CSE6250: Big Data Analytics in Healthcare
Homework 2

Jimeng Sun

Deadline: February 4, 2018, 11:55 PM AoE

- 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

Begin by downloading and extracting the Homework 2 tar file from Canvas. You should then see the file structure shown below:

```
hw2
|-- code
| |-- environment.yml
| |-- hive
| | \-- event_statistics.hql
| |-- lr
| | |-- lrsgd.py
| | |-- mapper.py
| | |-- reducer.py
| | |-- test.py
| | |-- testensemble.py
| | |-- train.py
| | \-- utils.py
| |-- pig
| |  |-- etl.pig
| |  \-- utils.py
| |-- zeppelin
| |   |-- bdh_hw2_zeppelin.json
| \-- sample_test
|    |-- sample_events.csv
|    |-- sample_mortality.csv
|    |-- expected
|    |  |-- hive
|    |  |    \-- statistics.txt
|    |  |-- pig
|    |  |    |-- aliveevents.csv
|    |  |    |-- deadevents.csv
|    |  |    |-- features
|    |  |    |-- features_aggregate.csv
|    |  |    |-- features_map.csv
|    |  |    |-- features_normalized.csv
|    |  |    |-- filtered.csv
|    |  |    \-- samples
|    |
|    |
|-- data
|    |-- events.csv
```

```
|    \-- mortality.csv
|-- homework2.pdf
\-- homework2.tex
```

About data

When you browse to the *hw2/data*, there are two CSV files which will be the input data in this assignment, *events.csv* and *mortality.csv*, as well as two smaller versions of each file that can be used for your debugging in the *code/sample_test* folder, *sample_events.csv* and *sample_mortality.csv*.

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 19122121 was prescribed to the patient, DIAG319049 means the patient was diagnosed with a disease with SNOMED code 319049, and LAB3026361 means that a laboratory test with LOINC code 3026361 was performed on the patient.
- **event_description**: Shows the text 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 to protect the 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	diagnosis was confirmed (all records will have value 1.0)	1.0
drug consumption	DRUG19122121	drug was prescribed (all records will have value 1.0)	1.0
laboratory test	LAB3026361	lab result from running this test on the patient	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]

A Logistic Regression classifier can be trained with historical health-care data to make future predictions. A training set D is composed of $\{(\mathbf{x}_i, y_i)\}_1^N$, where $y_i \in \{0, 1\}$ 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 $p(y_i = 1|\mathbf{x}_i) = \sigma(\mathbf{w}^T\mathbf{x}_i)$, where $\mathbf{w} \in \mathbf{R}^d$ is the learned coefficient vector and $\sigma(t) = \frac{1}{1+e^{-t}}$ is the sigmoid function.

Suppose your system continuously collects patient data and predicts patient severity using Logistic Regression. When patient data vector $\mathbf{x}$ arrives to your system, the system needs to predict whether the patient 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 feedback (truth label $y \in \{0, 1\}$) back to your system so that the system can be upgraded, i.e. $\mathbf{w}$ recomputed, to make better predictions in the future.

NOTE: We will not accept hand-written, screenshots, or other images for the derivations in this section. Please use Microsoft Word or Latex and convert to PDF for your final submission.

1.1 Batch Gradient Descent

The negative log-likelihood can be calculated according to

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

The maximum likelihood estimator $\mathbf{w}_{\text{MLE}}$ can be found by solving for $\arg\min_{\mathbf{w}} NLL$ through an iterative gradient descent procedure.

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

1.2 Stochastic Gradient Descent

If N and d are very large, it may be prohibitively expensive to consider every patient in D before applying an update to $\mathbf{w}$. One alternative is to consider stochastic gradient descent, in which an update is applied after only considering a single patient.

a. Show the log likelihood, l, of a single $(\mathbf{x}_t, y_t)$ 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 y_t at time t using $\mathbf{w}_{t-1}$ (assume learning rate η is given). [5 points]

c. What is 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 $l - \mu\|\mathbf{w}\|_2^2$, 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 $(\mathbf{x}_t, y_t)$ 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] **(no partial credit!)**

HINT: Update j-th element of $\mathbf{w}_t$, $\mathbf{w}_{tj}$, only when j-th element of $\mathbf{x}_t$, $\mathbf{x}_{tj}$, is non-zero. 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. You will also need the *hw2/data/* from Canvas.

