Massive Data Sets
Assignment No. 5

Note 1 **This assignment is to be done individually or in teams of 2**

Note 2 Working with other people is prohibited.

Note 3 Sharing source code of with other people outside your team is prohibited.

Note 4 The work you submit must be your own.

A note on Academic Integrity and Plagiarism

Please review the following documents:

- Standards for Professional Behaviour, Faculty of Engineering:
  `https://www.uvic.ca/engineering/assets/docs/professional-behaviour.pdf`
- Policies Academic Integrity, UVic:
  `https://www.uvic.ca/students/academics/academic-integrity/`
- Uvic's Calendar section on Plagirism:
  `https://www.uvic.ca/calendar/undergrad/index.php#/policy/Sk_0xsM_V`

Note specifically:

> Plagiarism
> Single or multiple instances of inadequate attribution of sources should result in a failing grade for the work. A largely or fully plagiarized piece of work should result in a grade of F for the course.

Submissions will be screened for plagiarism **at the end of the term**.
You are responsible for your own submission, but you could also be responsible if somebody plagiarizes your submission.

1 Objectives

After completing this assignment, you will have mining a massive dataset for knowledge.

- Data pre-preprocessing and preparation
- Using spark
- Clustering of categorical attributes

2 StackOverflow and programming languages

Does the programming language that one knows affects the languages that one learns? Probably. In that case, can we identify sets of languages that are commonly known by the same person?

StackOverflow provides a valuable dataset for this purpose. Many questions and answers in *StackOverflow* are tagged with a programming language. The fundamental assumption is that those who participate in asking or answering a question related to a given programming language are learning it. If a person asks a question regarding a programming language, then it is likely that person is learning or using that programming language. If a person answers a question, that person knows the language (at least to the point of feeling confident to know the answer). We will say that a person uses a language if the person has posted a question tagged with that language or has answered a question tagged with that language.

StackOverflow uses the term post to refer to both a question and an answer.

Thus, we can model *StackOverflow* as a graph: the vertices are the programming languages and the users; and the edges the posts that link both types of vertices.

2.1 Data Files

Download the *StackOverflow* data. You are provided three files: `posts.csv`, `users.csv`, `poststags.csv` and `languages.csv`. The first three files have a header. The latter does not. The headers are self-explanatory. We will only use:

- `userid`: the *StackOverflow* id of the user
- `postid`: the *StackOverflow* id of the post
- `parentid`: see below
- `tag`: the tags associated with a given question. Only tags present in the file `languages.csv` should be considered programming languages.

Note that posts are divided into *questions* (those with `posttypeid` equal to 1) and *answers* (those with `posttypeid` equal to 2). Only the questions have a tag. Every answer is linked to its question via a `parentid`. Therefore the tags of an answer are the tags of its question (its parent).

The file `languages.csv` is a list of tags that I have identified are programming languages. This list was built from the list of programming languages named used by Tiobe for its programming languages index[1]. The number is an index that will be useful during the clustering part of the assignment.

1. There are some tuples with an invalid userid. **Ignore any tuple where the userid, the postid, or the parentid is not an integer.**

3 Your task, should you choose to accept it

3.1 Cleaning up the data

The first task is preprocessing the data to create a csv data file that will be used as input.

[1]https://www.tiobe.com/tiobe-index/

1. To reduce “noise” we will ignore any post that has been tagged with two or more programming languages.
2. We ignore users who have only used one programming language
3. For any user, we will ignore languages that they have used only once. E.g. if a user has 3 C++ posts, 2 Java posts and 1 Scala post, then we will ignore the Scala post, and consider only the C++ and Java posts.

3.2 Part 1: Baskets of programming languages: baskets.csv

We need to map the data to baskets, and create a file, where each line is a basket.

1. Each user will correspond to a basket.
2. The contents of the basket are the programming languages this persons users.

Each record is comma delimited. Its first field is the userid, followed by all the programming languages that use has used. The languages should be ordered in lexicographical order.

Example:

```
1324763,c#,javascript
132475,php,sql
1324631,c,c++,java,matlab,postscript,python
```

Convert the input csv files to a file called `baskets.txt`. You can use SQL (sqlite) and/or Scala (including Spark. Document the process you used to create this file in a reproducible way (i.e. somebody else should be able to do it again).

3.3 Part 2: Frequent Basket Analysis

We will use spark to do frequent basket analysis. Specifically, we will use its implementation of the FP-growth algorithm. It is very similar to the a-priori algorithm. Use the following parameters:

1. Minimum support 0.02
2. Confidence = 0.5

Write a program in Scala Spark that:

1. takes as input (via the command line) the file generated in 3.4
2. outputs the most frequent sets of programming languages
   (a) each set should be in lexicographical order
   (b) languages are separated by comma (csv style)
   (c) **do not output entries of one language**.
3. orders the output:

(a) by frequency, descending

(b) then by language, ascending

Example:

```
91348,javascript,php
62915,java,javascript
56425,c#,javascript
36933,javascript,sql
35952,java,php
35350,javascript,python
```

Modify the file `baskets.scala` with your solution to this part. The input to this program (in the command line) is the datafile created in 3.4. Its output is the list of sets as indicated above.

This program is almost identical to the one in the Spark documentation for FPGrowth. You will have to do some coding to remove the userid and format the output properly.

3.4 Part 3: Programming languages as documents: libsvm format

We want to do cluster analysis. However, one problem is that the data we have is not Euclidean. It is not even ordinal: a language X is not larger or smaller than Y nor User A is not less or bigger than user B. That makes K-means useless for our purpose.

