Deep Learning for Sepsis Mortality Prediction

Melissa Lynn
mklynn2@illinois.edu

Group ID: 105, Paper ID: Paper is not from list
Presentation link: https://youtu.be/fCAFR944Jro
Code link: https://github.com/lynn0032/CS598_final_project

1 Introduction

In this project, I attempt to reproduce the results in (Perng et al., 2019) for predicting mortality in septic patients. In this work, Perng et al. use an autoencoder and a convolutional neural network for feature construction for other machine learning models. The work is interesting in its attempt to apply deep learning models to improve the ability to predict mortality in sepsis patients.

In my attempt to reproduce this work, I try to apply the same algorithms to a similar dataset of sepsis patients, obtained from MIMIC III database by (Hou et al., 2020). My goal was to evaluate the performance of the models used in (Hou et al., 2020) and (Perng et al., 2019), and confirm the superior performance of the deep learning models proposed by (Perng et al., 2019). However, in my attempt to reproduce the results of (Perng et al., 2019), I found serious issues in this paper that call the results into question. [1]

[1] My original intent was to reproduce the results of paper number 286, (Chu et al., 2020). This work uses electronic health records to predict the endpoint of heart failure. The "endpoint" the work is predicting is the eventual outcome, e.g. readmission for heart failure or mortality. Chu et al. use a recurrent neural network with adversarial learning and multi-task learning used as additional steps. This approach is interesting in how it combines ideas, and improves on the performance of RETAIN (and other models).

To demonstrate the reproducibility of this work, I intended to apply the same network to a different data set with a different task, predicting sepsis mortality. The review (Deng et al., 2022) provides a survey of results on sepsis prediction, including mortality predictions. My goal was to demonstrate that the approach proposed by Chu et al. can have similar performance improvements for prediction of sepsis mortality, as it did for heart failure endpoint prediction.

While verifying the performance of baseline models for predicting mortality in sepsis patients, I discovered significant issues in the paper (Perng et al., 2019), and I decided to focus my project on that paper instead.

Model	AUC	Accuracy (%)
RF	0.89	62.56
KNN	0.84	77.31
SVM	0.90	74.33
Softmax	0.88	82.73
PCA + RF	0.89	62.62
PCA + KNN	0.84	81.67
PCA + SVM	0.89	78.91
PCA + SoftMax	0.91	83.48
AE + RF	0.84	63.52
AE + KNN	0.81	80.64
AE + SVM	0.89	78.76
AE + SoftMax	0.90	84.17
CNN + RF	0.90	61.03
CNN + KNN	0.86	81.73
CNN + SVM	0.92	84.96
CNN + SoftMax	0.92	87.01

Figure 1: AUC values of models from (Perng et al., 2019), Table 2, for predicting 28-day mortality of sepsis patients

2 Scope of reproducibility

The review (Deng et al., 2022) gives an overview of various studies applying machine learning to prediction tasks for sepsis patients, including models that predict mortality. The review identified (Perng et al., 2019) as having the best performance for mortality prediction, where a convolutional neural network with softmax was used to predict mortality from sepsis. The performance results from the models of Perng et al. are included in Figure 1. These are random forests (RF), k-nearest neighbors (KNN), support vector machines (SVM), and Softmax. These models are combined with different methods of feature construction: principal component analysis (PCA), an autoencoder (AE), and a convolutional neural network (CNN).

As the data and code used in (Perng et al., 2019)

Model	AUC
XGBOOST	0.857 ± 0.019
LR	0.819 ± 0.019
SAPS-II	0.797 ± 0.016

Figure 2: AUC values of models from (Hou et al., 2020), Figure 4, for predicting 30-day mortality of sepsis patients

is not publicly available, my attempt to reproduce their results will be done using data from the MIMIC III database, as assembled and posted by (Hou et al., 2020). (Hou et al., 2020) similarly uses patient data to predict 30-day mortality for sepsis patients, but does not use any deep learning models. Instead, it uses extreme gradient boosting (XGBoost), a simplified acute physiology score (SAPS-II), and logistic regression (LR). XGBoost is identified as having the best performance. The performances of the models from Hou et al. are included in Figure 2.

