CSE6250: Big Data Analytics in Healthcare
Homework 5

Jimeng Sun

Deadline: Mar 25, 2019 Anytime on Earth
(i.e., 8am Mar 26, 2019, 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

Neural Networks and Deep Learning are becoming more popular and widely applied to many data science domain including healthcare applications. In this homework, you will implement various types of neural network with clinical data. For this homework, **Python** programming

will be required. See the attached skeleton code as a starting point for the programming questions. Also, you need to **make a single PDF file (*homework5_answer.pdf*)** of a compiled document for non-programming questions. You can use the attached LaTeXtemplate.

It is highly recommended to complete Deep Learning Lab first if you have not finished it yet. (PyTorch version could be different, but most of parts are same.)

Python and dependencies

In this homework, we will work with **PyTorch** 1.0 on Python 3.6 environment. If you do not have a python distribution installed yet, we recommend installing Anaconda (or miniconda) with Python 3.6. We provide *homework5/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'. Refer to the documantation for more details.

```
conda env create -f environment.yml
```

If you already have your own Python development environment (it should be Python 3.6), please refer to this file to find necessary libraries, which is used to set the same coding/grading environment.

Content Overview

```
homework5
|-- code
|   |-- etl_mortality_data.py
|   |-- mydatasets.py
|   |-- mymodels.py
|   |-- plots.py
|   |-- tests
|   |   |-- test_all.py
|   |-- train_seizure.py
|   |-- train_variable_rnn.py
|   |-- utils.py
|-- data
|   |-- mortality
|   |   |-- test
|   |   |   |-- ADMISSIONS.csv
|   |   |   |-- DIAGNOSES_ICD.csv
|   |   |   |-- MORTALITY.csv
|   |   |-- train
|   |   |   |-- ADMISSIONS.csv
```

```
|   |   |   |-- DIAGNOSES_ICD.csv
|   |   |   |-- MORTALITY.csv
|   |   |-- validation
|   |   |   |-- ADMISSIONS.csv
|   |   |   |-- DIAGNOSES_ICD.csv
|   |   |   |-- MORTALITY.csv
|   |-- seizure
|   |   |-- seizure_test.csv
|   |   |-- seizure_train.csv
|   |   |-- seizure_validation.csv
|-- environment.yml
|-- homework5.pdf
|-- homework5_answer.tex
```

Implementation Notes

Throughout this homework, we will use *Cross Entropy* loss function, which has already been applied in the code template. **Please note that this loss function takes activation logits as its input, not probabilities. Therefore, your models should not have a softmax layer at the end.** Also, you may need to control the training parameters declared near the top of each main script, e.g., the number of epochs and the batch size, by yourself to achieve good model performances.

You will submit a trained model for each type of neural network, which will be saved by the code template, and they will be evaluated on a hidden test set in order to verify that you could train the models. Please make sure that your saved/submitted model files match your model class definitions in your source code. **You will get 0 score for each part in either case you do not submit your saved model files or we cannot use your model files directly with your code.**

Unit tests

Some unit tests are provided in *code/tests/test_all.py*. You can run all tests, **in your `code` directory**, simply by

```
python -m pytest
```

NOTE: the primary purpose of these unit tests is just to verify the return data types and structures to match them with the grading system.

1 Epileptic Seizure Classification [50 points]

For the first part of this homework, we will use Epileptic Seizure Recognition Data Set which is originally from UCI Machine Learning Repository. You can see the three CSV files under the folder *homework5/data/seizure*. Each file contains a header at the first line, and each line represent a single EEG record with its label at the last column (Listing 1). Refer to the link for more details of this dataset.

```
X1,X2,...,X178,y
-104,-96,...,-73,5
-58,-16,...,123,4
...
```

Listing 1: Example of seizure data

Please start at *train_seizure.py*, which is a main script for this part. If you have prepared the required Python environment, you can run all procedures simply by

```
python train_seizure.py
```

1.1 Data Loading [5 points]

First of all, we need to load the dataset to train our models. Basically, you will load the raw dataset files, and convert them into a PyTorch TensorDataset which contains data tensor and target (label) tensor. Since each model requires a different shape of input data, you should convert the raw data accordingly. Please look at the code template, and you can complete each type of conversion at each stage as you progress.

