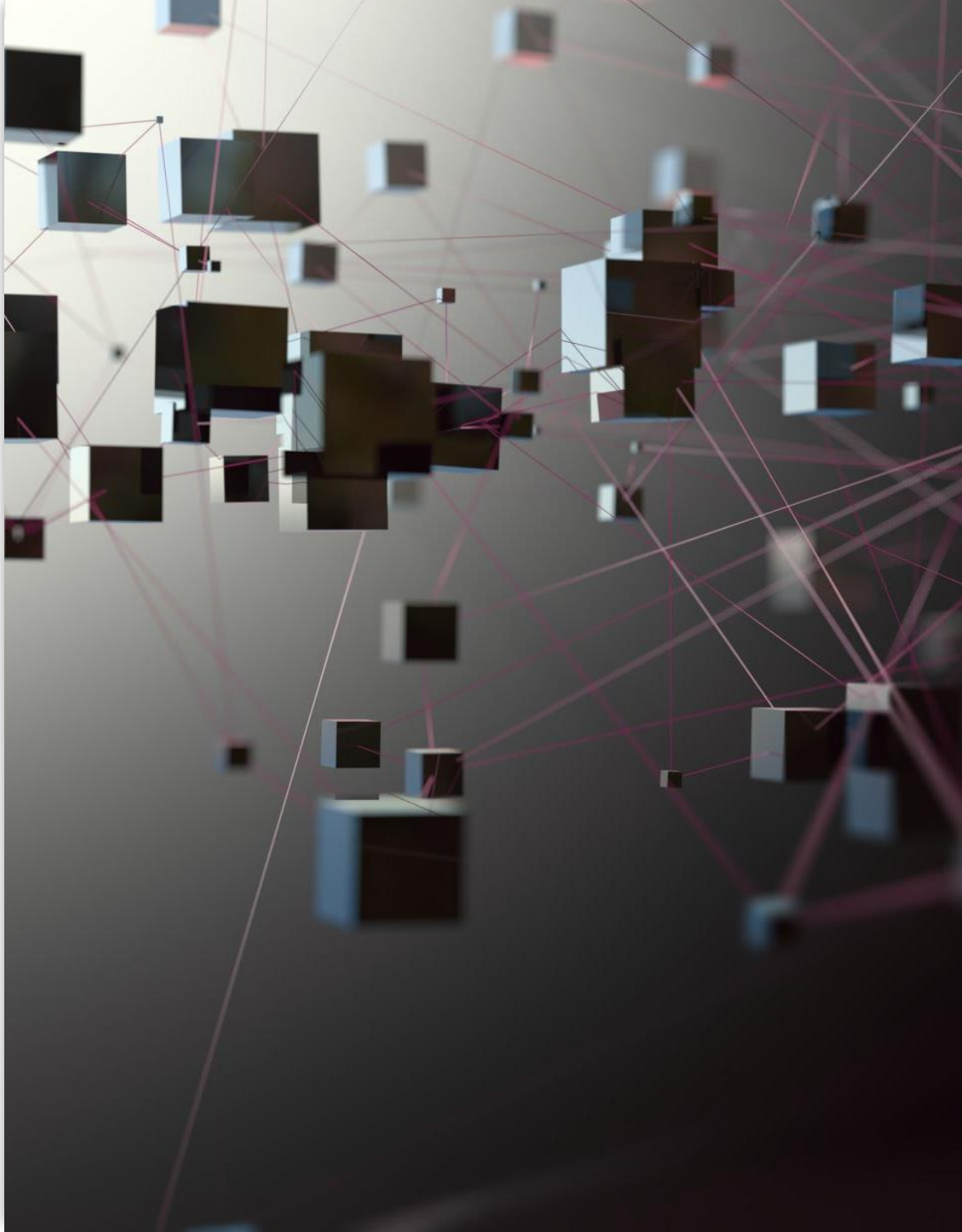
Early Detection of Decompensation in ICU Patients

Jacqueline Long, Wenxuan Zhang

Outline

- Introduction
- Feature Selection
- Data Preprocessing
- Model
- Result
- Conclusion



Introduction

Objective - to develop a predictive model using time series classification to detect early signs of decompensation

- Predict the likelihood of mortality within 24 hours of a patient's admission to the ICU
- By leveraging machine learning techniques and the comprehensive data available in the MIMIC-III dataset, we seek to identify early indicators of fatal outcomes in critically ill patients.

