CSE8803: Big Data Analytics in Healthcare
Homework 4

Jimeng Sun

Deadline: 11:55 PM AoE, Mar 19, 2017

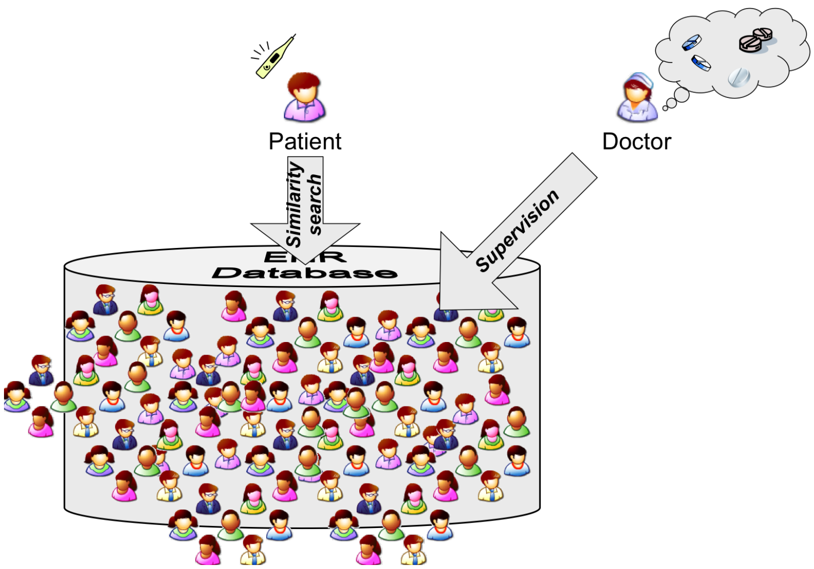
- Discussion is encouraged, but each student must write his/her own answers and explicitly mention any collaborators.
- Each student is expected to respect and follow GT Honor Code.
- Please type the submission with LaTeXor Microsoft Word. We don't accept hand written submission.
- In this homework you will also be graded on the performance of your algorithm implementation. Implement the algorithms as efficiently as possible. If we found that your Spark code is not parallel (e.g. unnecessary *collect*), we will deduct points.
- Use the programming template provided to you and DO NOT change declaration of any existing methods but you are free you add new methods.

Overview

Patients often exhibit highly complex clinical presentations in the clinic, making it difficult to determine optimal treatment solutions or understand health risks in any particular patient.

Meanwhile, electronic health record systems and health records provide rich information on aspects of patients such as diagnosis and medication histories. These data can be leveraged in order to identify patients that are similar to each other via patient similarity algorithms. The insight from patient similarity may be used for applications such as allocation of resources, determining targeted treatment plans, or constructing cohorts for predictive modeling studies.

There are several strategies for patient similarity, including graph based algorithms. In this homework, you will study related concepts and implement simple algorithms to compute patient similarity. You will be required to implement those algorithms in Spark GraphX using Scala.



Prerequisites [0 points]

For programming problem, you will be given the code skeleton in this zip file. Then you need to download data zip file from S3 inside your code folder and unzip that.

```
cd code
wget https://s3.amazonaws.com/cse8803bdh/hw4/data.tar.gz
tar -zxvf data.tar.gz
```

If you are a mac user you should be able to use below command to compile and run the code by

```
sbt/sbt compile run
```

And to run the test cases:

```
sbt/sbt test
```

Otherwise, you will need to refer to SBT installation manual to update *sbt/sbt* script first. Then you can call in above way.

1 Heterogeneous patient graph [25 points]

Graphical models are one way to represent patient EHR data. Unlike the traditional approaches for data storage, such as relational databases, graphical models can give us insight into the relations among patients, diagnosis, medications, labs, etc. Not much research has been done on using graphs in healthcare applications, needless to say, there is no existing implementation that uses Spark GraphX to construct a patient graph and perform various analyses using those new big data tools.

Implement a code that takes as input patients, diagnoses, medications, and labs and returns GraphX model that you will use in subsequent steps to perform additional tasks. Your algorithm for building the graph will take as input four data files listed below extracted from MIMIC2 database. The algorithm should be implemented in the **GraphLoader.load** function. ***Please do not modify the function declaration. You will lose points for doing so.***

The following files will be provided for you to construct the graph (ensure that those files reside in your **data** directory):

- **PATIENT.csv**: Each line represents a patient with some demographics, such as gender and age.
- **DIAGNOSTIC.csv**: Each line represents a diagnosis for a corresponding patient ID. In addition to the diagnosis and patient ID the file contains other information such as the date and diagnosis sequence (primary, secondary, etc.).
- **MEDICATION.csv**: Each line represents a medication order. The name of the medication is found in one of the columns on this file.
- **LAB.csv**: Each line represents a lab result. The name of the lab, the units for the lab, and the value for the lab are found in specific columns on this file.