a. Complete `load_seizure_dataset` in `mydatasets.py` [5 points]

1.2 Multi-layer Perceptron [15 points]

We start with a simple form of neural network model first, Multi-layer Perceptron (MLP).

a. Complete a class `MyMLP` in `mymodels.py`, and implement 3-layer MLP similar to the Figure 1. Use a hidden layer composed by 16 hidden units, followed by a sigmoid activation function. [3 points]

b. Calculate the number of "trainable" parameters in the model with providing the calculation details. How many floating-point computation will occur **APPROXIMATELY** when a new single data point comes in to the model? **You can make your own assumptions on the number of computations made by each elementary arithmetic, e.g., add/subtraction/multiplication/division/negation/exponent take 1 operation, etc.** [5 points]

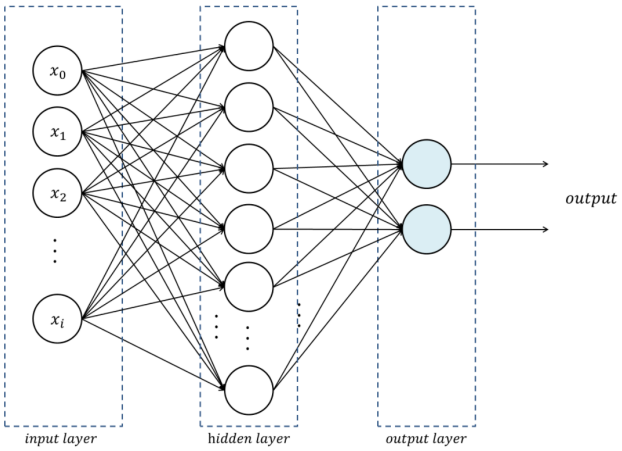
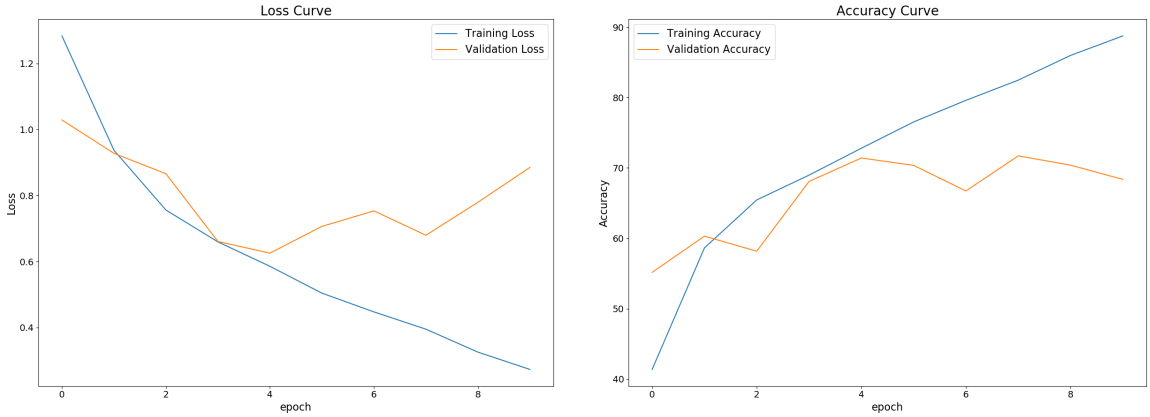


Figure 1: An example of 3-layer MLP



(a) Loss curves

(b) Accuracy curves

Figure 2: Learning curves

It is important to monitor whether the model is being trained well or not. For this purpose, we usually track how some metric values, loss for example, change over the training iterations.

c. Complete a function `plot_learning_curves` in `plots.py` to plot loss curves and accuracy curves for training and validation sets as shown in Figure 2a. You should control the number of epochs (in the main script) according to your model training. Attach the plots for your MLP model in your report. [2 points]

After model training is done, we also need to evaluate the model performance on an unseen (during the training procedure) test data set with some performance metrics. We will use *Confusion Matrix* among many possible choices.

d. Complete a function `plot_confusion_matrix` in `plots.py` to make and plot a confusion matrix for test set similar to Figure 3. Attach the plot for your MLP model in your report. [2 points]

