_2, \ldots, a_t$, where a_1 is the first element to arrive and a_t is the current element. Let c be a small constant, such as 10^{-6} or 10^{-9}. Define the exponentially decaying window for this stream to be the sum

$$\sum_{i=0}^{t} a_{t-i}(1-c)^i \tag{1}$$

The effect of this definition is to spread out the weights of the stream elements as far back in time as the stream goes. Going back to the problem at hand, we establish a threshold, say 1/4, so that if the popularity score for a hashtag goes below this number, its score is dropped from the counting. At each microbatch, do the following:

1. For each hashtag whose score we are currently maintaining, multiply its score by $(1 - c)$.
2. For each hashtag H in the microbatch
   - If there is currently a score for H, add 1 to that score. If there is no score for H, create one and initialize it to 1.
3. If any maintained score is below the threshold 1/4, drop that element.

In class `Exercise_2.java` implement the processing of the incoming tweets and print the *Top 20* hashtags and the *Median* hashtag using the exponentially decaying window model. However, if you feel that any of them (or both) does not make sense to be implemented in such model, instead (directly in the code as a comment) explain why.