Important note: every record in the diagnostic, medication and lab CSV files corresponds to an edge in the graph, representing an event. Therefore, a single patient can have multiple events related to the same diagnosis, medication or lab causing multiple edges to be created between the same patient and diagnosis. To simplify the graph, you will only create a single edge between a patient and diagnosis in the graph using the most recent event information. The same applied for medications and labs. For example, suppose we have the sample diagnostic data in the Table 1 below, you will create an edge for the event in the highlighted row only.

Table 1: Sample diagnostic data

PatientID	icd9code	encounterID	date	sequence
3	774.6	2075	211574	1
3	774.6	2099	249345	1
3	774.6	2125	507510	2
.	.	.	.	.

Your task is to use the files above to generate a bipartite graph in GraphX containing patient, diagnosis, medication and lab vertices. You will then create edges that will only connect patients to diagnosis, medication and lab. Details about each vertex and edge follows:

Patient vertex: a vertex containing patient related information stored in a *PatientProperty* class which extends *VertexProperty*. The *PatientProperty* class contains the fields:

- *patientID*
- *sex*
- *dob*: date of birth
- *dod*: date of death

Diagnostic vertex: a vertex containing diagnosis related information stored in a *DiagnosticProperty* class which extends *VertexProperty*. The *DiagnosticProperty* class contains the follow fields:

- *icd9code*: the ICD9 diagnosis code

Lab result vertex: a vertex containing lab result information stored in a *LabResultProperty* class which extends *VertexProperty*. The *LabResultProperty* class contains the fields:

- *testName*: name associated with the lab result

Medication vertex: a vertex containing medication related information stored in a *MedicationProperty* class which extends *VertexProperty*. The *MedicationProperty* class contains the fields:

- *medicine*: medication name

The graph should contain three types of edges: patient-lab, patient-diagnostic and patient-medication. Similar to the vertices, each of those edges also have properties and are defined as follows:

- **Patient-lab edge**: an edge containing information linking a patient to a lab result, which is stored in a *PatientLabEdgeProperty* class which extends *EdgeProperty*. The *PatientLabEdgeProperty* class contains *labResult* which is of *LabResult* class defined in models.

- **Patient-diagnostic edge**: an edge containing information linking a patient to a diagnosis, which is stored in a *PatientDiagnosticEdgeProperty* class which extends *EdgeProperty*. The *PatientDiagnosticEdgeProperty* class contains *diagnostic*, which is a *Diagnostic* class defined in models.

- **Patient-medication edge**: an edge containing information linking a patient to a medication, which is stored in a *PatientMedicationEdgeProperty* class which extends *EdgeProperty*. The *PatientMedicationEdgeProperty* class contains *medication*, which is a *Medication* class defined in models.

Notice that there are no edges between patients, or between diagnosis, medications and labs.
From this section you are to perform the following tasks:

- Construct patient heterogeneous graph as discussed above.
- All edges in the graph should be bi-directional.
- Make sure for patient vertices you use the **patientID** as a **VertexId** and for other types of vertices generate vertex IDs.
- Please implement your code in **GraphLoader.load()**. DO NOT change the method signature and you are allowed to add any other secondary methods that you can call from these two methods.

2 Compute Jaccard coefficient [15 points]

Jaccard coefficient is one of the simplest approaches for computing similarities among objects. For instance, given two patients each described by a set of diagnosis, medication and lab results such that $P_i = \{Dx1, Rx3, Lab6...,\}$ and $P_j = \{Lab3, Dx2, Rx5...,\}$ the Jaccard similarity between the two patients is given by

$$s_{ij} = \frac{|P_i \cap P_j|}{|P_i \cup P_j|}$$

Two patients are completely similar if $s_{ij} = 1$ and dissimilar if $s_{ij} = 0$.
Using the Jaccard similarity, you are to perform the following tasks:

- Please implement your code in **Jaccard.jaccardSimilarityOneVsAll()**. DO NOT change the method signature and you are allowed to add any other secondary methods that you can call from these two methods. ***Please do not modify the function declaration. You will lose points for doing so.***

3 Random walk with restart [20 points]

Random walk with restart (RWR) is a simple variation of PageRank. With PageRank, you start at a graph vertice and move to one of the adjacent vertices at each step. You also have a random probability where you jump to a random vertice instead of one of the adjacent vertice. With RWR, you also have a random jump probability (a.k.a reset probability), but instead of jumping to a random vertice you jump to the vertice you began with.