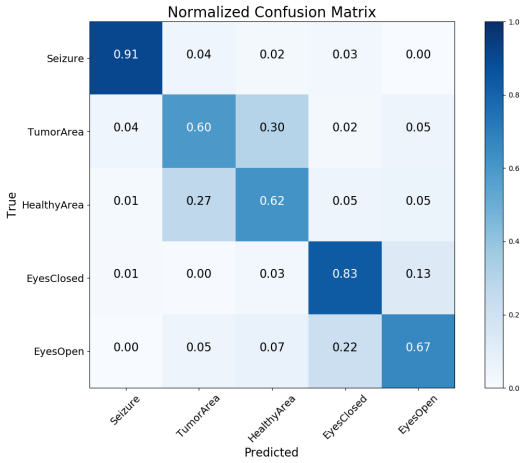


Figure 3: An example of confusion matrix

You have implemented a very simple MLP model. Try to improve it by changing model parameters, e.g., a number of layers and units, or applying any trick and technique applicable to neural networks.

e. Modify your model class `MyMLP` in `mymodels.py` to improve the performance. It still must be a MLP type of architecture. Explain your architecture and techniques used. Briefly discuss about the result with plots. [3 points]

1.3 Convolutional Neural Network (CNN) [15 points]

Our next model is CNN. Implement a CNN model constituted by a couple of convolutional layers, pooling layer, and fully-connected layers at the end as shown in Figure 4.

a. Complete `MyCNN` class in `mymodels.py`. Use two convolutional layers, one with 6 filters of the kernel size 5 (stride 1) and the other one with 16 filters with the kernel size 5 (stride 1), and they must be followed by Rectified Linear Unit (ReLU) activation. Each convolutional layer (after ReLU activation) is followed by a max pooling layer with the size (as well as stride) of 2. There are two fully-connected (aka dense) layer, one with 128 hidden units followed by ReLU activation and the other one is the output layer that has five units. [5 points]

b. Calculate the number of "trainable" parameters in the model with providing the calculation details. How many floating-point computation will occur **APPROXIMATELY** when a new single data point comes in to the model? **You can make your own assumptions on the number of computations made by each elementary arithmetic, e.g., add/subtraction/multiplication/division/negation/exponent take 1 operation, etc.** [5 points]

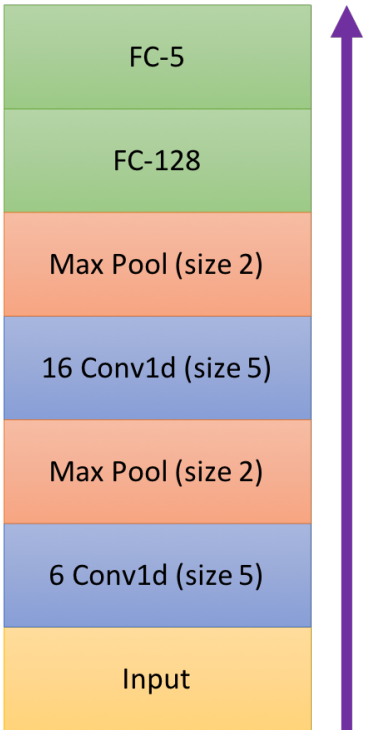


Figure 4: CNN architecture to be implemented. k Conv1d (size d) means there are k numbers of Conv1d filters with a kernel size of d. Each convolution and fully-connected layer is followed by ReLU activation function except the last layer.

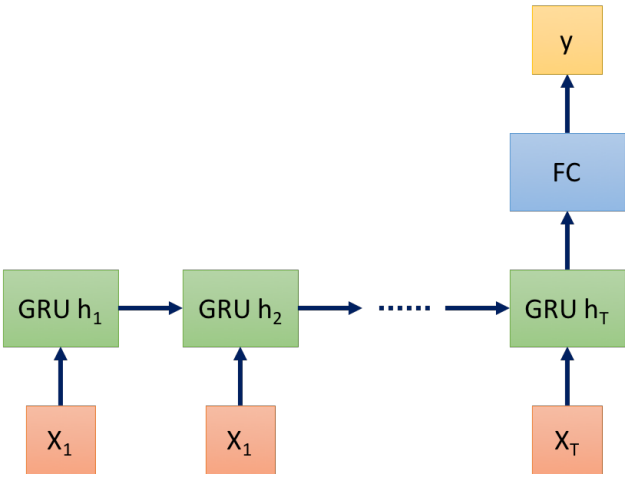


Figure 5: A many-to-one type of RNN architecture to be implemented.

