CSE6250: Big Data Analytics in Healthcare
Homework 3

Jimeng Sun

Deadline: Oct 3, 2022, 8:00 AM EST

- 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 the GT Honor Code.
- Please type the submission with LaTeX or Microsoft Word. We **will not** accept hand written submissions.
- Please **do not** change the filenames and function definitions in the skeleton code provided, as this will cause the test scripts to fail and subsequently no points will be awarded.

Overview

Accurate knowledge of a patient's disease state is crucial to proper treatment, and we must understand a patient's phenotypes (based on their health records) to predict their disease state. There are several strategies for phenotyping including supervised rule-based methods and unsupervised methods. In this homework, you will implement both type of phenotyping algorithms using Spark.

Prerequisites [0 points]

This homework is primarily about using Spark with Scala. We strongly recommend using our bootcamp virtual environment setup to prevent compatibility issues.

However, since we use the Scala Build Tool (SBT) as integrate tool, and use JVM as the runtime environment, you should be fine running it on your local machine.

Please see the build.sbt file for the full list of dependencies and versions.

Begin the homework by downloading the hw3.zip from Canvas, which includes the skeleton code and test cases.

You should be able to immediately begin compiling and running the code with the following command (from the *code/* folder):

```
sbt compile run
```

And you can run the test cases with this command:

```
sbt compile test
```

1 Programming: Rule based phenotyping [35 points]

Phenotyping can be done using a rule-based method. The Phenotype Knowledge Base (PheKB) provides a set of rule-based methods (typically in the form of decision trees) for determining whether or not a patient fits a particular phenotype.

In this assignment, you will implement a phenotyping algorithm for type-2 diabetes based on the flowcharts below. The algorithm should:

- Take as input event data for diagnoses, medications, and lab results.
- Return an RDD of patients with labels (*label*=1 if the patient is case, *label*=2 if the patient is control, *label*=3 otherwise).

You will implement the *Diabetes Mellitus Type 2* algorithms from PheKB. We have reduced the rules for simplicity, which you can find in the images below. However, you can refer to the full description for more details if desired.

The following files in *code/data/* folder will be used as inputs:

- **encounter_INPUT.csv**: Each line represents an encounter and contains a unique encounter ID, the patient ID (Member_ID), and many other details about the counter. *Hint: sql join*
- **encounter_dx_INPUT.csv**: Each line represents an encounter and contains any resulting diagnoses including a description and ICD9 code.
- **medication_orders_INPUT.csv**: Each line represents a medication order including the name of the medication.
- **lab_results_INPUT.csv**: Each line represents a lab result including the name of the lab (Result_Name), the units of the lab output, and lab output value.

