Reproduction of paper 228 for CS598 DLH4H in Spring 2022

Daoxing Wang, Zuliang Weng
{wang264, zwe}@illinois.edu

Group ID: 37
Paper ID: 228, Difficulty: easy
Presentation link: https://youtu.be/h9lZkxve6wU
Code link: https://github.com/jeffzlweng/DII-Challenge

1 Introduction

Sepsis is a life-threatening condition, and its high cost adds a burden to the healthcare system. Studies show that early prediction can significantly improve the survival in sepsis patients. There have been extensive research on such topic. However, most models were built with limited success. Traditional models and approaches suffer in two ways: they predict too late, and they are hard to interpret. Most models were also built in ICU settings rather than in the more critical Emergency Department. The authors' goal is to predict sepsis 4 hours before onset for hospitalized adult patients.

There are two use cases with two different goals. In case 1, all the observations are sampled from septic patients, and the goal is to find whether a model can tell if a patient is likely to have high sepsis risk a few hours before the onset. In case 2, observations are from patients who have sepsis onset in the next four hours, as well as those who do not have sepsis at all; the goal is to predict whether sepsis occurs in the following 4 hours.

The original authors proposed a Long Short-Term Memory (LSTM) based model, which outperforms the traditional models and gives the model clinical interpretability (with attention mechanism and global max pooling).

2 Scope of reproducibility

This research paper claims that on the private dataset provided, the proposed LSTM model achieved an average AUC of 0.892, the mean of 0.940 and 0.845 from the two use cases discussed in the introduction. Following the paper, we managed to clean the data set for both use cases and successfully reproduce the result for both case 1 and case 2. The results will be discussed in the section-4.

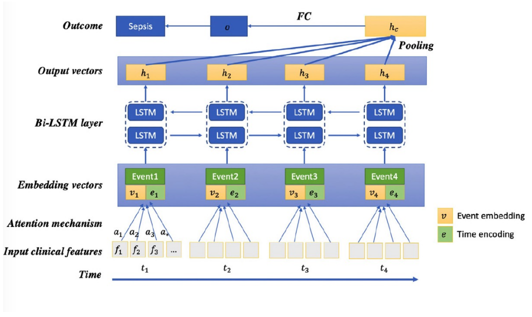

Figure 1: Model Architecture

3 Methodology

This research paper comes with a GitHub repository, and our work uses that as a template. We start by forking from that repository and checking all the changes to our repository, including changes needed to run the code on our hardware. Also, the libraries used may be outdated and need upgrading. Data sets used in building the models are also part of the repo, so we do not need to acquire that from MMIC website. Both members of the team have both Windows machines and Mac, but none of the machines has an Nvidia graphic card, so GPU was not used in the training process.

3.1 Model descriptions

The architecture of the proposed LSTM-based model is presented in figure-1. The first component is the the Bidirectional LSTM which used the concatenation of the medial event embedding vector v and time encoding e as input. Then global max pooling is performed on the output vector h to produce the patient representation vector. The last step is a fully connected layer to predict the probability of sepsis onset in the next 4 hours. There are a total of X parameters for this model.

Hyperparameter Name	Value
Learning rate	0.0001
Embed vocab size	2031
Embed size	256
Hidden size	256
Number of layers	2
Dropout rate	0.25

Table 1: Hyperparameters for LSTM based model

3.2 Data descriptions

The dataset for this research is a private dataset downloaded from the original paper's GitHub repository. This dataset is separate in multi CSV files, and data cleaning is needed before the data can be used for model training.

This dataset contains the EHR (Electronic Health Records) for 100 patients. Among those patients, some patients experience sepsis while some patients do not. Vitals of related medical testing and procedures are provided at the time of such events. The vital field name in string is mapped to an integer, and related vitals are further grouped together. Moreover, each feature is standardized by using the min-max scaling technique before training. The response variable of this research is whether patients are not experiencing sepsis in the next four hours (binary label of 0 and 1).