For my project, I attempted to verify the claims from (Perng et al., 2019) that the best performance for predicting sepsis mortality was achieved using a CNN for feature selection combined with SoftMax for prediction. In addition to the models evaluated in (Perng et al., 2019), I also evaluated Gradient Boosting as proposed by (Hou et al., 2020). Note that SoftMax and logistic regression are equivalent in the context of this project.

In training and evaluating these models, I was unable to reproduce the results of (Perng et al., 2019). I found significant issues in their methods and additional errors in this paper. Most seriously, their choice to use a convolutional neural network doesn't make sense in this context, and the architecture of their CNN is impossible given the size of their input data.

However, I was able to verify that the use of Gradient Boosting proposed by (Hou et al., 2020) does improve the ability to predict mortality in sepsis patients, as well as investigating the impacts of various methods of feature selection and construction.

This is a worthwhile task, as it shows that the results of (Perng et al., 2019) may not be reliable, while also verifying the results of (Hou et al., 2020).

2.1 Addressed claims from the original paper

For my project, I tested the following claims:

- (Perng et al., 2019) claim that they obtained the best performance (among their tested models) for predicting sepsis mortality by using a convolutional neural network for feature construction followed by SoftMax for prediction. I was unable to reproduce this claim, and I do not believe that this claim is accurate.

- (Hou et al., 2020) claim that they obtained the best performance (among their tested models) for predicting sepsis mortality by using XGBoost. I was able to successfully reproduce this claim.

3 Methodology

I evaluated the following models: random forest (RF), k-nearest neighbors (KNN), support vector machine (SVM), logistic regression (LR), and gradient boosting (GB). Note that gradient boosting should produce comparable results to XGBoost, with the exception that XGBoost trains faster. These models are be combined with the following methods of dimensionality reduciton and feature construction: none, principal component analysis, and an autoencoder. A convolutional neural network was also evaluated.

I wrote the code to construct and evaluate these models, referencing the SciKit-Learn documentation (sci) and the PyTorch documention (PyT). I was able to train and evaluate all models in Jupyter notebooks using Google Colaboratory.

For preprocessing, I used mean value imputation to fill in missing values, and I normalized each feature to have mean 0 and standard deviation 1. To evaluate the models, I split the data into a training set (80% of the data) and a testing set (20% of the data). Models were trained on the training set, and evaluated on the testing set. I evaluated the models using AUC and accuracy, for comparability with the results of (Perng et al., 2019) and (Hou et al., 2020). Because there is a class imbalance in the target (more patients survived than died), I also computed precision and recall for each model.

3.1 Model descriptions

For brevity, I will focus on describing the autoencoder and convolutional neural network used in (Perng et al., 2019).

3.1.1 Autoencoder

An autoencoder consists of two parts, an encoder and a decoder. The encoder reduces the dimen-

sionality of the data, while the decoder attempts to reconstruct the original data from the compressed features. They are trained simultaneously, comparing the input to the network with the output, to try to make the reconstruction as accurate as possible.

In (Perng et al., 2019), the input consisted of 53 features. The encoder compressed the data to 16 features, with a fully connected layer with the rectified linear unit (ReLU) activation function. The decoder used a fully connected layer with a sigmoid activation function to recover 53 features from the 16 features produced by the autoencoder. The authors write that they used batch normalization before each activation function, and a dropout rate of 20%. Training was done with an Adam optimizer with 0.001 as the learning rate, and mean squared error (MSE) as the loss function. The "batch" was the entire training dataset, and the maximum number of epochs was 10,000.

After training the autoencoder, the encoder was used for feature construction on the data, which was then used as input to the various prediction models tested.

There are a couple of issues with the autoencoder architecture as described by (Perng et al., 2019). First, the decoder used the sigmoid activation function, which means that the outputs of the autoencoder would have values in the interval $[0, 1]$, thus could not produce negative outputs, while there are negative values in the input features. This limits the ability of the autoencoder to reconstruct the input data. The authors claim that the feature values were normalized to be between -1 and 1 using the normalization

$$z = \frac{x - \mu}{\sigma}.$$

If this were the case, then a $\tanh$ activation function might solve this issue. However, the normalization described does not guarantee values between -1 and 1. Instead, in normalizes the data to have mean 0 and standard deviation 1, but values less than -1 and greater than 1 will still occur.