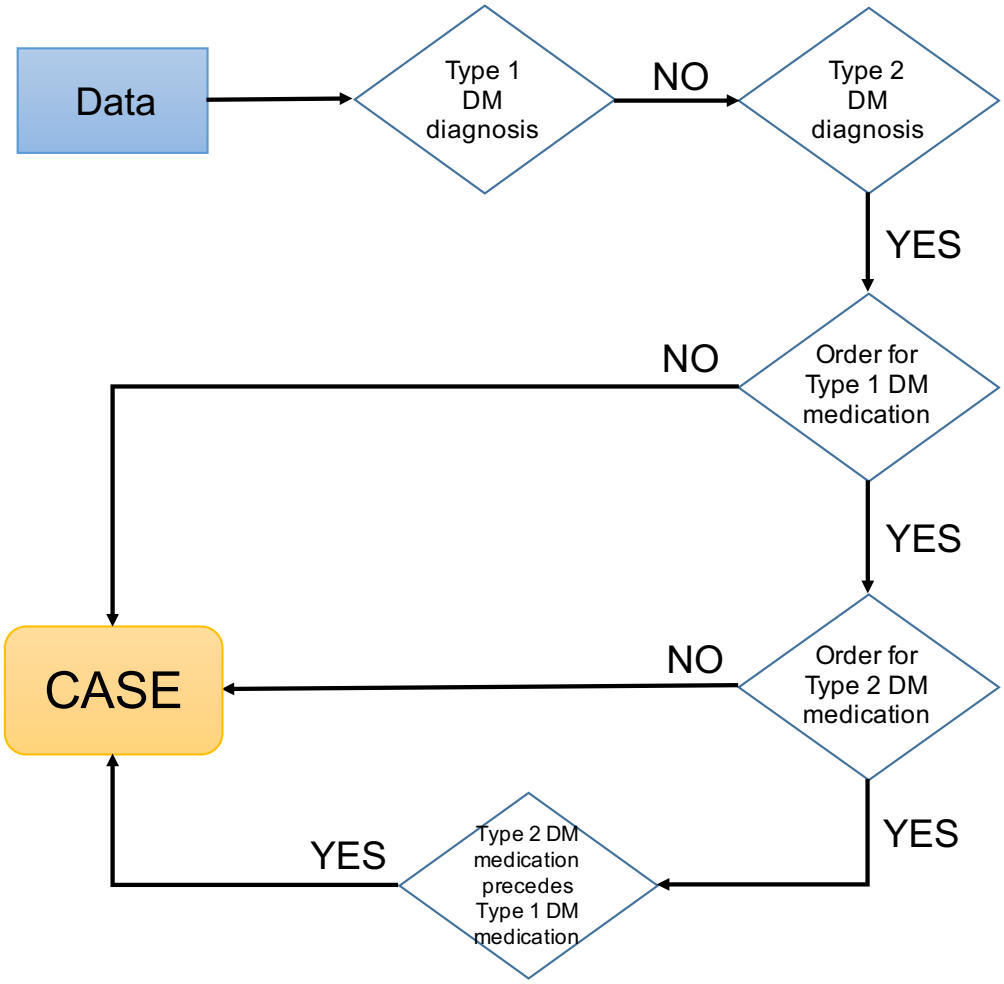

Figure 1: Determination of cases

For your project, you will load input CSV files from the *code/data/* folder. You are responsible for transforming the .csv's from this folder into RDDs.

The simplified rules which you should follow for phenotyping of Diabetes Mellitus Type 2 are shown below. These rules are based on the criteria from the PheKB phenotypes, which have been placed in the *phenotyping_resources/* folder.

- **Requirements for Case patients**: Figure 1 details the rules for determining whether a patient is case. Certain parts of the flowchart involve criteria that you will find in the *phekb_criteria/* folder as outlined below:

  - **T1DM_DX.csv**: Any ICD9 codes present in this file will be sufficient to result in `YES` for the *Type 1 DM diagnosis* criteria.
  - **T1DM_MED.csv**: Any medications present in this file will be sufficient to result in `YES` for the *Order for Type 1 DM medication* criteria. Please also use this list for the *Type 2 DM medication preceeds Type 1 DM medication* criteria.

- **T2DM_DX.csv**: Any of the ICD9 codes present in this file will be sufficient to result in `YES` for the *Type 2 DM diagnosis* criteria.
- **T2DM_MED.csv**: Any of the medications present in this file will be sufficient to result in `YES` for the *Order for Type 2 DM medication* criteria. Please also use this list for the *Type 2 DM medication preceeds Type 1 DM medication* criteria.

