CSE8803: Big Data Analytics in Healthcare
Homework 2

Jimeng Sun

Deadline: February 14, 2016

- 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.
- Please do not change the names and function definitions in the skeleton code provided, as this will cause the test scripts to fail and subsequently no points will be awarded. Built-in modules of python and the following libraries - numpy, scipy, scikit-learn can be used.

Overview

Accurate knowledge of a patient's condition is critical. Electronic monitoring systems and health records provide rich information for performing predictive analytics. In this homework, you will use ICU clinical data to predict the mortality of patients in one month after discharge.

It is your responsibility to make sure that all code and other deliverables are in the correct format and that your submission compiles and runs. We will not manually check your code. Thus non-runnable code will directly lead to 0 score.

About Code Skeleton and Raw Data

You need to download the homework2 zip from T-Square. After unzip the file, you should have below files

```
hw2
|-- data
|  |-- events.csv
|  \-- mortality.csv
|-- homework2.pdf
\-- homework2.tex
```

Download code

For this hw, we put code on github. You will need to download it via

```
cd hw2 #navigate to hw directory
git clone https://github.gatech.edu/hsu34/bdh-hw2.git code
```

About data

When you browse to the *hw2/data/*, there are two CSV files which will be the input data in this assignment.

The data provided in *events.csv* are event sequences. Each line of this file consists of a tuple with the format *(patient_id, event_id, event_description, timestamp, value)*.

For example,

```
1053,DIAG319049,Acute respiratory failure,2924-10-08,1.0
1053,DIAG197320,Acute renal failure syndrome,2924-10-08,1.0
1053,DRUG19122121,Insulin,2924-10-08,1.0
1053,DRUG19122121,Insulin,2924-10-11,1.0
1053,LAB3026361,Erythrocytes in Blood,2924-10-08,3.000
1053,LAB3026361,Erythrocytes in Blood,2924-10-08,3.690
1053,LAB3026361,Erythrocytes in Blood,2924-10-09,3.240
1053,LAB3026361,Erythrocytes in Blood,2924-10-10,3.470
```

- **patient_id**: Identifies the patients in order to differentiate them from others. For example, the patient in the example above has patient id 1053.
- **event_id**: Encodes all the clinical events that a patient has had. For example, DRUG19122121 means that a drug with RxNorm code as 19122121 was prescribed to the patient. DIAG319049 means the patient was diagnosed of disease with SNOMED

code of 319049 and LAB3026361 means that the laboratory test with a LOINC code of 3026361 was performed on the patient.

- **event_description**: Shows the description of the event. For example, DIAG319049 is the code for Acute respiratory failure and DRUG19122121 is the code for Insulin.
- **timestamp**: Indicates the date at which the event happened. Here the timestamp is not a real date but a shifted date for protecting privacy of patients.
- **value**: Contains the value associated to an event. See Table 1 for the detailed description.

event type	sample event_id	value meaning	example
diagnostic code	DIAG319049	diagnosed with a certain disease, value always be 1.0	1.0
drug consumption	DRUG19122121	prescribed a certain medication, value will always be 1.0	1.0
laboratory test	LAB3026361	test conducted on a patient and its value	3.690

Table 1: Event sequence value explanation

The data provided in *mortality_events.csv* contains the patient ids of only the deceased people. They are in the form of a tuple with the format *(patient_id, timestamp, label)*. For example,

```
37,3265-12-31,1
40,3202-11-11,1
```

The timestamp indicates the death date of a deceased person and a label of 1 indicates death. Patients that are not mentioned in this file are considered alive.