Furthermore, the authors' use of batch normalization does not make sense here. Batch normalization is typically used for training very deep neural networks, while the autoencoder they describe has only one hidden layer. They also do not divide the data into batches, instead taking the batch size to be the number of values in the training set.

3.1.2 Convolutional neural network

Convolutional neural networks (CNNs) are most commonly applied to tasks in computer vision, such as image classification. They use convolution and pooling layers to extract useful features from raw pixel data, which then are fed through fully connected layers for the prediction task. Because of the structure of the convolution and pooling layers, the outputs are unaffected by perturbations to the data (e.g., shifting pixels one pixel to the right). CNNs also use weight sharing to reduce the complexity of the model. Because of these advantages, CNNs are effective for tasks such as image classification.

In (Perng et al., 2019), a CNN is used for dimensionality reduction on the input data. Because the data is one-dimensional, a 1D CNN is used. The CNN includes three convolution layers, each with 8 filters, a filter size of 16, a stride of 1, and a padding size of 1. A ReLU activation function was used after each convolution layer. Between the convolution layers, max-pooling layers were used, with a pooling size of 2 and a stride of 2. After the convolution and pooling layers, the data was flattened, followed by a fully connected layer with 16 neurons (for dimensionality reduction), a fully connected layer with 128 neurons, and the output layer with 2 neurons. The fully connect layers also used a ReLU activation function, while the output layer used logistic regression.

The CNN was trained with an Adam optimizer with a learning rate of 0.001 and cross entropy loss, and the batch size was the entire training set.

Again, there are issues with the CNN as described in (Perng et al., 2019). First, the decision to apply a CNN in this context is quite strange. The features of the dataset are clinical variables, including demographic data, vital signs, and lab results. For example, some of the features were age, sex, temperature, heart rate, etc. In particular, there is no real meaning to which columns are next to each other in the dataset. Thus, I would not expect that the convolution and pooling layers of the network would produce meaningful features. The authors highlight the weight sharing and feature construction of a CNN, but they fail to justify why the tranlation-equivariance is desirable in this situation.

Also, the architecture described by the authors is impossible given the size of their data. For a one-dimensional convolution layer, the size of the

Layer	Input Size	Output Size
First Convolution	53	40
First Max Pooling	40	20
Second Convolution	20	7
Second Max Pooling	7	3
Third Convolution	3	*

Figure 3: Input and output sizes of layers of the CNN described in (Perng et al., 2019)

output L_{out} can be computed from the size of the input L_{in} with the formula

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2 \times \text{padding} - \text{filter size}}{\text{stride}} \right\rfloor + 1.$$

Similarly, for a pooling layer, the the size of the output L_{out} can be computed from the size of the input L_{in} with the formula

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} - \text{filter size}}{\text{stride}} \right\rfloor + 1.$$

In (Perng et al., 2019), the data begins with 53 features. The table 3 includes computations of the input and output size for the first few layers, using the above formulas and the parameters specified by (Perng et al., 2019). Note that at the third convolution layer, a kernel size of 16 becomes impossible, as the size of the input is only 3. Thus, the architecture described is not possible given the size of the input data.

3.2 Data descriptions

The paper (Hou et al., 2020) included data on sepsis patients selected from the MIMIC III database. This dataset consist of 89 features, including demographic data, information on their stay in the hospital, vital signs, and laboratory values.

In the paper (Perng et al., 2019), data was obtained from the Chang Gung Research Database, and the data is not publicly available. This dataset consists of 53 features of demographic data, vital signs, and lab results. Because of the similarities in the features collected, I believe that using the data of (Hou et al., 2020) is a reasonable substitution for attempting to reproduce their results.[2]

[2]My original intention was to assemble a new dataset from the MIMIC III database, selecting records of sepsis patients while retaining sequential information, in order to test the sequential models described in (Chu et al., 2020). I made some progress on this task before switching my focus to the issues in the paper (Perng et al., 2019).