The RWR algorithm will compute the random walk among all vertices in the graph. If there are n patients, d diagnosis, m medications and l labs, then the output of RWR is a vector of k elements, where $k = n + d + m + l$ is the number of vertices in the graph. Refer to J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, Neighborhood formation and anomaly detection in bipartite graphs, in Fifth IEEE International Conference on Data Mining, 2005, p. 8. for more details about RWR.

- Implement RWR by completing the **RandomWalk.randomWalkOneVsAll()** method in the **RandomWalk** object. Please implement your RWR on your own. You can refer to the GraphX library but do not directly use the existing function. Your RWR by default should be able to run for 100 iterations using a reset probability of 0.15 and return only the top 10 similar patients ignoring similarities between medications, diagnostics, and labs.

4 Power Iteration Clustering [15 points]

Power iteration clustering (PIC) is a scalable and efficient algorithm for clustering vertices of a graph given pairwise similarties as edge properties. MLlib includes an implementation of PIC, which takes an RDD of (srcId, dstId, similarity) tuples and outputs a model with the clustering assignments. The similarities must be nonnegative. PIC assumes that the similarity measure is symmetric. A pair (srcId, dstId) regardless of the ordering should appear at most once in the input data. You may use print statements for debugging but comment any print statements you added before submitting.

- For this question, your task is computing pairwise similarities between all patients. Please implement your code in **Jaccard.jaccardSimilarityAllPatients()**. DO NOT change the method signature and you are allowed to add any other secondary methods that you can call from this method. In **Main.main** you will see how this method is invoked [10 points]

- Please complete **PowerIterationClustering.runPIC()**. It is just a kind of wrapper to call Spark's built-in PIC implementation. You need to pass all pair similarities you get from the previous question as input for this function. Then, you can pass it though Spark's PIC implementation with the proper configuration. Please refer to PIC doc in spark. Use three clusters and 100 for maximum iterations. You have to return the clustering result as RDD[(patientID, clusterLabel)] where the type of variables are patientID: Long and clusterLabel: Int. [5 points]

5 Zeppelin [20 points]

Apache Zeppelin is an web based notebook that enables interactive data analytics (like Jupyter). Because you can execute your code piecewise interactively, you're encouraged to use this at the initial stage your development for fast prototyping and initial data exploration. Check out the course lab pages for a brief introduction on how to set it up and use it. Please answer and provide a proper chart for each question by completing the provided JSON file, `zeppelin\bdh_hw4_zeppelin.json`. Import this notebook file on Zeppelin first.

5.1 Exploratory data analysis of dataset from the Lab

For easier start, we will read in the dataset we have been using in the lab, case.csv and control.csv, first in this part. Read carefully the provided comments in the Notebook and answer the following questions:

- Transform raw data into table [1 point]
- Make a chart for the top 10 case patient with the most payment [2 points]
- Make a chart for the top 10 control patient with the most payment [2 points]
- Make a chart for the number of case and control patients in each event type (DIAG, DRUG, PROC, etc.) [3 points]

Fill the indicated TODOs in the notebook. Please refer to the example chart below.