c. Plot and attach the learning curves and the confusion matrix for your CNN model in your report. [2 points]

Once you complete the basic CNN model, try to improve it by changing model parameters and/or applying tricks and techniques.

d. Modify your model class `MyCNN` in `mymodels.py` to improve the performance. It still must be a CNN type of architecture. Explain your architecture and techniques used. Briefly discuss about the result with plots. [3 points]

1.4 Recurrent Neural Network (RNN) [15 points]

The final model you will implement on this dataset is RNN. For an initial architecture, you will use a Gated Recurrent Unit (GRU) followed by a fully connected layer. Since we have a sequence of inputs with a single label, we will use a many-to-one type of architecture as it is shown in Figure 5.

a. Complete `MyRNN` class in `mymodels.py`. Use one GRU layer with 16 hidden units. There should be one fully-connected layer connecting the hidden units of GRU to the output units. [5 points]

b. Calculate the number of "trainable" parameters in the model with providing the calculation details. How many floating-point computation will occur **APPROXIMATELY** when a new single data point comes in to the model? **You can make your own assumptions on the number of computations made by each elementary arithmetic, e.g., add/subtraction/multiplication/division/negation/exponent take 1 operation, etc.** [5 points]

c. Plot and attach the learning curves and the confusion matrix for your RNN model in your report. [2 points]

d. Modify your model class `MyCNN` in `mymodels.py` to improve the performance. It still must be a RNN type of architecture. Explain your architecture and techniques used. Briefly discuss about the result with plots. [3 points]

2 Mortality Prediction with RNN [45 points + 5 bonus]

In the previous problem, the dataset consists of the same length sequences. In many real-world problems, however, data often contains variable-length of sequences, natural language processing for example. Also in healthcare problems, especially for longitudinal health records, each patient has a different length of clinical records history. In this problem, we will apply a recurrent neural network on variable-length of sequences from longitudinal electronic health record (EHR) data. Dataset for this problem was extracted from MIMIC-III dataset. You can see the three sub-folders under the folder *homework5/data/mortality*, and each folder contains the following three CSV files.

- **MORTALITY.csv**: A pre-processed label file with the following format with a header. SUBJECT_ID represents a unique patient ID, and MORTALITY is the target label for the patient. **For the test set, all labels are -1, and you will submit your predictions on Kaggle competition at the end of this homework.**

```
SUBJECT_ID,MORTALITY
123,1
456,0
...
```

- **ADMISSIONS.csv**: A filtered MIMIC-III ADMISSION table file. It has patient visits (admissions) information with unique visit IDs (HADM_ID) and dates (ADMITTIME)

- **DIAGNOSES_ICD.csv**: A filtered MIMIC-III DIAGNOSES_ICD table file. It contains diagnosis information as ICD-9 code given to patients at the visits.

Please start at *train_variable_rnn.py*, which is a main script for this part. You can run all procedures simply by

```
python train_variable_rnn.py
```

once you complete the required parts below.

Again, you will submit a trained model, and it will be evaluated on a hidden test set in order to verify that you could train the models.

2.1 Preprocessing [10 points]

You may already have experienced that preprocessing the dataset is one of the most important part of data science before you apply any kind of machine learning algorithm. Here, we will implement a pipeline that process the raw dataset to transform it to a structure that we can use with RNN model. You can use typical Python packages such as Pandas and Numpy.

Simplifying the problem, we will use the main digits, which are the first 3 or 4 alphanumeric digits prior to the decimal point, of ICD-9 codes as features in this homework.

The pipeline procedure is very similar to what you have done so far in the past homework, and it can be summarized as follows.

1. Loading the raw data files.
2. Group the diagnosis codes given to the same patient on the same date.
3. Sorting the grouped diagnoses for the same patient, in chronological order.
4. Extracting the main code from each ICD-9 diagnosis code, and converting them into unique feature ids, 0 to $d-1$, where d is the number of unique main -digits codes.
5. Converting into the final format we want. Here we will use a List of (patient) List of (code) List as our final format of the dataset.