You will then need to install Python 2 for this homework due to limitations in Hadoop's Streaming Map Reduce interface. To install Python 2, connect to your Docker instance and begin by downloading the Anaconda 2 installation script and kicking it off with the commands below:

```
curl https://repo.continuum.io/archive/Anaconda2-5.0.1-Linux-x86_64.sh > conda2.sh
```

Kickoff the installation using *bash conda2.sh*. Follow the prompts, and when it asks if you want to change the installation path, enter */usr/local/conda2* instead of the default. You may chose to add this Python path to your bash profile when prompted, or you can manually call Python 2 using */usr/local/conda2/bin/python* for the remainder of the assignment.

Please ensure you are always using Python 2 for the remainder of the assignment or you may encounter problems!

2.1 Descriptive Statistics [10 points]

Computing descriptive statistics on the data helps in developing predictive models. In this section, 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.
- **Common Laboratory Test**: 5 most frequently conducted test.
- **Common Medication**: 5 most frequently prescribed medications.

While counting common diagnoses, lab tests and medications, 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. Furthermore, the count is not per patient but per code.

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. We only need the top 5 codes for common diagnoses, labs and medications. Their respective counts are not required.

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. Median Record Length 3. Max Record Length 4. Min Record Length		
Common Diagnosis		
Common Laboratory Test		
Common Medication		

Table 2: Descriptive statistics for alive and dead patients

Deliverable: code/hive/event_statistics.hql [10 points]

2.2 Transform data [20 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).

- **Observation Window:** The time interval containing events you will use to construct your feature vectors. Only events in this window should be considered. The observation window ends on the index date (defined below) and starts 2000 days (including 2000) prior to the index date.
- **Prediction Window:** A fixed time interval following the index date where we are observing the patient's mortality outcome. This is to simulate predicting some length of time into the future. 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 we will predict the patient's probability of dying during the subsequent prediction window. Events occurring on the index date should be considered within the observation window. Index date is determined 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:* [4 points] Use the definition provided above to compute the index date for all patients.
2. *Filter events:* [4 points] Consider an observation window (2000 days) and prediction window (30 days). Remove the events that occur outside the observation window.
3. *Aggregate events:* [4 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:* [4 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

```
(1, 1)
(0, 1)
(2, 2)
(3, 4)
```

5. *Normalization:* [4 points] In machine learning algorithms like logistic regression, it is important to normalize different features into the same scale. Implement min-max normalization on your results. (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 shown below:

```
<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 target), 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 [20 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

The training and testing data for 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. Similarly with Homework 1, we provide *code/environment.yml* which contains a list of libraries needed to setup the environment for this homework. You can use it to create a copy of conda 'environment' (http://conda.pydata.org/docs/using/envs.html#use-environment-from-file). If you already have your own Python development environment (it should be Python 2.7), please refer to this file to find necessary libraries. It will help you

install necessary modules for drawing an ROC curve. 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 **(no partial credit!)**] 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 5675 possible distinct 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:

```
https://s3.amazonaws.com/6250bdh/hw2/training.data
https://s3.amazonaws.com/6250bdh/hw2/testing.data
```

Deliverable: lr/lrsgd.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. We have already written the code for the reducer for you. 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 ← 1 to M do
    m ←Random;
    if m < r then
        Emit (i, v)
    end
end
```

Algorithm 1: Map function

input : (k, v)
output: Trained model

```
Fit model on v;
```

Algorithm 2: Reduce function

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

```
hdfs dfs -mkdir /hw2
hdfs dfs -put pig/training /hw2  # adjust train path as needed
```