Feature Selection

1

Table: ADMISSIONS, CHARTEVENTS, D_ITEMS, ICUSTAYS, PATIENTS, LABEVENTS

2

Data: subject_id, dod, intime, charttime, gender, admittime, dob, valuenum, itemID

3

Generate: age, hours_until_death, heart_rate, blood pressure, respiratory rate, oxygen saturation, mortality_24h

Extract Data

- SQL

```sql
CREATE TABLE patient_demographics AS
SELECT
    p.subject_id,
    p.gender,
    a.ethnicity,
    EXTRACT(epoch FROM (a.admittime - p.dob))/(60*60*24*365) as age_years
FROM patients p
JOIN admissions a ON p.subject_id = a.subject_id;
```

subject_id	gender	ethnicity	age_years
22	F	WHITE	64.97128234
23	M	WHITE	71.17890982
23	M	WHITE	75.30634323
24	M	WHITE	39.04294901
25	M	WHITE	58.98928082
26	M	UNKNOWN/NOT SPE	72.05379756

```sql
CREATE TABLE mortality_target AS
SELECT
    i.subject_id,
    i.hadm_id,
    i.icustay_id,
    EXTRACT(EPOCH FROM (i.intime - p.dod)) / 3600 AS hours_until_death,
    CASE
        WHEN p.dod BETWEEN i.intime AND i.intime + INTERVAL '24 hours' THEN 1
        ELSE 0
    END AS mortality_24h
FROM icustays i
LEFT JOIN patients p ON i.subject_id = p.subject_id
WHERE p.dod IS NOT NULL;
```

subject_id	hadm_id	icustay_id	hours_until_death	mortality_24h
268	110404	280836	-72.53944444	0
274	130546	254851	-71809.52111	0
275	129886	219649	-276.5186111	0
279	192224	204407	-290.6925	0
281	111199	257572	-163.2438889	0
282	119013	293262	-4422.980556	0
283	109185	231490	-2833.942778	0
283	144156	280531	-1616.354722	0
284	112354	223593	-10664.34056	0
286	106909	260225	-2809.0425	0
286	135917	240627	-387.2308333	0
287	174293	270957	-32407.77417	0
287	175954	245343	-32194.60528	0
291	113649	256641	-48216.90889	0
291	125726	275109	-12947.56194	0
291	126219	246725	-601.2830556	0
292	179726	222505	-5.508333333	1

Data Pre-Processing

- Missing value: filling missing values with calculated min or max

	subject_id	hadm_id	icustay_id	hours_until_death	mortality_24h	charttime	heart_rate	systolic_bp	diastolic_bp	respiratory_rate	oxygen_saturation	gender
0	3	145834	211552	5668.830278	0	NaN	NaN	NaN	NaN	NaN	NaN	M
1	3	145834	211552	5668.830278	0	NaN	NaN	NaN	NaN	NaN	NaN	M
2	9	150750	220597	106.882778	0	NaN	NaN	NaN	NaN	NaN	NaN	M
3	9	150750	220597	106.882778	0	NaN	NaN	NaN	NaN	NaN	NaN	M
4	11	194540	229441	5081.674444	0	NaN	NaN	NaN	NaN	NaN	NaN	F
...	...	...	...	...	...	...	...	...	...	...	...	...
5671559	99995	137810	229633	5578.100556	0	2147-02-10 15:00:00	82.0	122.0	95.0	21.0	97.0	F