The order of some steps can be swapped for implementation convenience. Please look at the `main` function in `etl_mortality_data.py` first, and do the following tasks.

a. Complete `convert_icd9` function in `etl_mortality_data.py`. Specifically, extract the the first 3 or 4 alphanumeric characters prior to the decimal point from a given ICD-9 code. Please refer to the ICD-9-CM format for the details, and note that there is no actual decimal point in the representation by MIMIC-III. [2 points]

b. Complete `build_codemap` function in `etl_mortality_data.py`. Basically, it is very similar to what you did in Homework 1 and is just constructing a unique feature ID map from the main-digits of the ICD-9 codes. [1 point]

c. Complete `create_dataset` function in `etl_mortality_data.py` referring to the summarized steps above and the TODO comments in the code. [7 points]

For example, if there are two patients with the visit information below (in an abstracted format),

```
SUBJECT_ID,ADMITTIME,ICD9_CODE
1, 2018-01-01, 123
1, 2018-01-01, 234
2, 2013-04-02, 456
```

```
2, 2013-04-02, 123
3, 2018-02-03, 234
2, 2013-03-02, 345
2, 2013-05-03, 678
2, 2013-05-03, 234
3, 2018-09-07, 987
```

the final visit sequence dataset should be

```
[[[0, 1]], [[3], [0, 2], [4, 1]], [[1], [5]]]
```

with the order of [Patient 1, Patient 2, Patient 3] and the code map {123: 0, 234: 1, 456: 2, 345: 3, 678: 4, 987: 5}.

Once you complete this pre-processing part, you should run it to prepare data files required in the next parts.

```
python etl_mortality_data.py
```

2.2 Creating Custom Dataset [15 points]

Next, we will convert the pre-processed data in the previous step into a custom type of PyTorch Dataset. When you create a custom (inherited) PyTorch Dataset, you should override at least `__len__` and `__getitem__` methods. `__len__` method just returns the size of dataset, and `__getitem__` actually returns a sample from the dataset according to the given index. Both methods have already been implemented for you. Instead, you have to complete the constructor of `VisitSequenceWithLabelDataset` class to make it a valid 'Dataset'. We will store the labels as a List of integer labels, and the sequence data as a List of matrix whose i-th row represents i-th visit while j-th column corresponds to the integer feature ID j. For example, the visit sequences we obtained from the previous example will be transformed to matrices as follows.

$$\text{P1} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \ \text{P2} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}, \ \text{P3} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

a. Complete `calculate_num_features` function and `VisitSequenceWithLabelDataset` class in `mydatasets.py`. [5 points]

PyTorch DataLoader will generate mini-batches for you once you give it a valid Dataset, and most times you do not need to worry about how it generates each mini-batch if your data have fixed size. In this problem, however, we use variable size (length) of data, e.g.,

visits of each patient are represented by a matrix with a different size as in the example above. Therefore, we have to specify how each mini-batch should be generated by defining `collate_fn`, which is an argument of DataLoader constructor. Here, we will create a custom collate function named `visit_collate_fn`, which creates a mini-batch represented by a 3D Tensor that consists of matrices with the same number of rows (visits) by padding zero-rows at the end of matrices shorter than the largest matrix in the mini-batch. Also, the order of matrices in the Tensor must be sorted by the length of visits in descending order. In addition, Tensors contains the lengths and the labels are also have to be sorted accordingly.

Please refer to the following example of a tensor constructed by using matrices from the previous example.

$$T[0,:,:] = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$T[1,:,:] = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T[2,:,:] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where T is a Tensor contains a mini-batch size of batch_size $\times$ max_length $\times$ num_features, and batch_size=3, max_length=3, and num_features=6 for this example.

b. Complete `visit_collate_fn` function in `mydatasets.py`. [10 points]

2.3 Building Model [15 points]

Now, we can define and train our model. Figure 6 shows the architecture you have to implement first, then you can try your own model.