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 pig/testing/* | python lr/testensemble.py -m models
```

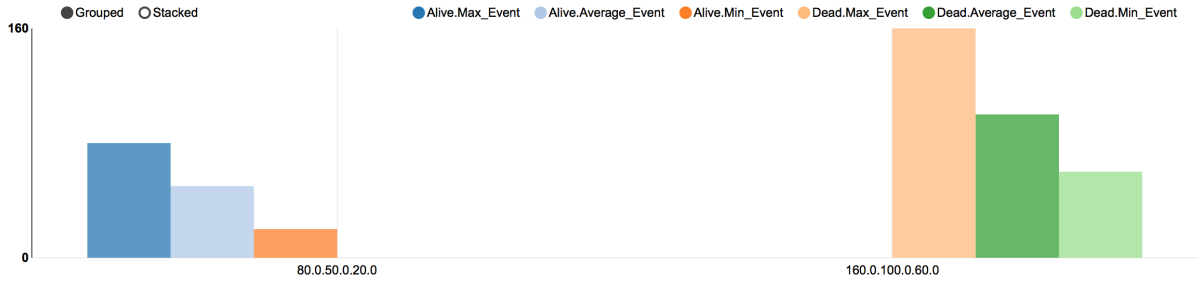
c. Compare the performance with that of previous problem and briefly analyze why the difference. [5 points]

2.5 Zeppelin [10 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 fill in the TODO section of the JSON file, `zeppelin\bdh_hw2_zeppelin.json`. Import this notebook file on Zeppelin first. For easier start, we will read in the dataset we have been using in the lab, events.csv and mortality.csv, first in this part. Read carefully the provided comments in the Notebook and fill-in the TODO section:

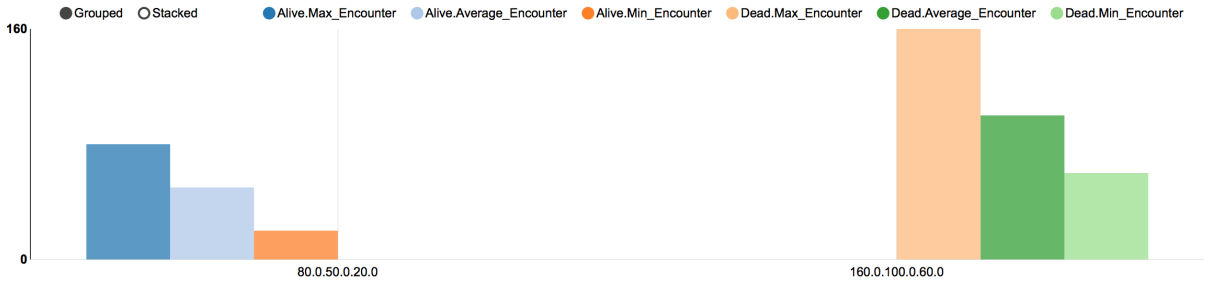
- **Event count**: Number of events recorded for a given patient. Note that every line in the input file is an event. [3 points]
- For the average, maximum and minimum event counts, show the breakdown by dead or alive. Produce a chart like below (please note that values and axis labels here are

for illustrative purposes only and maybe different with the actual data): [2 points]



- **Encounter count**: Count of unique dates on which a given patient visited the ICU. All the events: DIAG, LAB and DRUG - should be considered as ICU visiting events.[3 points]

- For the average, maximum and minimum encounter counts, show the breakdown by dead or alive. 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): [2 points]



Fill in the indicated TODOs in the notebook using ONLY Scala.

Please be aware that **you are NOT allowed to change the filename and any existing function declarations.**

The submission bdh_hw2_zeppelin.json file should have all cells run and charts visible. Please don't clear the cell output before submission.

Deliverable: `zeppelin\bdh_hw2_zeppelin.json`**[10 points]**

2.6 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
| | |-- lrsgd_fast.py (Optional)
| | |-- mapper.py
| | \-- testensemble.py
| |-- pig
| |  |-- etl.pig
| |  \-- utils.py
| \-- zeppelin
|    \-- bdh_hw2_zeppelin.json
\-- homework2_answer.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-hw2.tar.gz