Data Pre-processing

- Outliers: drop the age_year > 100 or age_year < 0

	subject_id	hadm_id	icustay_id	hours_until_death	mortality_24h	charttime	heart_rate	systolic_bp	diastolic_bp	respiratory_rate	oxygen_saturation	gender	age_years
8	19	109235	273430	9055.564167	0	NaN	NaN	NaN	NaN	NaN	NaN	M	300.201874
9	19	109235	273430	9055.564167	0	NaN	NaN	NaN	NaN	NaN	NaN	M	300.201874
22	34	144319	290505	8178.574444	0	2191-02-23 05:25:00	72.0	NaN	NaN	14.0	98.0	M	300.201914
23	34	144319	290505	8178.574444	0	2191-02-23 05:25:00	72.0	NaN	NaN	14.0	98.0	M	304.806094
24	34	144319	290505	8178.574444	0	2191-02-23 05:25:00	72.0	NaN	NaN	14.0	98.0	M	300.201914
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5666961	99936	107913	213906	36221.152500	0	2182-10-14 11:01:00	NaN	NaN	NaN	11.0	NaN	F	300.201592
5666962	99936	107913	213906	36221.152500	0	2182-10-14 12:00:00	62.0	149.0	65.0	15.0	96.0	F	300.201592

Methodology

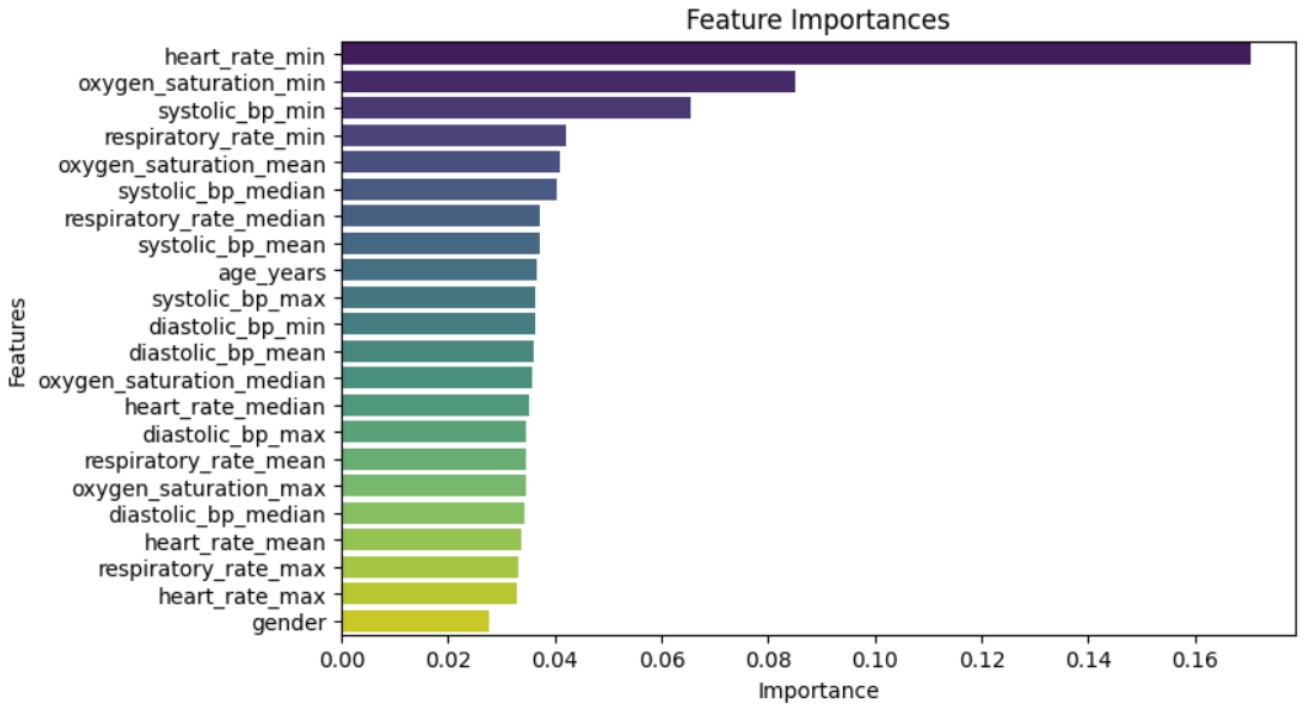
- Each patient have more than one record(rows)
- We use train_test_split based on subject_id to ensure that the same person does not appear in both the training set and the test set.

```
import numpy as np
from sklearn.model_selection import train_test_split

subject_ids = combined_info['subject_id'].unique()
print(len(subject_ids))

# Split subject IDs into train and test sets
train_ids, test_ids = train_test_split(subject_ids, test_size=0.2, random_state=42)
```

Model – Extreme Gradient Boosting

- Extreme Gradient Boosting(XGBoost)
  - operates on the principle of boosting weak learners sequentially by focusing on the errors of the prior learners in the sequence.
  - Capable of handling various types of data, including sequential and structured data.
  - Takes into account not only the immediate features, but also the interactions and relationships among all features.
  - Can learn from previous data patterns to continuously improve upon previous predictions by its iterative nature.

