CSE6250: Big Data Analytics in Healthcare
Homework 1

Jimeng Sun

Deadline: Sep 2, 2018 Anytime on Earth
(i.e., 8am Sep 3, 2018, EST)

- You can use 2 days of late submission throughout the semester **ONLY for homeworks**.
- 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. **We will apply anti-cheating software to check for plagiarism.** Any one who is flagged by the software will automatically receive 0 for the homework and be reported to the college.
- 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 you will receive no points in those failed tests. Built-in modules of python and the following libraries - pandas, numpy, scipy, scikit-learn can be used.
- 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 (not feasible given the class size). Thus **non-runnable code in our test environment will directly lead to 0 score** without comments.

Overview

Preparing the data, computing basic statistics and constructing simple models are essential steps for data science practice. In this homework, you will use clinical data as raw input to perform **Mortality Prediction**. For this homework, **Python** programming will be required. See the attached skeleton code as a start-point for the programming questions.

Also, you need to **make a single PDF file (*homework1_answer.pdf*)** of a compiled document for non-programming questions.

1 CITI Certification [10 points]

During this course, we will be working with the MIMIC database. MIMIC, although de-identified, still contains detailed information regarding the clinical care of patients, and must be treated with appropriate care and respect. In order to obtain access, it is necessary to finish the MIMIC CITI program training provided by MIT and get the certificate.

1. Navigate to the website: https://www.citiprogram.org/index.cfm?pageID=154&icat=0&ac=0. Under Register, select "Massachusetts Institute of Technology Affiliate" as your organization affiliation (not "independent learner") and create your account with your **GT email address (required)**.

2. Follow the links to add a Massachusetts Institute of Technology Affiliates course. In the Human Subjects training category, select the **"Data or Specimens Only Research"** course

3. Complete the course and save your completion report, which lists all modules completed with dates and scores.

Solution: Please include your certificates named $Certificate\_MIMIC.pdf$ as the submission format we provided at the end.

About Raw Data

Navigate to *homework1/data/train*. There are three CSV files which will be the input data in this homework.

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**: De-identified patient identiers. For example, the patient in the example above has patient id 1053.
- **event_id**: Clinical event identifiers. 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 conducted on the patient.
- **event_description**: Shows the description of the clinical event. For example, DIAG319049 is the code for Acute respiratory failure and DRUG19122121 is the code for Insulin.
- **timestamp**: the date at which the event happened. (Here the timestamp is not a real date.)
- **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.

The *event_feature_map.csv* is a map from an event_id (SNOMED, LOINC and RxNorm) to an integer index. This file contains *(idx, event_id)* pairs for all event ids.

Python and dependencies

In this homework, we will work on Python 3.6.5 environment. If you do not have a python distribution installed yet, we recommend installing Anaconda (or miniconda) with Python 3.6.5. We provide *homework1/environment.yml* which contains a list of libraries needed to set 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 3.6.5), please refer to this file to find necessary libraries, which is used to set the same coding/grading environment.

Alternatively, we have prepared the docker environment. You are highly encouraged to set the environment ready ASAP following the tutorials of this link, which you might use often in later homework. Please also activate *environment.yml* when needed in coding.

Running the tests

Test cases are provided for every module in this homework and operate on a subset of the data. To run a test, execute the following commands from the base folder i.e. homework1. 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_statistics.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

2 Descriptive Statistics [10 points]

Before starting analytic modeling, it is a good practice to get descriptive statistics of the input raw data. In this section, you need to write code that computes various metrics on the data described previously. A skeleton code is provided to you 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 hospital. All the events - DIAG, LAB and DRUG - should be considered as hospital visiting events.

- **Record length**: Duration (in number of days) between the first event and last event for a given patient.
  For example, if the first event is on 2014-05-21 and the last event is on 2014-05-24, the duration is 3 days. If a patient has only one event, the duration is 0.

a. Complete *src/event_statistics.py* to implement the required statistics.

Please be aware that **you are NOT allowed to change the filename and any existing function declarations.** Only numpy, scipy, scikit-learn and other built-in modules of python will be available for you to use. The use of *pandas* library is suggested.

b. Use *events.csv* and *mortality_events.csv* provided in **data/train** as input and fill Table 2 with actual values. Include this table in *homework1_answer.pdf*

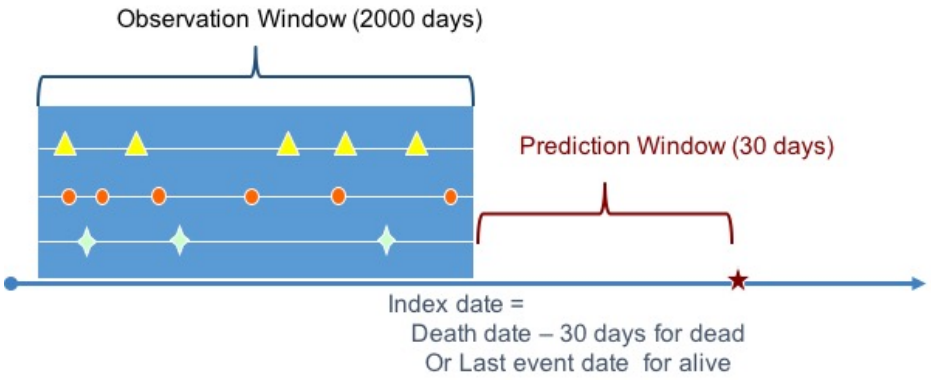
Metric	Deceased patients	Alive patients	Function to complete
Event Count 1. Average Event Count 2. Max Event Count 3. Min Event Count			event_count_metrics
Encounter Count 1. Average Encounter Count 2. Max Encounter Count 3. Min Encounter Count			encounter_count_metrics
Record Length 1. Average Record Length 2. Max Record Length 3. Min Record Length			record_length_metrics