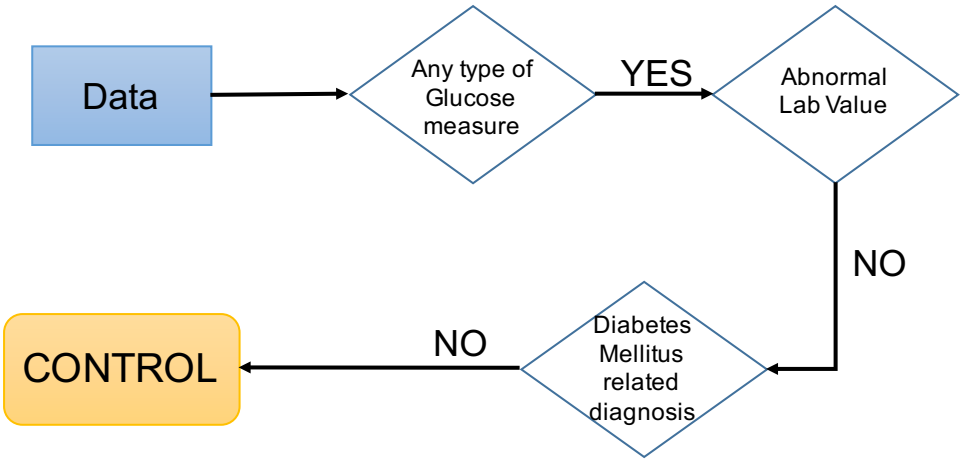

Figure 2: Determination of controls

- **Requirements for Control patients**: Figure 2 details the rules for determining whether a patient is control. Certain parts of the flowchart involve criteria that you will find in the *phekb_criteria/* folder as outlined below:
  - **ABNORMAL_LAB_VALUES_CONTROL.csv**: Any values described in this file should be considered abnormal for the *Abnormal Lab Value* criteria.
  - **DM_RELATED_DX.csv**: Any ICD9 codes present in this file will be sufficient to result in `YES` for the *Diabetes Mellitus related diagnosis* criteria.

In order to help you verify your steps, expected counts along the different steps have been provided in:

- `phenotyping_resources/expected_count_case.png`
- `phenotyping_resources/expected_count_control.png`

Any patients not found to be in the **control** or **case** category should be placed in the **unknown** category. Additional hints and notes are provided directly in the code comments, so please read these carefully.

a. Implement *edu.cse6250.main.Main.loadRddRawData* to load the input .csv files in the data folder as structured RDDs. [5 points]

b. Implement *edu.cse6250.phenotyping.T2dmPhenotype* to:
- Correctly identify case patients [7 points]
- Correctly identify control patients [7 points]
- Correctly identify unknown patients [7 points]

c. Implement *edu.cse6250.phenotyping.stat_calc* to:
- Mean of "Glucose" lab test results for case group [3 points]
- Mean of "Glucose" lab test results for control group [3 points]
- Mean of "Glucose" lab test results for unknown group [3 points]

You can use *edu.cse6250.phenotyping.T2dmPhenotypeTest.scala* for checking the results. To pass the test, your must identify all groups **b** correctly and calculate the specific statistics with relative error no greater than 1.0% (including computional error). Hints and notes are provided directly in the code comments.

2 Programming: Unsupervised Phenotyping via Clustering [60 points]

At this point you have implemented a supervised, rule-based phenotyping algorithm. This type of method is great for picking out specific diseases, in our case diabetes, but they are not good for discovering new, complex phenotypes. Such phenotypes can be disease subtypes (i.e. severe hypertension, moderate hypertension, mild hypertension) or they can reflect combinations of diseases that patients may present with (e.g. a patient with hypertension and renal failure). This is where unsupervised learning comes in.

2.1 Feature Construction [17 points]

You will need to start by constructing features out of the raw data to feed into the clustering algorithms. You will need to implement ETL using Spark with similar functionality as what you did in last homework using Pyspark. Since you know the diagnoses (in the form of ICD9 codes) each patient exhibits and the medications they took, you can aggregate this information to create features. Using the RDDs that you created in *edu.cse6250.main.Main.loadRddRawData*, you will construct features for the COUNT of medications, COUNT of diagnoses, and AVERAGE lab test value.

a. Implement the feature construction code in *edu.cse6250.features.FeatureConstruction* to create two types of features: one using all the available ICD9 codes, labs, and medications,

and another using only features related to the phenotype. See the comments of the source code for details.

2.2 Evaluation Metric [8 points]

Purity is a metrics to measure the quality of clustering, it's defined as

$$purity(\Omega, C) = \frac{1}{N} \sum_{k} \max_{j} |w_k \cap c_j|$$

where N is the number of samples, k is index of clusters and j is index of class. w_k denotes the set of samples in k-th cluster and c_j denotes set of samples of class j.

a. Implement the *getPurity* function in *edu.cse6250.clustering.Metrics*

2.3 K-Means Clustering [8 points]

Now you will perform clustering using Spark's MLLib, which contains an implementation of the k-means clustering algorithm as well as the Gaussian Mixture Model algorithm.