1 Logistic Regression [25 points]

1.1 Batch Gradient Descent

With a set of historical health-care data, you could train a Logistic Regression classifier to make prediction. A training set D is composed of $\{(\mathbf{x}_i, \mathbf{y}_i)\}_1^N$, where $\mathbf{y}_i \in \{0, 1\}$ is the label, and $\mathbf{x}_i \in \mathbf{R}^d$ is the feature vector of the i-th patient. In logistic regression we have $p(\mathbf{y} = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})$, where $\sigma(t) = \frac{1}{1+e^{-t}}$ is the sigmod function. The negative log-likelihood can be given as:

$$NLL\,(w) = -\sum_{i=1}^N \left[(1 - y_i)\log(1 - \sigma(\mathbf{w}^T\mathbf{x})) + y_i \log \sigma(\mathbf{w}^T\mathbf{x})\right]$$

a. Derive the gradient of the negative *log*-likelihood in terms of $\mathbf{w}$ for this setting. [5 points]

1.2 Stochastic Gradient Descent

Suppose your system continuously collects patient data and predicts their severity using Logistic Regression. When patient data vector $\mathbf{x}$ arrives to your system, the system needs to predict whether he/she has a severe condition (predicted label $\hat{y} \in \{0, 1\}$) and requires immediate care or not. The result of the prediction will be delivered to a physician, who can then take a look at the patient. Finally, the physician will provide a feedback(real label $y \in \{0, 1\}$) to your system, so that the system can be upgraded to make better predictions in the future. Here, you are going to derive the equation behind this setting.

a. Show the log likelihood l of a $(\mathbf{x}_t, \mathbf{y}_t)$ pair. [5 points]

b. Show how to update the coefficient vector $\mathbf{w}_t$ when you get a patient feature vector $\mathbf{x}_t$ and physician feedback label $\mathbf{y}_t$ at time t using $\mathbf{w}_{t-1}$(suppose learning rate η is given)? [5 points]

c. What's the time complexity of the update rule from **b** if $\mathbf{x}_t$ is very sparse? [2 points]

d. Briefly explain the consequence of using a very large η and very small η. [3 points]

e. Show how to update $\mathbf{w}_t$ under the penalty of L2 norm regularization. In other words, update $\mathbf{w}_t$ according to $l - \mu\|\mathbf{w}\|^2$, where μ is a constant. What's the time complexity? [5 points]

f. When you use L2 norm, you will find each time you get a new $(\mathbf{x}_t, \mathbf{y}_t)$ you need to update every element of vector $\mathbf{w}_t$ even if $\mathbf{x}_t$ has very few non-zero elements. Write the pseudo-code on how to update $\mathbf{w}_t$ lazily. [Extra 5 points]

HINT: Update i-th element of $\mathbf{w}_t$, $\mathbf{w}_{ti}$, only when i-th element of $\mathbf{x}_t$, $\mathbf{x}_{ti}$, is non-zero, and you can refer to Sec.10 and 11 and the appendix of this paper.

2 Programming [75 points]

First, follow the instructions to install the environment if you haven't done that yet. Then you need to use the *hw2/data/* from T-Square.

2.1 Descriptive Statistics [15 points]

Computing statistics on the data aids in developing predictive models. In this homework, you need to write HIVE code that computes various metrics on the data. A skeleton code is provided as a starting point.

The definition of terms used in the result table are described below:

- **Event count**: Number of events recorded for a given patient. Note that every line in the input file is an event.
- **Encounter count**: Count of unique dates on which a given patient visited the ICU.
- **Record length**: Duration (in number of days) between first event and last event for a given patient.
- **Common Diagnosis**: 5 Most frequently occurring disease.[1]
- **Common Laboratory Test**: 5 Most frequently conducted test.
- **Common Medication**: 5 Most frequently prescribed medication.

a. Complete *hive/event_statistics.hql* for computing statistics required in the question. Please be aware that **you are not allowed to change the filename.**

b. Use *events.csv* and *mortality.csv* provided in **data** as input and fill Table 2 with actual values.

Deliverable: code/hive/event_statistics.hql [15 points]

2.2 Transform data [25 points]

In this problem we will convert the raw data to standardized format using Pig. Diagnostic, medication and laboratory codes for each patient should be used to construct the feature vector and the feature vector should be represented in SVMLight format. You will work with *events.csv* and *mortality.csv* files provided in **data** folder.

Listed below are a few concepts you need to know before beginning feature construction (for details please refer to lectures).

[1] Count all the occurrences of the codes. e.g, if one patient has the same code 3 times, the total count on that code should include all 3.