Table 2: Descriptive statistics for alive and dead patients

Deliverable: src/event_statistics.py, homework1_answer.pdf [10 points]

3 Feature construction [30 points]

It is a common practice to convert raw data into a standard data format before running real machine learning models. In this section, you will work on *src/etl.py* file and implement the necessary python functions in this script. You will work with *events.csv*, *mortality_events.csv* and *event_feature_map.csv* files provided in **data/train** folder. The use of *pandas* library in this question is recommended. 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 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.
- Prediction Window: A fixed time interval that is to be used to make the prediction. Events in this interval should not be included in constructing feature vectors. The size of prediction window is 30 days.
- Index date: The day on which mortality is to be predicted. Index date is evaluated as follows:
  - For deceased patients: Index date is 30 days prior to the death date (timestamp field) in *data/train/mortality_events.csv*.
  - For alive patients: Index date is the last event date in *data/train/events.csv* for each alive patient.

Note: In the test_etl.py test, please **be careful** the format of **trailing space and line breaks** issue we provide in the *test_feature_order*. Adding a trailing blanks is not required in solution but adding it will not hurt grading. Line breaks are mandatory for grading.

Step - a. Compute the index date [8 points]

Use the definition provided above to compute the index date for all patients. Complete the method *calculate_index_date* provided in *src/etl.py* .

Deliverable: *src/etl.py, deliverables/etl_index_dates.csv*

Step - b. Filter events [5 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* .

Deliverable: *src/etl.py*, deliverables/etl_filtered_events.csv

c. Aggregate events [10 points]
To create features suitable for machine learning, we will need to aggregate the events for each patient as follows:

- **sum** values for diagnostics and medication events (i.e. event_id starting with DIAG and DRUG).
- **count** occurences for lab events (i.e. event_id starting with LAB).

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, 1.0)
(DIAG197320, 1.0)
(DRUG19122121, 2.0)
(LAB3026361, 4)
```

You will notice there are certain events with no entries in the values column. Handle these missing values by removing all events with null values while constructing the features. Next, replace each *event_id* with the $feature\_id$ provided in $data/train/event\_feature\_map.csv$

```
(708, 1.0)
(306, 1.0)
(2475, 2.0)
(3030, 3.35)
```

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). Complete the method *aggregate_events*

provided in *src/etl.py* .

Deliverable: *src/etl.py* and deliverables/etl_aggregated_events.csv

d. Save in SVMLight format [7 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.

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

The target value and each of the feature/value pairs are separated by a space character. Feature/value pairs 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:

```
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 alive or dead i.e. the label and it will be followed by a series of feature-value pairs sorted by the feature index (idx) value.

Deliverable: *src/etl.py*, deliverables/featured_svmlight.train and deliverables/features.train

4 Predictive Modeling [45 points]

4.1 Model Creation [15 points]

In the previous section, you constructed feature vectors for patients to be used as training data in various predictive models (classifiers). Now you will use this training data (deliverables/featured_svmlight.train) in 3 predictive models.

a. Implement Logistic Regression, SVM and Decision Tree. Skeleton code is provided in *src/models_partb.py, src/models_partc.py.*

b. Report performance metrics on the training data (deliverables/featured_svmlight.train). Skeleton code is provided in *src/models_partb.py.* You will evaluate and report the perfor-

mance of your predictive models based on the metrics listed in Table 3. Include this table in *homework1_answer.pdf*

Model	Accuracy	AUC	Precision	Recall	F-Score
Logistic Regression					
SVM					
Decision Tree					

Table 3: Model performance on training data

c. Evaluate your predictive models on a separate test dataset in *data/features_svmlight.validate* (binary labels are provided in that svmlight file as the first field). Skeleton code is provided in *src/models_partc.py*. You will report the performance of your predictive models based on the metrics listed in Table 4. Include this table in *homework1_answer.pdf*

Model	Accuracy	AUC	Precision	Recall	F-Score
Logistic Regression					
SVM					
Decision Tree					

Table 4: Model performance on test data

d. Based on the performance metrics on training and test data, please propose some strategies to improve the test performance and also provide the justification for your recommendation. For example, the strategies can be "gather more training data" or "do parameter tuning more on the algorithms".

Deliverable: src/models_partb.py, src/models_partc.py, homework1_answer.pdf [15 points]

4.2 Model Validation [10 points]

In order to fully utilize the available data and obtain more reliable results, machine learning practitioners use cross-validation to evaluate and improve their predictive models. You will demonstrate using two cross-validation strategies against Logistic Regression.