Feature importance analysis



- Our XGBoost model has been trained to identify the most predictive features for our target variable.

- The importance of features related to the minimums in physiological indicators suggests that extreme values are highly indicative of patient outcomes.

Model – Recurrent neural network

- RNN works on the principle of saving the output layer and feeding this back to the input in order to predict the output of the layer.
  - Can handle sequential data
  - Consider the current input and also the previously received inputs
  - Can memorize previous inputs due to its internal memory

TensorFlow's RNN

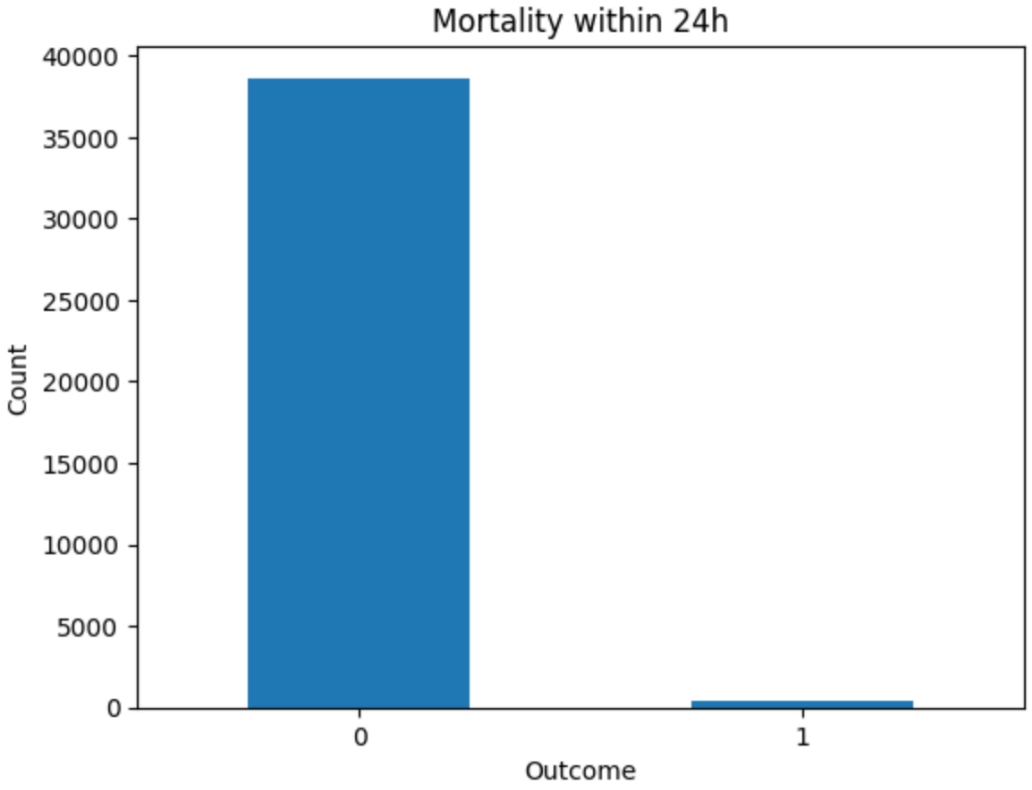
- Sequential model with a SimpleRNN layer followed by a Dense layer with a single unit and sigmoid activation function
- Optimizer: 'adam'
- The model is trained for 10 epochs with a batch size of 64, and a portion of the training data (10% indicated by validation_split=0.1) is held out for validation.

```
Epoch 1/10
46/46 [==============================] - 6s 36ms/step - loss: 2.5023 - accuracy: 0.8148 - val_loss: 0.1817 - val_accuracy: 0.9907
Epoch 2/10
46/46 [==============================] - 1s 14ms/step - loss: 1.2143 - accuracy: 0.9883 - val_loss: 0.2714 - val_accuracy: 0.9783
Epoch 3/10
46/46 [==============================] - 1s 14ms/step - loss: 0.6856 - accuracy: 0.9886 - val_loss: 0.2232 - val_accuracy: 0.9814
Epoch 4/10
46/46 [==============================] - 1s 16ms/step - loss: 0.4529 - accuracy: 0.9897 - val_loss: 0.1136 - val_accuracy: 0.9969
Epoch 5/10
46/46 [==============================] - 1s 15ms/step - loss: 0.2771 - accuracy: 0.9879 - val_loss: 0.0961 - val_accuracy: 1.0000
Epoch 6/10
46/46 [==============================] - 1s 19ms/step - loss: 1.7468 - accuracy: 0.9803 - val_loss: 0.3039 - val_accuracy: 0.9164
Epoch 7/10
46/46 [==============================] - 1s 15ms/step - loss: 0.3077 - accuracy: 0.9272 - val_loss: 0.3114 - val_accuracy: 0.9350
Epoch 8/10
46/46 [==============================] - 1s 14ms/step - loss: 0.2588 - accuracy: 0.9376 - val_loss: 0.2898 - val_accuracy: 0.9412
Epoch 9/10
46/46 [==============================] - 1s 16ms/step - loss: 0.2284 - accuracy: 0.9472 - val_loss: 0.2628 - val_accuracy: 0.9474
Epoch 10/10
46/46 [==============================] - 1s 16ms/step - loss: 0.1964 - accuracy: 0.9583 - val_loss: 0.2328 - val_accuracy: 0.9598
<keras.src.callbacks.History at 0x7871578bf040>
```

