CSE6250: Big Data Analytics in Healthcare
Homework 2

Jimeng Sun

Deadline: Sep 19, 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 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:

```
homework2
|-- src
|   |-- event_statistics.py
|   |-- etl.py
|   |-- lr
|   |   |-- lrsgd.py
|   |   |-- train.py
|   |   |-- test.py
|   |   |-- utils.py
|-- tests
|   |-- test_statistics.py
|   |-- test_etl.py
|-- sample_test
|   |-- sample_events.csv
|   |-- sample_mortality.csv
|-- data
|    |-- events.csv
|    |-- mortality.csv
|-- deliverables
|-- environments.yml
|-- homework2.pdf
|-- homework2.tex (optional)
```

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 (Column Name: patientid, Data Type: Integer).
- **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 (Column Name: eventid, Data Type: String).
- **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 (Column Name: eventdesc, Data Type: String).
- **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 (Column Name: etimestamp, Data Type: Date).
- **value**: Contains the value associated to an event. See Table 1 for the detailed description (Column Name: value, Data Type: Float).

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 (Column Name: mtimestamp, Data Type: Date) and a label of 1 indicates death (Column Name: label, Data Type: Integer). Patients that are not mentioned in this file are considered alive.

Environment setup

The environment we will be using in this homework is basically pyspark and some python libraries for machine learning. We have provided you with *hw2/environment.yml* to set up your environment. For Mac/Linux, you can directly use it to create a conda 'environment' (http://conda.pydata.org/docs/using/envs.html#use-environment-from-file).

Besides, you need to use conda-forge to manually install pyspark(version must be 3.3.0) in this environment:

```
conda install -c conda-forge pyspark=3.3
```

Remember to check the version of pyspark before proceeding with the homework. (How to use conda-forge to install pyspark).

For windows 10, you need extra steps of install java, apache-spark, hadoop winutils, and add/edit environment variables, you may refer this instruction: (set up pyspark for windows)

Running the tests

Test cases are provided for every module in this homework and operate on handmade data. To run a test, execute the following commands from the base folder i.e. homework2. If any of the test cases fail, an error will be shown. For example to test the statistics computed, the following command should be executed:

```
nosetests tests/test_etl.py --nologcapture
```

A single test can also be run using this syntax:

```
nosetests tests/<filename>:<test_method> --nologcapture
```

Remember to use the right *filename* and *test_method* in above command line. For more information about basic usage of nosetests, please refer to this LINK

1 Logistic Regression [27 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.
- Question 1 is closely related with later programming questions, you will use derived formulas from this question to code the map-reduce. So be careful here.

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 w_j which is a dimension $\mathbf{w}$ for this setting, i.e., $\frac{\partial NNL(D,\mathbf{w})}{\partial w_j}$. [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. You do not need to consider regularization until 1.2.e.

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

b. Derive $\frac{\partial l}{\partial w_j}$. Show how to update the coefficient vector w_t^j when you get a patient feature vector $\mathbf{x}_t$ and physician feedback label y_t at time t using w_{t-1}^j (assume learning rate η is given). [5 points] (**NOTE:** The log likelihood function in **a.** is positive, different from the NLL in **1.1**, so you should update the coefficient to maximize the positive log likelihood. You can compare with minimizing NLL, the results should be the same)

c. Suppose m is the total number of features (regardless of whether they are non-zero), n is the total number of non-zero features for each sample and T is the number of iterations. What is the smallest time complexity (in big-O notation) of the update rule from **b** if $\mathbf{x}_t$ by all iterations? What if dimension is very sparse, i.e., n is small constant, what can the complexity be (in big-O notation)? [4 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. The learning rate η should be applied to $\frac{\partial}{\partial w_j}(l - \mu\|\mathbf{w}\|_2^2)$. What's the time complexity (use the same notation from **c**)? [5 points]

2 Programming [68 points]

Make sure your environment is set up right. Version control is crucial to this homework.