3.3 Hyperparameters

Despite the issues described above, I tried to use the hyperparameters of models proposed by (Perng et al., 2019) as much as possible. In my attempt to reproduce the authors' autoencoder, I faithfully replicated the architecture described, except that I reduced the number of features to 40 instead of 16. I trained the autoencoder for 100 epochs, and used a batch size of 64. As in (Perng et al., 2019), I used 0.001 as the learning rate.

The impossible architecture of the CNN remained an issue the dataset I used, which has 89 features. Thus, I could not exactly replicate the CNN as described. Instead, I reduced the filter size of the convolution layers to 3 and the number of neurons in the first fully connected layer to 1024, keeping the rest of the hyperparameters as described in (Perng et al., 2019). I used a batch size of 64, and only trained the CNN for 10 epochs, as it was not improving. I also used a standalone CNN for classification, rather than attempting to use it for dimensionality reduction. As in (Perng et al., 2019), the convolution layers had 8 filters, stride 1, and padding size 1. The pooling layers had pooling size 2 an stride 2. The second fully connected layer had 128 neurons, and the output layer had 2 neurons. I trained the network with Adam optimizer with a learning rate of 0.001.

3.4 Implementation

I wrote the code to construct and evaluate these models, referencing the SciKit-Learn documentation (sci) and the PyTorch documention (PyT). I was able to train and evaluate all models in Jupyter notebooks using Google Colaboratory.

LINK GITHUB

3.5 Computational requirements

My code was run using a Jupyter Notebook in Google Colaboratory, with the default CPU. I used the commands `%time` and `%memit` to check the time and memory requirements for training each model, and those results are included in Figure 4.

4 Results

I evaluated models using accuracy, precision, recall and AUC. The results for these models with no feature construction are given in Figure 5. From these results, we see that gradient boosting had the highest accuracy, recall, and AUC. The random forest had a higher precision, but poor recall.

Model	Peak Memory	Total CPU Time
LR	296.12 MiB	562 ms
SVM	296.14 MiB	1.39 s
RF	296.29 MiB	2.89 s
GB	296.34 MiB	6.65 s
KNN	296.34 MiB	296 ms
PCA	205.98 MiB	629 ms
AE	455.53 MiB	9.74 s
CNN	693.91 MiB	1 m 43 s

Figure 4: Computational Requirements of Training Models

Model	Accuracy	Precision	Recall	AUC
LR	0.8849	0.7967	0.5506	0.7583
SVM	0.8651	0.8767	0.3596	0.6736
RF	0.8739	0.9315	0.3820	0.6876
GB	0.8980	0.8761	0.5562	0.7686
KNN	0.8026	0.4917	0.3315	0.6242

Figure 5: Model Performance with no feature construction

Thus, I conclude that gradient boosting has the best performance.

The results for these models with PCA for dimensionality reduction are given in Figure 6. Given the dimensionality reduction, the results are predictably worse than with no feature construction. Interestingly, we see that logistic regression has the best performance when combined with PCA.

The results for these models with an autoencoder used for dimensionality reduction are given in Figure 7. We obtain similar results to PCA, finding worse performance than with no dimensionality reduction, and the best performance from logistic regression.

The convolutional neural network only achieved an accuracy of 80.5% (the survival rate of patients), and classified all patients as surviving. Thus, the CNN produced the worst results of all models evaluated.

Model	Accuracy	Precision	Recall	AUC
LR	0.8651	0.7670	0.4438	0.7056
SVM	0.8596	0.8676	0.3315	0.6596
RF	0.8465	0.7969	0.2865	0.6344
GB	0.8454	0.7177	0.3427	0.6550
KNN	0.8114	0.5238	0.3708	0.6445

Figure 6: Model Performance with dimensionality reduction from PCA

Model	Accuracy	Precision	Recall	AUC
LR	0.8651	0.7570	0.4551	0.7098
SVM	0.8662	0.9	0.3539	0.6722
RF	0.8618	0.9333	0.3146	0.6546
GB	0.8607	0.8312	0.3596	0.6709
KNN	0.8081	0.5114	0.3764	0.6446

Figure 7: Model Performance with dimensionality reduction from an autoencoder

4.1 Result 1

In the paper (Perng et al., 2019), the authors claimed that the CNN with SoftMax gives the best performance. I was not able to reproduce this result.