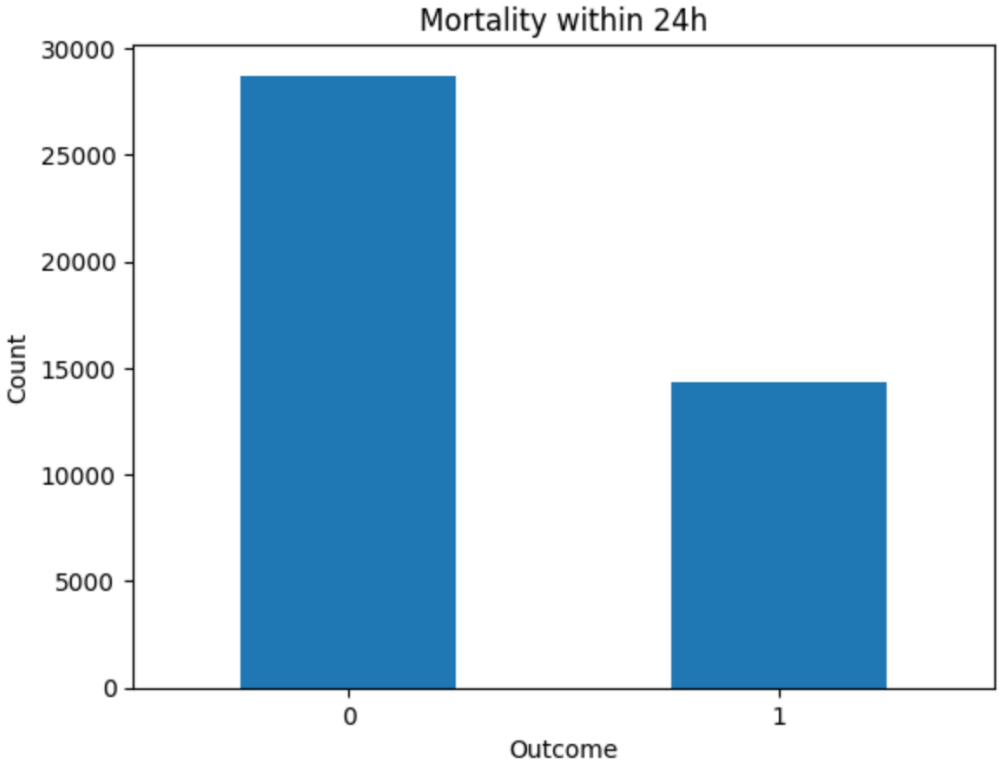
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc}")
```

