THE UNIVERSITY OF SYDNEY

School of Information Technologies
A/Prof Uwe Roehm

**COMP5349: Cloud Computing** **1.Sem./2017**

# Assignment 3: Apache Spark/Flink Programming

**Group Work: 15%** **30.05.2017**

## Introduction

In this assignment, you shall write a series of Apache Spark or Apache Flink programs to analyze a *gene expression* data set that was collected to study the activity (the *expression*) of individual genes of patients who are treated for various diseases. Note that this is a different data set than we have used in assignments 1 or 2. The analysis is structured into three separate tasks. You will practice dataflow programming techniques and also observe the ease of use of the Spark and Flink systems.

## Input Data Set Description

The input data can be found in the HDFS directory `/share/genedata` in our cluster. There are several subdirectories with csv files named `GEO.txt` (gene expression data) and `PatientMetaData.txt`. All these files have a comma-delimited format.

The main data set is from DNA microarrays which measure the relative activity of certain genes ('gene expression') using cell samples from patients. The corresponding files are called `GEO.txt` and each line representing some gene expression value for some patient:

```
patientid, geneid, expression_value
```

Note that there is a header row that just gives the meaning of each column:

- The `patientid` column specifies the ID of the patient whose cells were analysed.
- `geneid` is an ID of a specific gene whose activity (*expression*) level was measured.
- `expression_value` represents a numerical value on how much a given gene is active in the patient's cells. For the purpose of this assignment, we simply consider a gene to be expressed (or active) if its level is above 1,250,000.

The meta-data for the patients are stored in the csv file `PatientMetaData.txt` Each line of this file contains a record with the following `comma-delimited` information:

```
id, age, gender, postcode, diseases, drug_response
```

The `id` column specifies a patient's ID, while the `age` and the `postcode` columns give the age of a patient and in which postcode they live. The `gender` column specifies a patient's gender ('f' for female patients and 'm' for male patients). In the `diseases` column, we find one or more disease types for all diseases that a patient is treated for (a patient can have more than one disease). The individual disease types are separated by space. The last column is the `drug-response` which specifies how well a patient is responding to his current drug treatment (higher is better).

## Analysis Task Descriptions

You shall implement three different analysis tasks of the data set in either Apache Flink or Spark (as assigned by your tutor):

1. **Task 1: Number of cancer patients with certain active genes *per cancer type***
   The first task is an explorative analysis of the gene expression data similar to the previous assignments. You shall write a Flink or Spark program that determines how many cancer patients have a high expression level of gene 42. We consider a gene being (strongly) expressed if its expression value is larger than 1,250,000. A cancer patient is any patient who is treated for either breast-cancer, prostate-cancer, pancreatic-cancer, leukemia, or lymphoma.

   The output file should have the following format, ordered by number of expressed genes in descending order (cancer types with most gene-42-patients first); cancer types with the same number of occurrences of gene 42 should be listed alphabetically:

   `cancer_type \t number_of_patients_with_gene42_expressed`

2. **Task 2: Frequent Itemset Mining**
   In the second task, we are interested in common combinations of genes expressed with cancer patients. To do so, you shall implement the first part of the iterative Apriori algorithm for frequent itemset mining (for a detailed discussion of this algorithm see Page 4), to identify the frequent combinations (*frequent itemsets*) on our gene expression data.

   We are again only interested in gene expressions from cancer patients. A cancer patient is any patient who is treated for either breast-cancer, prostate-cancer, pancreatic-cancer, leukemia, or lymphoma. Only consider strongly expressed genes, ie. such genes which have an expression value above 1,250,000.

   Determine all frequent itemsets of expressed genes. A frequent itemset is any combination of expressed genes with a *support count* (number of occurrences) at least a minimum support count. This minimum support should be a program parameter. By default, check for itemsets with a support value of 30% of all patient samples. You can also define a maximum itemset size as a program parameter, which would allow you to limit the number of iterations.

   The output file should list the frequent itemsets in the following tab-delimited format, ordered by descending support, and with each itemset ordered by increasing gene-ID:

   `support \t` $\texttt{item}_1$ `\t` $\texttt{item}_2$ `\t ... \t` $\texttt{item}_k$

3. **Task 3: Association Rule Generation**
   In the third task, you shall generate *association rules* from the frequent itemsets that you found in previous Task 2: For every frequent itemset $S$ of size $2$ or larger, that was identified in Task 2, generate all non-empty subsets $R$. For every such subset $R$ output a rule of the form $R \Rightarrow (S - R)$ iff the *confidence* value for this rule is at least a *confidence threshold*. This confidence threshold should be a parameter of your program, with a default value of $60\%$ . The confidence of a rule is defined as: $confidence = \frac{support(S)}{support(R)}$

   The output should have the following format (ordered by descending confidence value):

   *`R \t (S-R) \t confidence_value`*

   where $R$ and $(S - R)$ are the items from the right-hand respectively left-hand side of an association rule, separated by space.

## Special Coding Requirements

1. You will solve this assignment with either Apache Flink or Apache Spark, depending on which framework you did not use in assignment 2. The tutors will assign each team to one of these two systems so that each team now programs in the second dataflow framework which it did not use in the previous assignment 2. This will ensure that by the end of the semester, each team will have worked with both systems.