From the clustering, we can discover groups of patients with similar characteristics. You will cluster the patients based upon diagnoses, labs, and medications. If there are d distinct diagnoses, l distinct labs and m distinct medications, then there should be $d + l + m$ distinct features.

a. Implement k-means clustering for $k = 3$. Follow the hints provided in the skeleton code in *edu.cse6250.main.Main.scala:testClustering*. [5 points]

b. Compare clustering for the $k = 3$ case with the ground truth phenotypes that you computed for the rule-based PheKB algorithms. Specifically, for each of *case*, *control* and *unknown*, report the percentage distribution in the three clusters for the two feature construction strategies. Report the numbers in the format shown in Table 1 and Table 2. [3 points]

Percentage Cluster	Case	Control	Unknown
Cluster 1	x%	y%	z%
Cluster 2	xx%	yy%	zz%
Cluster 3	xxx%	yyy%	zzz%
	100%	**100%**	**100%**

Table 1: Clustering with 3 centers using all features

Percentage Cluster	Case	Control	Unknown
Cluster 1	x%	y%	z%
Cluster 2	xx%	yy%	zz%
Cluster 3	xxx%	yyy%	zzz%
	100%	**100%**	**100%**

Table 2: Clustering with 3 centers using filtered features

2.4 Clustering with Gaussian Mixture Model (GMM) [8 points]

a. Implement GaussianMixture for $k = 3$. Follow the hints provided in the skeleton code in *edu.cse6250.main.Main.scala:testClustering*. [5 points]

b. Compare clustering for the $k = 3$ case with the ground truth phenotypes that you computed for the rule-based PheKB algorithms. Specifically, for each of *case*, *control* and *unknown*, report the percentage distribution in the three clusters for the two feature construction strategies. Report the numbers in the format shown in Table 1 and Table 2. [3 points]

2.5 Clustering with Streaming K-Means [11 points]

When data arrive in a stream, we may want to estimate clusters dynamically and update them as new data arrives. Spark's MLLib provides support for the streaming k-means clustering algorithm that uses a generalization of the mini-batch k-means algorithm with **forgetfulness**.

a. Show why we can use streaming K-Means by deriving its update rule and then describe how it works, the pros and cons of the algorithm, and how the forgetfulness value balances the relative importance of new data versus past history. [3 points]

b. Implement StreamingKMeans algorithm for $k = 3$. Follow the hints provided in the skeleton code in *edu.cse6250.main.Main.scala:testClustering*. [5 points]

c. Compare clustering for the $k = 3$ case with the ground truth phenotypes that you computed for the rule-based PheKB algorithms. Specifically, for each of *case*, *control* and *unknown*, report the percentage distribution in the three clusters for the two feature construction strategies. Report the numbers in the format shown in Table 1 and Table 2. [3 points]

2.6 Discussion on K-means and GMM [8 points]

We'll now summarize what we've observed in the preceeding sections:

a. Briefly discuss and compare what you observed in 2.3b using the k-means algorithm and 2.4b using the GMM algorithm. [3 points]

b. Re-run k-means and GMM from the previous two sections for different k (you may run it each time with different k). Report the purity values for all features and the filtered features for each k by filling in Table 3. Discuss any patterns you observed, if any. [5 points]

NOTE: Please change k back to 3 in your final code deliverable!

k	K-Means All features	K-Means Filtered features	GMM All Features	GMM Filtered features
2				
5				
10				
15				

Table 3: Purity values for different number of clusters

3 Submission [5 points]

The folder structure of your submission should be as below or your code will not be graded. Please **discard** all other **unrelated** files following the below structure. You can display fold structure using *tree* command. You may add additional methods, additional dependencies, but make sure existing methods signature doesn't change. It's your duty to make sure your code can be compiled with the provided SBT. Be aware that writeup is within code root.

```
<your gtid>-<your gt account>-hw3
|-- homework3_answer.pdf
|-- build.sbt
|-- project
|   |-- build.properties
|   |-- CompileSettings.scala
|   \-- plugins.sbt
|-- sbt
|   \-- sbt
\-- src
    \-- main
        |-- java(optional)
        |-- resources(optional)
        \-- scala
            \-- edu
                \-- cse6250
                    |-- clustering
```

```
|   |-- Metrics.scala
|   \-- package.scala
|-- features
|   \-- FeatureConstruction.scala
|-- helper
|   \-- CSVHelper.scala
|   \-- SessionCreator.scala
|-- main
|   \-- Main.scala
|-- model
|   \-- models.scala
\-- phenotyping
    \-- T2dmPhenotype.scala
```

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

```
tar -czvf <your gtid>-<your gt account>-hw3.tar.gz \
  <your gtid>-<your gt account>-hw3
```

'gtid' is the 9-digit id. 'gt account' is the prefix for your original gt email address, for example 'abc123'. It is not your alias gt email prefix. Please double check and make sure your 'gt account' is correct.