```
31/31 [==============================] - 0s 9ms/step - loss: 0.2527 - accuracy: 0.9513
Test Accuracy: 0.9513184428215027
```

Imbalanced Data and Solution



- Oversampling(SMOTE) && Undersampling



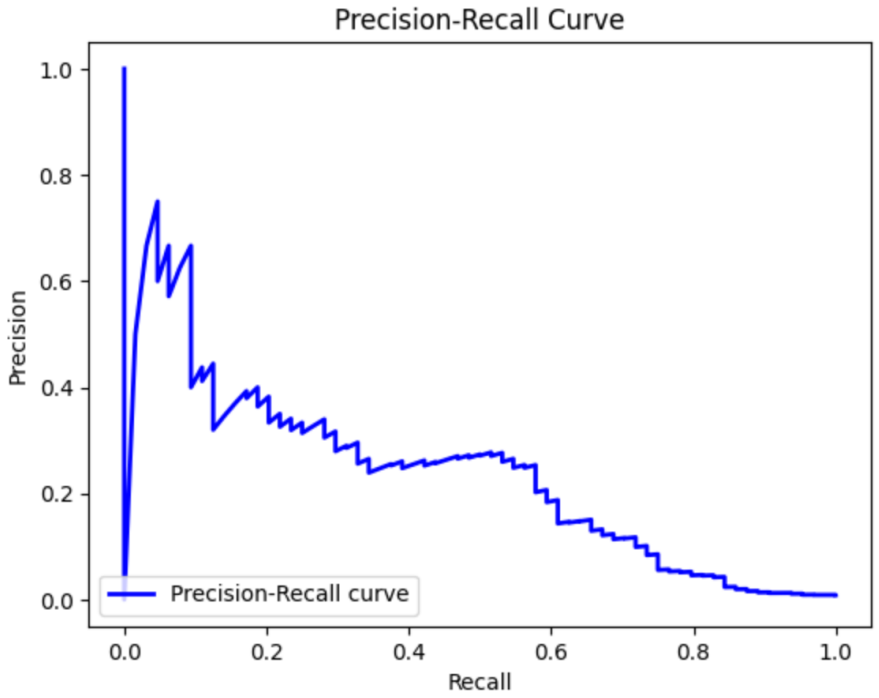
Pytorch RNN

- A network with an RNN layer and a fully connected layer
  - Size of the input features
  - Size of hidden layer
  - Number of layers
  - Number of output classes

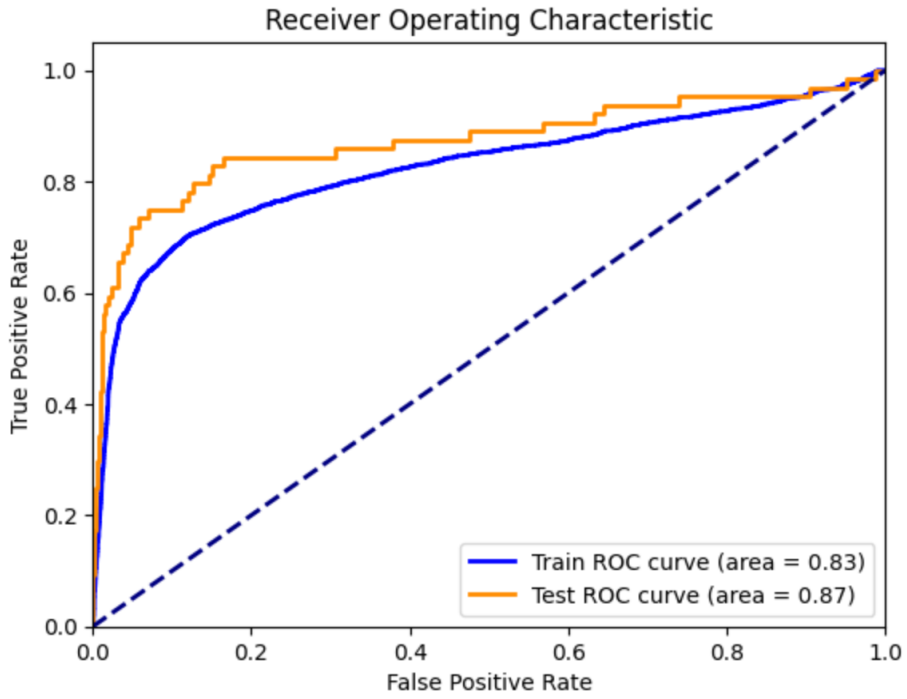