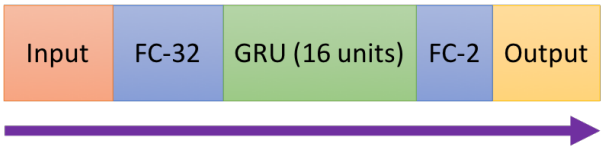


Figure 6: A RNN architecture to be implemented with variable-length of sequences.

a. Complete `MyVariableRNN` class in `mymodels.py`. First layer is a fully-connected layer

with 32 hidden units, which acts like an embedding layer to project sparse high-dimensional inputs to dense low-dimensional space. The projected inputs are passed through a GRU layer that consists of 16 units, which learns temporal relationship. It is followed by output (fully-connected) layer constituted by two output units. In fact, we can use a single output unit with a sigmoid activation function since it is a binary classification problem. However, we use a multi-class classification setting with the two output units to make it consistent with the previous problem for reusing the utility functions we have used. Use `tanh` activation funtion after the first FC layer, and remind that you should not use `softmax` or `sigmoid` activation after the second FC (output) layer since the loss function will take care of it. Train the constructed model and evaluate it. Include all learning curves and the confusion matrix in the report. [10 points]

b. Modify your model class `MyVariableRNN` in `mymodels.py` to improve the performance. It still must be a RNN type that supports variable-length of sequences. Explain your architecture and techniques used. Briefly discuss about the result with plots. [5 points]

2.4 Kaggle Competition [5 points + 5 bonus]

NOTE: NO late submission will be accepted for this Kaggle competition.

You should be familiar with Kaggle, which is a platform for predictive modeling and analytics competitions, that you used in the past homework. You will compete with your class mates using the result by your own model that you created in Q2.3.b.

Throughout this homework, we did not use `sigmoid` activation or `softmax` layer at the end of the model output since CrossEntropy loss applies it by itself. For Kaggle submission, however, we need to have a soft-label, probability of mortality, for each patient. For this purpose, you have to complete a function `predict_mortality` function in the script to evaluate the test set and collect the probabilities.

a. Complete `predict_mortality` function in `train_variable_rnn.py`, submit the generated predictions (output/mortality/my_predictions.csv) to the Kaggle competition, and achieve AUC > 0.6. [5 points]

NOTE: it seems Kaggle accepts 20 submissions at maximum per day for each user.

Submit your **soft-labeled** CSV prediction file (MORTALITY is in the range of [0,1]) you generate to this Kaggle competition[1] (**you can join the competition via this link only**) created specifically for this assignment to compete with your fellow classmates. A **gatech.edu** email address is required to participate; follow the sign up direction using your GT email address, or if you already have an account, change the email on your existing account via your profile settings so that you can participate. **Make sure your display**

[1] https://www.kaggle.com/t/752ea84a6fb846f6a8d081a15502a108

name (not necessarily your username) matches your GT account username, e.g., san37, so that your Kaggle submission can be linked to the grading system.

Evaluation criteria is AUC, and the predicted label should be 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.

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.

3 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 code files should be in 'code' folder. You should not change the given function names in the source code. All other unrelated files will be discarded during testing and you will get ZERO score for Submission part. Make sure your codes can be compiled/run normally, otherwise you will get full penalty without comments. Submissions with a different structure of directories, files, or tar file name will be excluded from grading with 0 score.**

```
<your gtid>-<your gt account>-hw5
|-- homework5_answer.pdf
|-- code
   |-- etl_mortality_data.py
   |-- mydatasets.py
   |-- mymodels.py
   |-- plots.py
   |-- train_seizure.py
   |-- train_variable_rnn.py
```

```
   |-- utils.py
|-- output
   |-- mortality
      |-- MyVariableRNN.pth
   |-- seizure
      |-- MyCNN.pth
      |-- MyMLP.pth
      |-- MyRNN.pth
```

Create a tar archive of the folder above with the following command and submit the tar file (should have one folder named '< *yourGTid* >-< *yourGTaccount* >-hw5' inside the .tar.gz file).

```
tar -czvf <your GTid>-<your GT account>-hw5.tar.gz \
  <your GTid>-<your GT account>-hw5
```

Example submission: 901234567-gburdell3-hw5.tar.gz

Common Errors which are not accepted:

- Underscore: 901234567_gburdell3_hw5.tar.gz
- Zip file: zip your files and rename it