4.2 Result 2

In the paper (Hou et al., 2020) that a gradient boosting classifier gives the best result of the studied algorithms. I have reproduced this result.

4.3 Additional results not present in the original paper

I testing a CNN with an architecture that can actually be used with this data, which was not the case in (Perng et al., 2019). I also tested PCA and the autoencoder for dimensionality reduction with more models than were present in the original paper, and I used a different dataset. I was successful at reproducing the result of a different paper, by (Hou et al., 2020).

5 Discussion

I was not able reproduced the result of (Perng et al., 2019) that the CNN with SoftMax gives the best performance. As discussed above, the architecture described is impossible given the input size, and a CNN is not an appropriate choice for this task. My attempt to train a CNN for this task produced the worst performing model. Furthermore, the authors' autoencoder also had issues with its design.

In addition to the errors in the models, there were several claims in (Perng et al., 2019) that I do not believe to be accurate. In their introduction, they claim "Few studies have compared [machine learning models'] accuracy in medical prediction models, and there have been no studies to date in which combining feature extraction and classification of machine learning models has been used to increase discrimination ability." Given what I have learned in CS 598, I confidently believe this claim

to be false, and that it was false as of the writing of the paper. There are many issues with terminology in the paper, including using "medium" instead of "median", using "vector" instead of "feature", claiming that AUC can only be computed for regression methods, and inaccurate descriptions of PCA and KNN.

These issues, along with with the issues in model architectures, should have not have made it through peer review. This paper should have not been published in its current form.

A strength of my approach was using a similar but distinct dataset to test the reproducibility of the result. A weakness was that I was not also able to test the results with the original dataset, which could have provided additional insight into the validity of the results.

5.1 What was easy

Once I started using the dataset from (Hou et al., 2020), I found it straightforward to train baseline classification models, as the data was in a highly usable form.

5.2 What was difficult

When I was originally trying to reproduce the results of (Chu et al., 2020), I spent a great deal of time understanding the structure of the MIMIC III database and figuring out how to extract data to train the sequential models. Unfortunately, as I shifted the focus of my project, I was unable to use this work.

It was difficult working with the paper (Perng et al., 2019), to decide if the issues I encountered were my own misunderstanding or if they were mistakes by the authors. Once I found multiple serious errors, I became more confident in my own understanding, and I was able to make more progress working through the paper.

5.3 Recommendations for reproducibility

The authors should publish their code publicly in a GitHub repository. This would be helpful to determine if the authors made mistakes in describing the architecture of their models, or if the architecture really was nonsensical.

Furthermore, the reviewers and editors who allowed this paper to be published should have done a more thorough review of the paper, and required significant revisions prior to publication.

6 Communication with original authors

I have not yet communicated with the original authors. I am not sure how to go about this, given the serious issues I have found in their results.

References

Pytorch documentation. https://pytorch.org/docs/stable/index.html. Accessed: 2022-04-16.

Scikit-learn documentation. https://scikit-learn.org/stable/. Accessed: 2022-04-16.

Jiebin Chu, Wei Dong, and Zhengxing Huang. 2020. Endpoint prediction of heart failure using electronic health records. *Journal of Biomedical Informatics*, 109:103518.

Hong-Fei Deng, Ming-Wei Sun, Yu Wang, Jun Zeng, Ting Yuan, Ting Li, Di-Huan Li, Wei Chen, Ping Zhou, Qi Wang, and Hua Jiang. 2022. Evaluating machine learning models for sepsis prediction: A systematic review of methodologies. *iScience*, 25(1):103651.

Nianzong Hou, Ming zhao Li, Lu He, Bing Xie, Lin Wang, Rumin Zhang, Yong Yu, Xiaodong Sun, Zhengsheng Pan, and Kai Wang. 2020. Predicting 30-days mortality for mimic-iii patients with sepsis-3: a machine learning approach using xgboost. *Journal of Translational Medicine*, 18.

Jau-Woei Perng, I-Hsi Kao, Chia-Te Kung, Shih-Chiang Hung, Yi-Horng Lai, and Chih-Min Su. 2019. Mortality prediction of septic patients in the emergency department based on machine learning. *Journal of Clinical Medicine*, 8(11).