```
Epoch [1/10], Loss: 0.6467
Epoch [2/10], Loss: 0.6161
Epoch [3/10], Loss: 0.4839
Epoch [4/10], Loss: 0.4115
Epoch [5/10], Loss: 0.3183
Epoch [6/10], Loss: 0.3022
Epoch [7/10], Loss: 0.4329
Epoch [8/10], Loss: 0.2284
Epoch [9/10], Loss: 0.3953
Epoch [10/10], Loss: 0.3605
```

Result

Precision-Recall curve



ROC curve

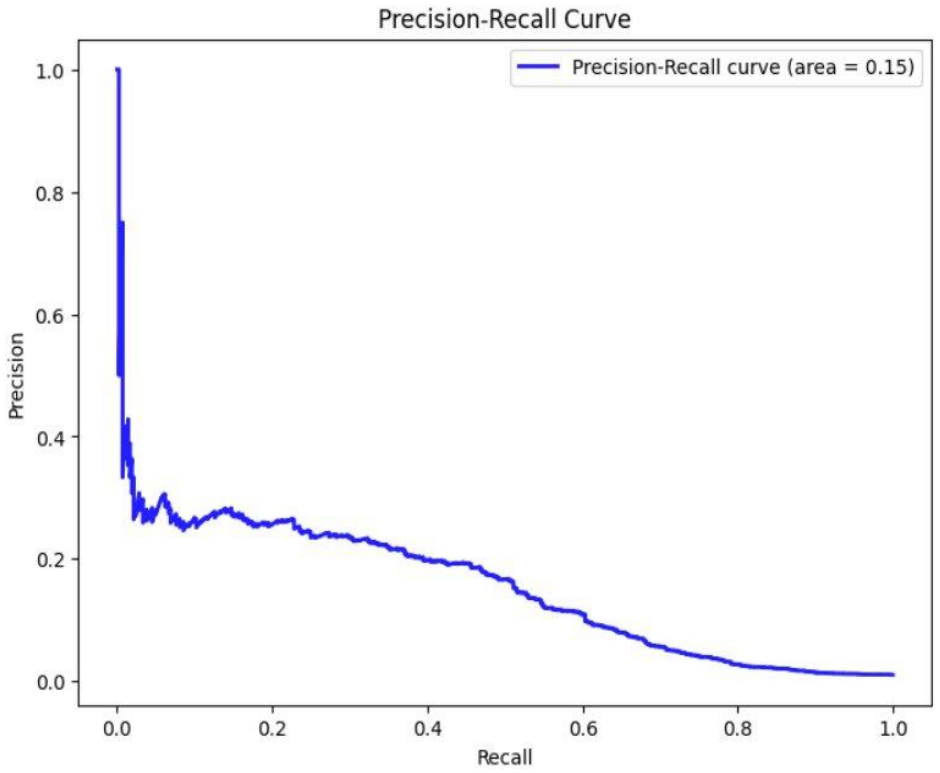


Model – LSTM

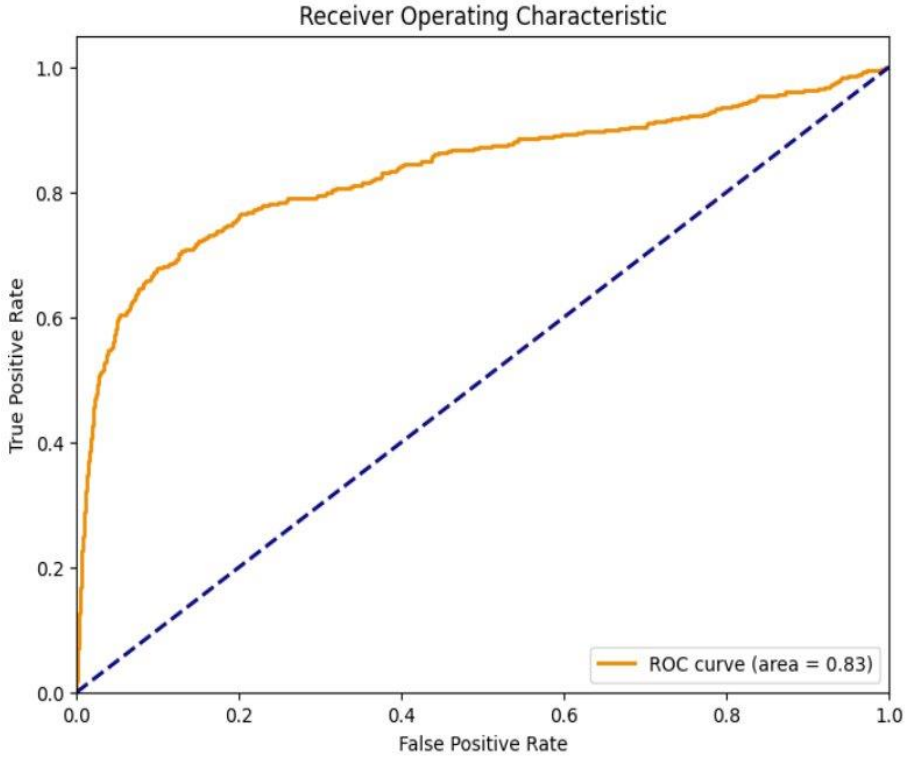