3.3 Hyperparameters

The paper does not report the hyperparameters for the proposed LSTM-based model. In our research, we adapted the default value of the hyperparameters in the provided code. It is worth mentioning that hyperparameter searching logic is not provided in the original paper, and only grid-search logic was provided. Interestingly, the grid-search consists of a for loop through pre-defined seeds, which have no meaningful effect. Without hyperparameter tuning, the models are trained in training data and tested on the testing dataset. Table1 present the key hyperparameters being used in our model.

3.4 Implementation

The implementation details, including data processing, model building, analytics, and code provided by the authors, reside in our GitHub repository. The link is the following: `https://github.com/jeffzlweng/DII-Challenge`.

In our implementation, the contribution to the original research is fivefold and is listed below:

- Convert to Python 3 because Pytorch does not support Python 2.7 in the Windows system.
- Add key checking logic to handle edge cases in the data generation pipeline.
- Correct some documentation errors.
- Fine-tune grid-search logic to include more hyperparameters.
- Add missing code for running validation only and test only

3.5 Computational requirements

The LSTM-based model takes about 10 minutes to train on the private dataset for case 1 on a PC with the following spec:

- Processor Intel(R) Core(TM) i78700 CPU @ 3.20GHz, 3192 Mhz, 6 Core(s), 12 Logical Processor(s)
- Installed Physical Memory (RAM) 32.0 GB
- No GPU is used

The total number of validating epochs is 20, and it takes about 30 seconds on average for each epoch. On the same machine, it takes about 13 minutes to run case 2. The total number of validating epochs is also 20, and each validating epoch takes 39 seconds. It is worth noting that we found that the code will not run on an M1 Mac due to a wrong instruction exception. This is probably due to M1's lack of support for some of the Python dependency libraries needed. Moreover, there is no Pytorch version that supports Python 2 in Windows.

4 Results

In our research aims to reproduce similar AUCs as the authors claimed. There are two use cases in the original paper. For case 1, the original paper claimed to have an AUC of 0.94, while for case 2, the original paper claimed to have an AUC of 0.84.

4.1 Result 1

In Table-2 we list all the testing results from all 20 epochs, and the best result among them is epoch 12. This epoch has an AUC score of 0.988, F1 Score of 0.941, Accuracy of 0.944, Recall of 1, and Precision equal to 0.888. This is even higher than what the authors claimed, and we believe this result is too good to be true.

Epoch	**AUC**	**F1 Score**	**Accuracy**	**Recall**	**Precision**
epoch1	0.850	0.667	0.611	0.875	0.538
epoch2	0.762	0.636	0.611	0.750	0.545
epoch3	0.962	0.823	0.833	0.875	0.778
epoch4	0.937	0.736	0.722	0.875	0.636
epoch5	0.837	0.778	0.778	0.875	0.700
epoch6	0.712	0.631	0.611	0.750	0.545
epoch7	0.825	0.736	0.722	0.875	0.636
epoch8	0.662	0.631	0.611	0.750	0.545
epoch9	0.912	0.736	0.722	0.875	0.6363
epoch10	0.925	0.727	0.667	1.000	0.571
epoch11	0.925	0.761	0.722	1.000	0.615
epoch12	0.987	0.941	0.944	1.000	0.8888
epoch13	0.900	0.842	0.833	1.000	0.727
epoch14	0.837	0.778	0.778	0.875	0.700
epoch15	0.887	0.875	0.889	0.875	0.875
epoch16	0.962	0.842	0.833	1.000	0.727
epoch17	0.850	0.636	0.556	0.875	0.500
epoch18	0.925	0.800	0.833	0.750	0.857
epoch19	0.862	0.857	0.889	0.750	1.000