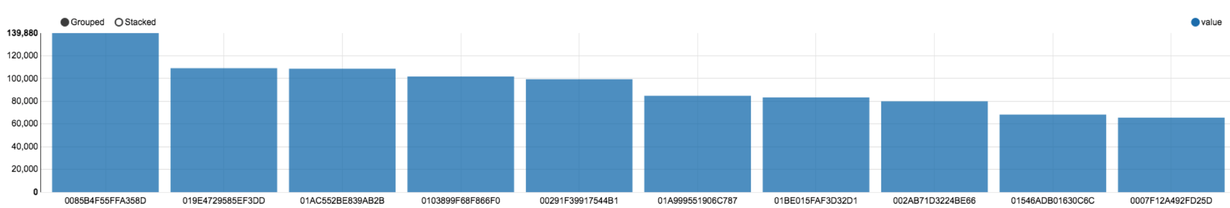


Figure 1: Example chart for the first 2 question of Q5.1

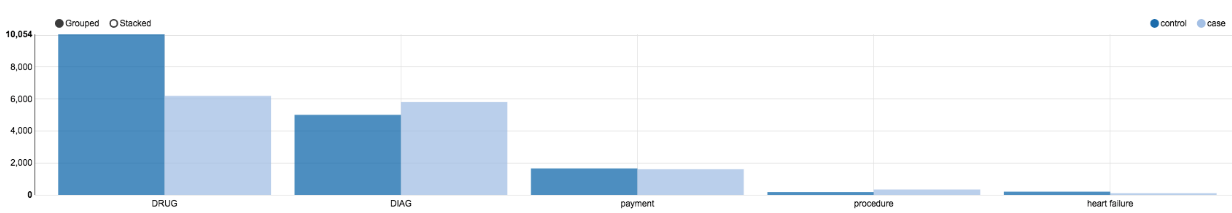
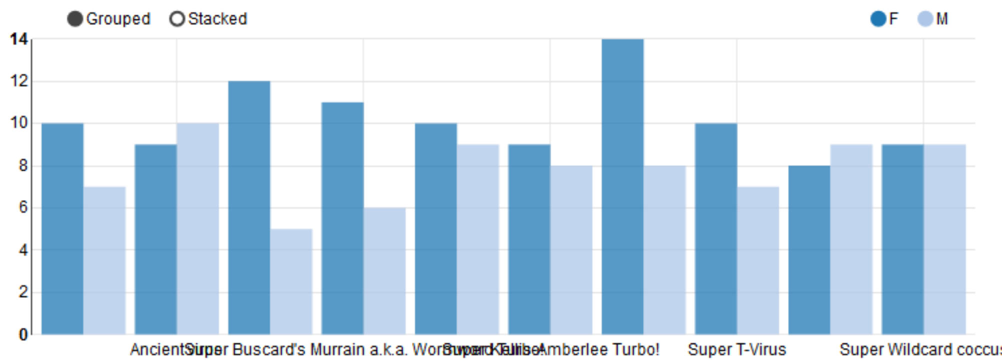


Figure 2: Example chart for the 3rd question of Q5.1

5.2 Descriptive Statistics on the raw input data for HW4

In this part, we will perform some simple descriptive statistics to understand the data. We want to find out if a particular gender is more susceptible to some common diagnoses. Specifically, read in patient and diagnostic data to answer these questions:

- Load diagnostic data from file [1 point]
- What are the number of male and female patients? Make a chart for it, e.g. pie chart, bar chart, etc. [2 points]
- Get the top 10 ICD-9 codes in diagnostics (by number of occurrences). For these top 10 codes, show the breakdown by sex. Produce a chart like below (please note that values and axis labels here are for illustrative purposes only and maybe different with the actual data): [4 points]



Fill in the indicated TODOs in the notebook.

5.3 Random walk with restart

In this part, we will visualize the results of Random walk with restart. If you could not complete **Q3 Random Walk with Restart**, please try to use the result from **Q2 Jaccard Coefficient** instead. You can manually copy the list of similar patients into Zeppelin Notebook as the easiest way, or you can save your result as a file and load it from Zeppelin if you would like to. Using the list of similar patients, please answer the following questions with proper charting:

- Make a separate table for similar patients or directly use those in the next problems if you can [1 point]
- What are the sexes of the top 10 patients most similar to patient 9? [2 points]
- How many are alive and dead? [2 points] (You can use the top 10 similar patient ids you got for patient 9 and found which one of those 10 are alive or dead.)

You are allowed to choose the most appropriate visualization to answer any of these questions.

6 Submission[5 points]

The folder structure of your submission should be as below. You may display fold structure using *tree* command. All other unrelated files will be discarded during testing. You could add additional methods, additional dependencies, but make sure existing methods signature doesn't change. It's your duty to make sure your code is compilable with provided sbt.

```
<your gtid>-<your gt account>-hw4
|-- build.sbt
|-- project
|   |-- build.properties
|   \-- plugins.sbt
|-- src
|   \-- main
|       \-- scala
|           \-- edu
|               \-- gatech
|                   \-- cse8803
|                       |-- clustering
|                       |   \-- PowerIterationClustering.scala
|                       |-- graphconstruct
|                       |   \-- GraphLoader.scala
|                       |-- ioutils
|                       |   \-- CSVUtils.scala
|                       |-- jaccard
|                       |   \-- Jaccard.scala
|                       |-- main
|                       |   \-- Main.scala
|                       |-- model
|                       |   \-- models.scala
|                       \-- randomwalk
|                           \-- Randomwalk.scala
|-- zeppelin
    \-- bdh_hw4_zeppelin.json
```

Create a tar archive of the folder above with the following command and submit the tar file.

```
tar -czvf <your gtid>-<your gt account>-hw4.tar.gz \
  <your gtid>-<your gt account>-hw4
```