Metric	Deceased patients	Alive patients
Event Count 1. Average Event Count 2. Max Event Count 3. Min Event Count		
Encounter Count 1. Average Encounter Count 2. Max Encounter Count 3. Min Encounter Count		
Record Length 1. Average Record Length 2. Max Record Length 3. Min Record Length		
Common Diagnosis		
Common Laboratory Test		
Common Medication		

Table 2: Descriptive statistics for alive and dead patients

- Observation Window: The time interval you will use to identify relevant events. Only events present in this window should be included while constructing feature vectors. The size of observation window is 2000 days(including 2000).
- Prediction Window: A fixed time interval that is to be used to make the prediction. Events in this interval should not be included while constructing feature vectors. The size of prediction window is 30 days.
- Index date: The day on which mortality is to be predicted. Event occured at index date should be considered within observation window. Index date is evaluated as follows:
  - For deceased patients: Index date is 30 days prior to the death date (timestamp field) in *mortality.csv*.
  - For alive patients: Index date is the last event date in *events.csv* for each alive patient.

You will work with the following files in *code/pig* folder

- **etl.pig**: Complete this script based on provided skeleton.
- **utils.py**: Implement necessary User Defined Functions (UDF) in Python in this file (optional).

In order to convert raw data from events to features, you will need a few steps:

1. *Compute the index date:*[5 points] Use the definition provided above to compute the index date for all patients.

2. *Filter events:*[5 points] Consider an observation window (2000 days) and prediction window (30 days). Remove the events that occur outside the observation window.

3. *Aggregate events:* [5 points] To create features suitable for machine learning, we will need to aggregate the events for each patient as follows:

   - **count:** occurrence for diagnostics, lab and medication events (i.e. event_id starting with DRUG, LAB and DIAG respectively) to get their counts.

   Each event type will become a feature and we will directly use event_id as feature name. For example, given below raw event sequence for a patient,

```
1053,DIAG319049,Acute respiratory failure,2924-10-08,1.0
1053,DIAG197320,Acute renal failure syndrome,2924-10-08,1.0
1053,DRUG19122121,Insulin,2924-10-08,1.0
1053,DRUG19122121,Insulin,2924-10-11,1.0
1053,LAB3026361,Erythrocytes in Blood,2924-10-08,3.000
1053,LAB3026361,Erythrocytes in Blood,2924-10-08,3.690
1053,LAB3026361,Erythrocytes in Blood,2924-10-09,3.240
1053,LAB3026361,Erythrocytes in Blood,2924-10-10,3.470
```

   We can get feature value pairs(*event_id*, *value*) for this patient with ID *1053* as

```
(DIAG319049, 1)
(DIAG197320, 1)
(DRUG19122121, 2)
(LAB3026361, 4)
```

4. *Generate feature mapping:* [5 points] In above result, you see the feature value as well as feature name(*event_id* here). Next, you need to assign an unique identifier for each feature. Sort all unique feature names in ascending alphabetical order and assign continuous feature id starting from 0. Thus above result can be mapped to

```
(2, 1)
(1, 1)
(3, 2)
(4, 4)
```

5. *Normalization:*[5 points] Further, in machine learning algorithm like logistic regression, it is important to normalize different features into the same scale using an approach like min-max normalization (hint: $min(x_i)$ maps to 0 and $max(x_i)$ 1 for feature x_i, $min(x_i)$ is zero for **count** aggregated features).

6. *Save in SVMLight format:* If the dimensionality of a feature vector is large but the feature vector is sparse (i.e. it has only a few nonzero elements), sparse representation should be employed. In this problem you will use the provided data for each patient to construct a feature vector and represent the feature vector in SVMLight format.

   ```
   <line> .=. <target> <feature>:<value> <feature>:<value>
   <target> .=. +1 | -1 | 0 | <float>
   <feature> .=. <integer> | qid
   <value> .=. <float>
   <info> .=. <string>
   ```

   For example, the feature vector in SVMLight format will look like:

   ```
   1 2:0.5 3:0.12 10:0.9 2000:0.3
   0 4:1.0 78:0.6 1009:0.2
   1 33:0.1 34:0.98 1000:0.8 3300:0.2
   1 34:0.1 389:0.32
   ```

   where, 1 or 0 will indicate whether the patient is dead or alive i.e. the label and it will be followed by a series of feature-value pairs sorted by the feature index (idx) value.