Table 2: Numerical results for case 1

Epoch	**AUC**	**F1 Score**	**Accuracy**	**Recall**	**Precision**
epoch1	0.813	0.620	0.479	0.960	0.466
epoch2	0.447	0.680	0.570	1.000	0.515
epoch3	0.799	0.627	0.457	1.000	0.457
epoch4	0.728	0.664	0.556	0.960	0.507
epoch5	0.921	0.635	0.475	1.000	0.465
epoch6	0.873	0.630	0.470	0.990	0.460
epoch7	0.933	0.710	0.638	0.970	0.560
epoch8	0.893	0.670	0.561	0.980	0.510
epoch9	0.932	0.758	0.714	0.980	0.618
epoch10	0.884	0.750	0.710	0.950	0.619
epoch11	0.931	0.758	0.723	0.950	0.631
epoch12	0.893	0.752	0.714	0.950	0.623
epoch13	0.892	0.775	0.769	0.871	0.698
epoch14	0.897	0.769	0.769	0.841	0.708
epoch15	0.880	0.790	0.796	0.841	0.745
epoch16	0.757	0.690	0.7692	0.564	0.890
epoch17	0.791	0.616	0.678	0.564	0.678
epoch18	0.860	0.768	0.733	0.970	0.636
epoch19	0.729	0.581	0.628	0.564	0.600

Table 3: Numerical results for case 2

4.2 Result 2

In Table-3 we list the results from all 20 epochs for case 2, and the best among them is epoch 11. This epoch report an AUC score of 0.931, F1 score of 0.758, Accuracy of 0.723, Recall of 0.950, and Precision of 0.631. Compared with the original result, this is also higher than the authors' claimed.

4.3 Analysis

For result one presented in 4.1, the AUC score is very close to the authors' result. One thing particularly interesting is that we have a recall rate equal to 1, which means that we successfully select out all patients that will have sepsis conditions. In this case, false positive is more acceptable than false negative as the latter post a significant risk to patients and could delay patients' treatment.

For result two presented in 4.2, we also got outstanding results. However, the biggest concern is that different runs returned very different results.

4.4 Future Plan

There are a few concerns we have regarding the reproduction of this paper.

- The result for case 2 is better than the authors claimed. This is uncommon, so we want to confirm that.
- Fine-tune grid-search logic to include more hyperparameters.
- The ablation study for case 1.
- Test the model with more data sets.
- If time permits, use GAN or generative models to generate more data for testing.
- Present our code and findings via a YouTube video.

5 Discussion

After a few revisions, we were able to run the source code provided by the authors against the data attached in the codebase. We got comparable results compared to the results in the original paper. However, the concern is that the dataset used for training is surprisingly small. master.csv and label.csv only contain 100 data points, and yet it was able to reproduce excellent results.

5.1 What was easy

- The original codebase is hosted in Github, making it quite easy to fork and collaborate between team members.
- It's relatively easy to run authors' code. After some revisions, we could get it running in Windows using Python3.
- Data is all attached to the code base, so we did not have to spend time getting the data.
- Authors took great care in printing out necessary info to monitor the model building progress.
- Documentation is adequate to get things started.

5.2 What was difficult

- To fully understand the details of the code. It's easy to get code to run, but whenever we try to debug some issues, the logic of the code is hard to follow. We think there were bugs in some cases, but we were not entirely sure.
- The code for running validation only or test only is missing. It is not too hard to add it back, but it took us some time to figure out why it did not run.
- To make sense of why such good results can be attained from such a small data set.

5.3 Recommendations for reproducibility

- Fix errors in documentation.
- Add comments on code blocks to improve readability.
- Upgrade to Python 3, which allows Windows users to reproduce this paper.

6 Communication with original authors

To get a better understating of the work. We send an email to the original author regarding a couple of questions during our reproduction of the original work. The questions include some broken code and the tiny sample in the private dataset. No response has been received yet from the original authors.

7 Reference of the original paper

Zhang, D., Yin, C., Hunold, K.M., Jiang, X., Caterino, J.M. and Zhang, P., 2021. An interpretable deep-learning model for early prediction of sepsis in the emergency department. Patterns, 2(2), p.100196.