2.1 Descriptive Statistics [15 points]

Computing descriptive statistics on the data helps in developing predictive models. In this section, you need to write PySpark 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 [3 points]**: Number of events recorded for a given patient. Note that every line in the input file is an event.
- **Encounter Count [3 points]**: Count of unique dates on which a given patient visited the ICU.
- **Record Length [3 points]**: Duration (in number of days) between first event and last event for a given patient.
- **Common Diagnosis [2 points]**: 5 most frequently occurring disease.
- **Common Laboratory Test [2 points]**: 5 most frequently conducted test.
- **Common Medication [2 points]**: 5 most frequently prescribed medications.

While counting common diagnoses, lab tests and medications, count all the occurrences of the codes and sort them by descending counts. 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 *src/event_statistics.py* for computing statistics required in the question.(Just keep the original results of each statistics, do not round any results) [9 points]. 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 (you can keep two decimal places for float numbers when fill the form) [6 points]. 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. Median Record Count 3. Max Encounter Count 4. 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: src/event_statistics.py

2.2 Transform data [28 points]

In this problem, we will convert the raw data to standardized format using PySpark. 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.py**: Complete this script based on provided skeleton.

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. Complete the method *calculate_index_dates* provided in *src/etl.py*
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. Complete the method $filter\_events$ provided in *src/etl.py*
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)
```

Complete the method *aggregate_events* provided in *src/etl.py*

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)
```

Complete the method *generate_feature_mapping* provided in *src/etl.py*

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). Complete the method *normalization* provided in *src/etl.py*

6. *Convert to SVMLight feature pair:* [4 points] 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:

```
[<feature>:<value> <feature>:<value> <feature>:<value>...]
<feature> .=. <integer> (generated event identifier)
<value> .=. <float> (3 places behind decimal)
```

Feature vector is a List. Each of the feature/value pairs is a String. Feature and value are connected by a colon, Feature/value pairs in MUST be ordered by increasing feature number. Features with value zero can be skipped. For example, the feature vector in SVMLight format will look like:

```
["2:0.500", "3:0.120", "10:0.900", "2000:0.300"]
["4:1.000", "78:0.600", "1009:0.200"]
["33:0.100", "34:0.980", "1000:0.800", "3300:0.200"]
```

Complete the method *svmlight_convert* provided in *src/etl.py*

7. *Create SVMLight samples:* [4 points] Concatenate the (mortality) label and generated sparse features of each patient to a string of SVMLight format (prepare to be saved in TXT files). Each string is a sample for machine learning model. Each sample is contains a target and a series of Feature/value pairs (no patient id) which is shown as below:

```
<line> .=. <target> <feature>:<value> <feature>:<value>
<target> .=. 1 | 0
```

the target value is 1 or 0 indicating whether the patient is dead or alive. The target value and each of the Feature/value pairs are separated by a space character. The strings (lines in TXT file) will look like:

```
1 2:0.500 3:0.120 10:0.900 2000:0.300
0 4:1.000 78:0.600 1009:0.200
1 33:0.100 34:0.980 1000:0.800 3300:0.200
```

Complete the method *svmlight_samples* provided in *src/etl.py*

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

```
python ./src/etl.py
```

Deliverable: src/etl.py

2.3 SGD Logistic Regression [25 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 ETL problem.

To better understand the performance of your classifier, you will need to use standard metrics like AUC. Our environment has provided necessary modules for drawing an ROC curve.

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 only, as we will not guarantee the availability of any third party modules while testing your code. [20 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]

Deliverable: lr/lrsgd.py

3 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. Please make sure your code can compile/run normally, otherwise you will get full penalty without comments. Organize your files same as the below framework, otherwise you will get 0/5 for Submission part.

```
<your gtid>-<your gt account>-hw2
|-- src
    |-- event_statistics.py
    |-- etl.py
    |-- lrsgd.py
|-- homework2_answer.pdf
```

Please create this folder manually, put all the files inside the folder, and create a zip archive using the following command and submit the tar file (keep in mind the folder '< *yourGTid* >-< *yourGTaccount* >-hw2' is required before creating the tar file) onto **Canvas** (PySpark is not well supported on Gradescope now). Example submission: 901234567-gburdell3-hw2.zip

- Wrong submission with wrong archive format or file name will lose up to 100 points!