We need a strategy to deal with this problem. Fortunately this is a problem that language processing people have been having (word A is not less or bigger than word B). We will transcode our data such that a user is a *document* and the programming languages the user uses are the *words* in the document. We will then use a clustering algorithm that has been designed to process documents: Latent Dirichlet Allocation (LDA). Given a set of words used in a document, the LDA output is a set of sets of words. Each set of words is considered a “topic”, and a topic suggests a shared theme. One major difference with K-means is that topics might overlap, while k-means clusters do not.

The assumption we make is that by treating the languages a user uses as a document, the topics found will correspond to set of languages that are likely to be used by the same user.

Documents can have thousands of potential words. For this reason a popular format to encode this data is the libsvm format. See file Spark's file `data/mllib/sample_lda_libsvm_data.txt`. Its contents are:

```
0 1:1 2:2 3:6 4:0 5:2 6:3 7:1 8:1 9:0 10:0 11:3
1 1:1 2:3 3:0 4:1 5:3 6:0 7:0 8:2 9:0 10:0 11:1
2 1:1 2:4 3:1 4:0 5:0 6:4 7:9 8:0 9:1 10:2 11:0
3 1:2 2:1 3:0 4:3 5:0 6:0 7:5 8:0 9:2 10:3 11:9
4 1:3 2:1 3:1 4:9 5:3 6:0 7:2 8:0 9:0 10:1 11:3
5 1:4 2:2 3:0 4:3 5:4 6:5 7:1 8:1 9:1 10:4 11:0
6 1:2 2:1 3:0 4:3 5:0 6:0 7:5 8:0 9:2 10:2 11:9
7 1:1 2:1 3:1 4:9 5:2 6:1 7:2 8:0 9:0 10:1 11:3
8 1:4 2:4 3:0 4:3 5:4 6:2 7:1 8:3 9:0 10:0 11:0
9 1:2 2:8 3:2 4:0 5:3 6:0 7:2 8:0 9:2 10:7 11:2
10 1:1 2:1 3:1 4:9 5:0 6:2 7:2 8:0 9:0 10:3 11:3
11 1:4 2:1 3:0 4:0 5:4 6:5 7:1 8:3 9:0 10:1 11:0
```

Each row represents a document and should have:

1. a unique index identifying it. Use the userid.
2. a non empty list of pairs, separated by space. Each pair corresponds to a word.
3. each pair is separated by a colon.
   (a) the first number is the word (one-based).
   (b) the second number is the weight.
4. pairs should be ordered by the word index (left to right).
5. if a word in the language does not have a corresponding pair in the file, it is assumed the weight of this word is zero.

In our case the word is the programming language, and the weight can be either 0 (the programming language has not been used by the user) or 1 (the programming language has been used by the user).

For example, the following input file would indicate that user 33 has used languages 30, 108, 110, 162 and 200.

```
33 30:1 108:1 110:1 162:1 200:1
```

Create the dataset needed by spark's LDA using the libsvm format. As in Part 1 (see), You can use SQL (sqlite) and/or Scala (including Spark. Document the process you used to create this file in a reproducible way (i.e. somebody else should be able to do it again). Call this file `documents.txt`

3.5 Part 4: Clustering of languages

Using the Spark's documentation as reference [2] write a Spark program that does LDA clustering of the programming languages. In contrast to k-means, the topics we will find (sets of programming languages) are not mutually exclusive. However, LDA gives each component of a topic a weight.

1. Set the seed to 0L (**important**). This guarantees repetitive runs.
2. Set k to 25 (number of clusters)
3. Set the maximum number of iterations to 20.
4. Print only the programming languages of a cluster that have a $weight >= 0.05$
   (a) The languages in each cluster should be ordered by weight
   (b) Print the index and name of the programming language.

Note that the resulting clusters are already ordered based on their importance (result of `model.describeTopics`).
Your output should look like this:

[2] `https://spark.apache.org/docs/latest/ml-clustering.html#latent-dirichlet-allocation-lda`

```
Cluster 0

haskell,0.18927944518242928
assembly,0.16018189623866624
rust,0.1089947036818862
lisp,0.10621147697568878
ocaml,0.07085374234762805
verilog,0.06118053096699929
vhdl,0.057562438543738936

Cluster 1
[...]
```

Modify the file `topics.scala` with your solution to this Part. This program take as input (in the command line) the datafile created in 3.4 and outputs the list of clusters are indicated above.

3.6 Presentation

Create a 5 minute video where you demonstrate each of the 4 programs above and discuss your results.

Make sure you address the following 5 points:

1. The process used to create the data
2. The results of the basket analysis
3. The results of the clustering
4. Answer the following questions:
   - What are frequent items lists that include scala?
   - What topics/clusters does scala belong to?

3.7 Hints

- You will probably spend more time doing the data preparation than using the algorithms.
- The programs to do the basket analysis and clustering will be similar to the Spark tutorials. Most of your code will be to read and prepare the input and format the output.

4 Other information and restrictions

1. You are not allowed to use mutation in your program (var variables nor mutable data structures).
2. You can use any libraries you choose otherwise.

5 What to submit

Submit, via Brigthspace:

1. Part 1 and Part 3:
   - The programs you used to convert the data. The should be SQL and Scala.
   - A document describing the process you used
2. Part 1:
   - The file `baskets.txt`
3. Part 2
   - The program `basket.scala`
4. Part 3:
   - The file `documents.txt`
5. Part 2
   - The program `topics.scala`
6. A URL to your video.

Submit these files in a single zip file.