Big Data Analytics
Prof. Dr. Martin Theobald, Alessandro Temperoni & Jingjing Xu
Summer Semester 2022 – University of Luxembourg
Exercise Sheet #3 – Due on May 19, 2022 (before the lecture starts)

Please upload the source code of your solutions on or before the due date to the provided Moodle assignment as a single zip file using your group id as the file name. Provide some brief instructions on how to run your solution to each problem in a file called `Problem_X.txt`*, and report also the most important results (such as number of results, runtimes, etc.) of your solutions to that problem within this file. Remember that all solutions may be submitted in groups of up to 3 students.*

***Note:* Since this exercise sheet imposes intensive computational workloads, it is highly recommended to run the following experiments on the IRIS HPC cluster by using either the interactive or batching modes of the Spark launcher scripts (see once more the "Getting Started" guide provided on Moodle for detailed instructions).**

Latent Semantic Analysis of Wikipedia Movie Plots — 12 Points

Problem 1. Consider the "*Wikipedia Movie Plots*" dataset as it is available from Kaggle:

`https://www.kaggle.com/datasets/jrobischon/wikipedia-movie-plots`

You may download the dataset (in CSV format) either from the above URL or via Moodle. This openly available dataset captures 8 fields (*release year*, *title*, *origin/ethnicity*, *director*, *cast*, *genre*, *wiki page*, and *plot*) scraped from 134,164 Wikpedia movie articles. For this problem, we will focus on the usage of LSA to analyze the *plot* fields of these articles. Follow the general methodology provided in `RunLSA-shell.scala` to solve the following tasks.

(a) Adapt the provided parsing functions of the shell script to read the *title*, *genre* and *plot* fields of the articles into an initial DataFrame in Spark (see Exercise Sheet #2 and/or the lecture slides for details). Also make sure to apply a proper schema to your DataFrame object. **2 Points**

(b) Next, add an additional column called `features` to your DataFrame, which contains a list of lemmatized text tokens extracted from each of the *plot* fields using the NLP-based `plainTextToLemmas` function of the given shell script. **2 Points**

(c) Translate the `RunLSA-shell.scala` shell script (including the new steps from (a)–(b)) into an actual Scala object called `RunLSA.scala`, and compile this object into a jar file called `RunLSA.jar` which can then also be executed via `spark-submit` on the HPC cluster. **2 Points**

(d) Compute an SVD decomposition of the 134,164 movie plots contained in your DataFrame by using the following two basic parameters:

- $numFreq = 5000$ for the number of frequent terms extracted from all Wikipedia articles, and
- $k = 25$ for the number of latent dimensions used for the SVD.

(e) Based on the two `topTermsInTopConcepts` and `topDocsInTopConcepts` functions provided in the `RunLSA-shell.scala` shell script, compute the *top-25 terms* and the *top-25 documents*, each under the $k = 25$ *latent concepts*, for the above SVD.

Additionally modify the `topDocsInTopConcepts` function, such that it also prints the top-5 most frequent *genre* labels of the Wikipedia articles returned by this function under each of the latent concepts.

Manually inspect the results to determine if the SVD improves the representation of the documents. **2 Points**

(f) Modify the provided `topDocsForTermQuery` function such that it computes the *Cosine measure* (instead of the inner product) between a translated document vector $\boldsymbol{d}'$ and a similarly translated query vector $\boldsymbol{q}'$ over the latent semantic space:

$$Cosine(\boldsymbol{d}', \boldsymbol{q}') = \frac{\boldsymbol{d}' \cdot \boldsymbol{q}'}{\|\boldsymbol{d}'\|_2 \cdot \|\boldsymbol{q}'\|_2}$$

Sort the matching documents in descending order of Cosine similarities, and finally return the *top-25 document vectors* (and corresponding *title* entries) with the highest similarities to each such keyword query.

Finally, think of 5–10 interesting keyword queries for movies and report their results. **2 Points**

Summarize the results (incl. the Spark runtimes) of this experiment in your `Problem_1.txt` file.

Outlier Detection for Gearbox Readings **12 Points**

Problem 2. Consider the "*Gearbox Fault Detection*" dataset released by the NASA in 2009 and available from Kaggle:

`https://www.kaggle.com/datasets/hetarthchopra/gearbox-fault-detection-dataset-phm-2009-nasa`

You may download the dataset (in CSV format) either from the above URL or via Moodle. Apply and (if necessary) adapt the `RunKMeans-shell.scala` script to detect possible outliers among the 14M gearbox readings provided in this dataset. Start with a reasonably small amount of clusters (e.g. using $k = 1..10$), and also aim to provide a plot of a (reasonably small) sample of the readings by using a visualization technique of your choice (submitting the source code to generate the visualization from your data is sufficient; images are optional).

Summarize the results (incl. the Spark runtimes) of this experiment in your `Problem_2.txt` file. Your solution will be graded based on the appropriateness and creativity of your applied methodology.

Social Network Analysis in GraphX **12 Points**

Problem 3. Consider the "*Social networks: Twitter*" dataset as it is available from the below URL:

`http://snap.stanford.edu/data/ego-Twitter.html`

Consider `RunGraphX-shell.scala` as a basis to solve this exercise. Focus on the `twitter_combined.txt` file as the combined set of edges (constructed from 973 individual networks) in this social-network graph to solve the following parts of this exercise. In addition to this file, the dataset also provides the following kinds of features for various nodes:

- `nodeId.feat` – binary indicators for which features are associated with a given node id
- `nodeId.featnames` – actual node features in the form of hashtags and user ids from Twitter

(a) Load and parse the `twitter_combined.txt` file into an initial RDD into your Spark environment which will be used as edge RDD for GraphX. Extract all distinct node ids from this file to create an additional vertex RDD for GraphX. Use both the vertex and edge RDDs to initialize your GraphX graph; parse the features from the additional files and assign them as the vertex labels. **3 Points**

(b) Perform a *connected-components* analysis of the combined set of edges and check if the number of your connected components indeed matches 973. **2 Points**

(c) Compute the following three graph-analysis techniques (as they are already implemented by the provided shell script): *degree-distribution*, *average clustering-coefficient*, and the *average path length* among pairs of nodes to your Twitter graph. How would you thus describe the graph? **2 Points**

(d) Use the built-in PageRank API of GraphX to find the 250 nodes with the highest PageRank value. Compare this ranking of nodes with a ranking you obtain by simply selecting the 250 nodes with the highest in- and out-degree of incoming and outgoing edges. **2 Points**

(e) Perform the methodology introduced in the lecture for the χ^2 *test of independence*[1] in order to verify the following hypothesis:

"The degree of a node is independent of the number of features associated with that node."

To do so, aggregate and discretize both the degree (of incoming and outgoing edges) and the number of features (hashtags and user ids) associated with each node into 10x10 buckets of the following form:

```
0:  [0-10) features,  0:  [0-10) hashtags:  #nodes
1:  [10-20) features, 0:  [0-10) hashtags:  #nodes
   ...
9:  [90-∞) features, 9:  [90-∞) hashtags:  #nodes
```

Next, determine the *degrees of freedom* of the corresponding χ^2 distribution and compute the p-value for the above data distribution. Would you thus accept or reject the above hypothesis? **3 Points**

Summarize the results (incl. the Spark runtimes) of this experiment in your `Problem_3.txt` file.

[1] See: `https://spark.apache.org/docs/latest/ml-statistics.html`