- K-fold: Divide all the data into k groups of samples. Each time $\frac{1}{k}$ samples will be used as test data and the remaining samples as training data.
- Randomized K-fold: Iteratively random shuffle the whole dataset and use top specific percentage of data as training and the rest as test.

a. Implement the two cross-validation strategies in *src/cross.py*.

1. **K-fold:** Use the number of iterations k=5;
2. **Randomized K-fold**: Use a test data percentage of 20% and k=5 for the number of iterations for Randomized

b. Report the average Accuracy and AUC in Table 5. Include this table in *homework1_answer.pdf*

CV strategy	Accuracy	AUC
K-Fold		
Randomized		

Table 5: Cross Validation

NOTE: You will use the features that you constructed in Section 3 as the entire dataset for this problem.

Deliverable: src/cross.py, homework1_answer.pdf [10 points]

4.3 Creating Your Best Model [15 points]

In this part, you will create your own best predictive model and set of features to attempt to obtain better performance compared to what you just worked on. You are advised to try out different things to improve the performance of your predictive model. One may try out new features, or use feature selection techniques to reduce the feature set, or tune the parameters of the predictive model or try ensemble techniques. However, one **must not** change the observation window and prediction window.

You should use the data available in *data/train* to construct features and train your predictive model. It is advisable to use cross validation and AUC as the metric to determine the relative performance of your model. Your final model will be evaluated on a (separate) test set for which the labels are unknown to you. The events in the observation window corresponding to the test patients are available in *data/test/events.csv*. If you are using the same features as in Section 3 for the test set, you may use the feature map in *data/test/event_feature_map.csv*.

a. Implement your predictive model in *src/my_model.py*. You are free to use your own features and predictive model. Please ensure that your predicted labels are either 0 or 1. Report your features of the test patients in *deliverables/test_features.txt*. Submit your predictions in the form of a csv file (patient_id, predicted label) in *deliverables/my_predictions.csv*

b. Write a short paragraph on your best predictive model (based on cross validation and AUC) and the other models that you tried. What was the rationale behind your approach? Did your model perform better than in the previous section? Include this in *homework1_answer.pdf*

Deliverable 1: deliverables/test_features.txt
(Refer to the skeleton code for the required format)
Deliverable 2: src/my_model.py
Deliverable 3: deliverables/my_predictions.csv

4.4 Kaggle [5 + up to 5 bonus points]

Kaggle is a platform for predictive modelling and analytics competitions. Submit your *deliverables/my_predictions.csv* to a kaggle competition created specifically for this assignment to compete with your fellow classmates. A **gatech.edu** email address is required to participate; follow the sign up directions using your university email address or if you already have an account, change the email on your existing account via your profile settings so you can participate. Make sure your display name (not necessarily your username) matches either your actual **full name or your GT account username** so your kaggle submission can be linked back to Canvas(Do NOT use nickname, otherwise we do not know whom takes this competition and your score won't count in this case).

Evaluation criteria is AUC. The predicted label is a soft label, which represents the possibility of mortality for each patient you predict. The label range is between 0 and 1. 50% of the data is used for the public leaderboard where you can receive feedback on your model. The final private leaderboard will use the remaining 50% of the data and will determine your final class ranking. More specific details can be found on the kaggle competition website.

Score at least 0.6 AUC to receive 5 points of credit. Additional bonus points can be received for your performance according to the following:

- Top 10%: 5 bonus points
- Top 15%: 4 bonus points
- Top 20%: 3 bonus points
- Top 25%: 2 bonus points
- Top 30%: 1 bonus point

Percentages are based on the entire class size, not just those who submit to kaggle.

5 Submission [5 points]

The folder structure of your submission should be as below. You can use the *tree* command to dispay and verify the folder structure is as seen below. **All modified src code should be in src folder. You should not change the given function names in source code. All other unrelated files will be discarded during testing and you will get ZERO score for Submission part. Make sure your codes com compile/run normally, otherwise you will get full penalty without comments**

```
<your gtid>-<your gt account>-hw1
|-- Certificate_MIMIC.pdf
|-- homework1_answer.pdf
|-- src
   |-- event_statistics.py
   |-- etl.py
   |-- models_partb.py
   |-- models_partc.py
   |-- cross.py
   |-- utils.py
   |-- my_model.py
|-- deliverables
   |-- etl_index_dates.csv
   |-- etl_filtered_events.csv
   |-- etl_aggregated_events.csv
   |-- features_svmlight.train
   |-- features.train
   |-- test_features.txt
   |-- my_predictions.csv
```

Create a tar archive of the folder above with the following command and submit the tar file (should have one folder named '< *yourGTid* >-< *yourGTaccount* >-hw1' inside the .tar.gz file).

```
tar -czvf <your GTid>-<your GT account>-hw1.tar.gz \
  <your GTid>-<your GT account>-hw1
```

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

Common Errors which are not accepted:

- Underscore: 901234567_gburdell3_hw1.tar.gz
- Zip file: zip your files and rename it