To run your pig script in local mode, you will need the command:

```
sudo pig -x local etl.pig
```

Deliverable: pig/etl.pig and pig/utils.py [**25 points**]

2.3 SGD Logistic Regression [15 points]

In this question, you are going to implement your own Logistic Regression classifier in python using the equations you derived in question **1.2.e**. To help you get started, we have provided a skeleton code. You will find the relevant code files in *lr* folder. You will train and test a classifier by running

1. cat path/to/train/data | python train.py -f <number of features>

2. cat path/to/test/data | python test.py

Training and testing data of this problem will be output from previous Pig ETL problem.

To better understand the performance of your classifier, you will need to use standard metrics like AUC. A script file named **install-conda.sh** is provided to help you install necessary modules for drawing ROC curve. The script is tested in Vagrant VM. You may need to modify it if you want to install it somewhere else. Remember to restart the terminal after installation.

a. Update the **lrsgd.py** file. You are allowed to add extra methods, but please make sure the existing method names and parameters remain unchanged. Use standard modules of Python 2.7 only, as we will not guarantee the availability of any third party modules while testing your code. [10 points]

b. Show the ROC curve generated by test.py in this writing report for different learning rates η and regularization parameters μ combination and briefly explain the result. [5 points]

c. [Extra 10 points] Implement using the result of question **1.2.f**, and show the speed up. Test efficiency of your approach using larger data set *training.data*, which has 299135 number of features. Save the code in a new file lrsgd_fast.py. We will test whether your code can finish witin reasonable amount of time and correctness of trained model. The training and testing data set can be downloaded from:

```
http://s3.amazonaws.com/cse8803/training.data
http://s3.amazonaws.com/cse8803/testing.data
```

Deliverable: lr/lrgsd.py and optional lr/lrsgd_fast.py [15 points]

2.4 Hadoop [15 points]

In this problem, you are going to train multiple logistic regression classifiers using your implementation of previous problem with Hadoop in parallel. The pseudo code of Mapper and Reducer are listed as Algorithm 1 and Algorithm 2 respectively. Find related files in *lr* folder.

input : Ratio of sampling r, number of models M, input pair (k, v)
output: key-value pairs

for $i \leftarrow 1$ **to** M **do**
    $m \leftarrow$ `Random`;
    if $m < r$ **then**
        `Emit` (i, v)
    end
end

Algorithm 1: Map function

You need to copy *training*(the output of Pig ETL) into HDFS using command line.

```
hdfs dfs -mkdir hw2
```

input : (k, v)
output: Trained model
1 Fit model on v;

Algorithm 2: Reduce function

```
hdfs dfs -put training hw2/ #  adjust training's path ondemand
```

a. complete the **mapper.py** according to pseudo code. [5 points]
You could train 5 ensembles by invoking

```
hadoop jar \
   /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -D mapreduce.job.reduces=5     \
  -files lr \
  -mapper "python lr/mapper.py -n 5 -r 0.4" \
  -reducer "python lr/reducer.py -f <number of features>" \
  -input hw2/training \
  -output hw2/models
```

Notice that you could apply other parameters to reducer. To test the performance of ensembles, copy the trained models to local via

```
hdfs dfs -get hw2/models
```

b. Complete the **testensemble.py** to generate the ROC curve. [5 points]

```
cat testing.data | python lr/testensemble.py -m models
```

c. Compare the performance with that of previous problem and briefly analyze why the difference. [5 points]

2.5 Submission [5 points]

The folder structure of your submission should be as below. You can display fold structure using *tree* command. All other unrelated files will be discarded during testing.

```
<your gtid>-<your gt account>-hw2
|-- code
| |-- hive
| | \-- event_statistics.hql
| |-- lr
| | |-- lrsgd.py
| | |-- mapper.py
| | \-- testensemble.py
| \-- pig
```

```
|   |-- etl.pig
|   \-- utils.py
\-- homework2answer.pdf
```

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

```
tar -czvf <your gtid>-<your gt account>-hw2.tar.gz \
  <your gtid>-<your gt account>-hw2
```

Example submission: 901234567-gburdell3-hw1.tar.gz