2. The purpose of this assignment is to experience iterative programming in Spark/Flink. You are hence not allowed to use any pre-defined machine learning libraries for this assignment, but rather you should implement the Apriori algorithm yourself.

3. If you use any code fragments or code cliches from third-party sources, you must reference those properly. Include a statement on which parts of your submission are from yourself.

4. Always test your code using a small data set (eg. `/share/genedata/test/`) first.

5. Do not copy the entire input data to your HDFS home directory or Linux home directory

## Deliverable

There are three deliverables: **source code**, a brief **program design documentation** (up to 3 pages), and a **self-reflection survey**. All are due on Week 13 Friday 9$^{th}$ of June (late submission penalty is waived until Tuesday June 13). The marking rubric will be available on eLearning. Please submit the source code and a soft copy of design documentation as a zip or tar file in eLearning.

Your document should contain the following:

- **Job Design Documentation**
  In your document, describe the Flink/Spark jobs you use to implement Tasks 1 to 3. For each job, briefly describe the different transformation functions. If you use a user-defined functions, classes or operators, please describe those too.
- Include a table with the **execution times of your assignment tasks** (Task 1, Task 2 and Task 3) for small and large data sets.
- Include as appendix the HDFS location of your final output files from various executions.

**Self-Reflection Survey:** As part of the submission, you are also asked to fill in a self-reflection survey, one per each team member. This survey will ask you to assess your experience with your chosen implementation systems (Flink or Spark).

### Group Member Participation

If members of your group do not contribute sufficiently you should alert your tutor as soon as possible. The tutors have the discretion to scale the group's mark for each member as follows:

| Level of contribution | Proportion of final grade received |
|---|---|
| No participation. | 0% |
| Passive member, but full understanding of the submitted work. | 50% |
| Minor contributor to the group's submission. | 75% |
| Major contributor to the group's submission. | 100% |

## Background: Apriori Algorithm

The subject of this assignment is to implement the *Apriori algorithm* (Agrawal and Srikant, 1994) with Apache Spark or Flink. The Apriori algorithm is a data mining algorithm for identifying frequent itemsets and association rules in databases: Which set of items occur together in the data set with high probability? The algorithm was initially motivated by business scenarios where the goal is to identify popular product combinations. This is reflected by many textbook examples which explain the algorithm in the context of a market basket analysis for a supermarket or an online store, and where the algorithm is used to analyse a set of purchase transactions. A good example is: `http://nikhilvithlani.blogspot.com.au/2012/03/apriori-algorithm-for-data-mining-made.html`

In this assignment we will apply the Apriori algorithm to a scenario from Genome Biology: Here, the 'transactions' are cell samples from patients for which microarray gene-expression experiments were performed. A DNA microarray is a scientific method that allows to measure the activity of a large number of known genes in a cell sample simultaneously; the resulting 'expression values' in the microarray data reflect how much various genes are active the given cell sample of some patient. Depending on the functionality of the analysed cells and the health of a patient, genes can be under- or over-expressed in this expression data. The 'product items' from the business scenario are highly active ('over-expressed') genes whose expression level is over a certain threshold. Apart from these different contexts, the algorithm itself works exactly the same.

**Frequent Itemset Mining.** In its first phase, the Apriori algorithm finds *frequent itemsets*: Which set of items occur often together in the same transactions in the database? In our Genome Biology scenario, the question translates to identify combinations of genes which are often expressed together in the patients' cell samples. We refer to the number of cell samples, where this is the case, as the *support count* of an itemset. We do not expect these gene combinations to occur with every individual patient, but at least with a certain threshold of patients: A *frequent itemset* is any combination expressed genes with a support count at least a given *minimum support count*.

Apriori uses a 'bottom up' approach, where frequent itemsets are extended one item at a time. The algorithm starts by first finding individual frequent items (itemsets of size 1), which in our scenario are individual expressed genes in the cell samples of more than *minimum-support* patients.

It then iteratively generates candidate itemsets of length $k+1$ from frequent itemsets of length $k$. In each iteration, Apriori extends the already known frequent items sets of length $k$ with each of the individual frequent item as found in Step 1. It then prunes from this set of candidate itemsets of length $k+1$ all those whose support count is below the support threshold of cell samples, ie. it only keeps itemsets with a *support count* $\geq$ *minimum support count*. The algorithm terminates when no further frequent itemsets are found. The result of the first phase is a set of frequent $k$-itemsets of size $k = 1$, $k = 2$, $k = 3$, etc.

**Association Rule Generation.** In the second phase, one can now generate *association rules* from the identified frequent itemsets as follows: For every frequent itemset $S$, that was identified in Phase 1, generate all non-empty subsets $R$. For every such subset $R$ output a rule of the form $R \Rightarrow (S - R)$ iff the *confidence* value for this rule is above (or equal to) a confidence threshold. The confidence of a rule is defined as: $confidence = \frac{support(S)}{support(R)}$

**References:**

Rakesh Agrawal and Ramakrishnan Srikant. *Fast algorithms for mining association rules in large databases*. In: Proceedings of VLDB, pages 487-499, September 1994.