- LSTM is a special type of RNN capable of learning long-term dependencies and avoid the vanishing gradient.
  - 2 layers: BiLSTM for capturing past and future info
  - Can make more accurate predictions by considering the entire context of the input sequence.
  - Improved ability to learn from data where the gap between important information is extensive.

Result

Precision-Recall curve



ROC curve

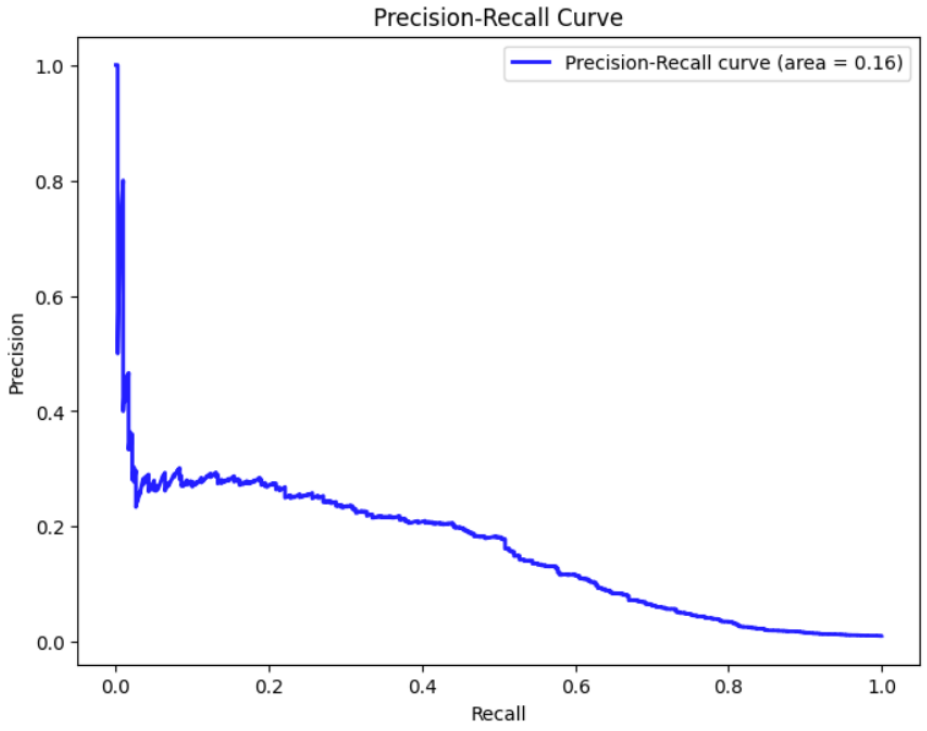


Model – Temporal Convolutional Network

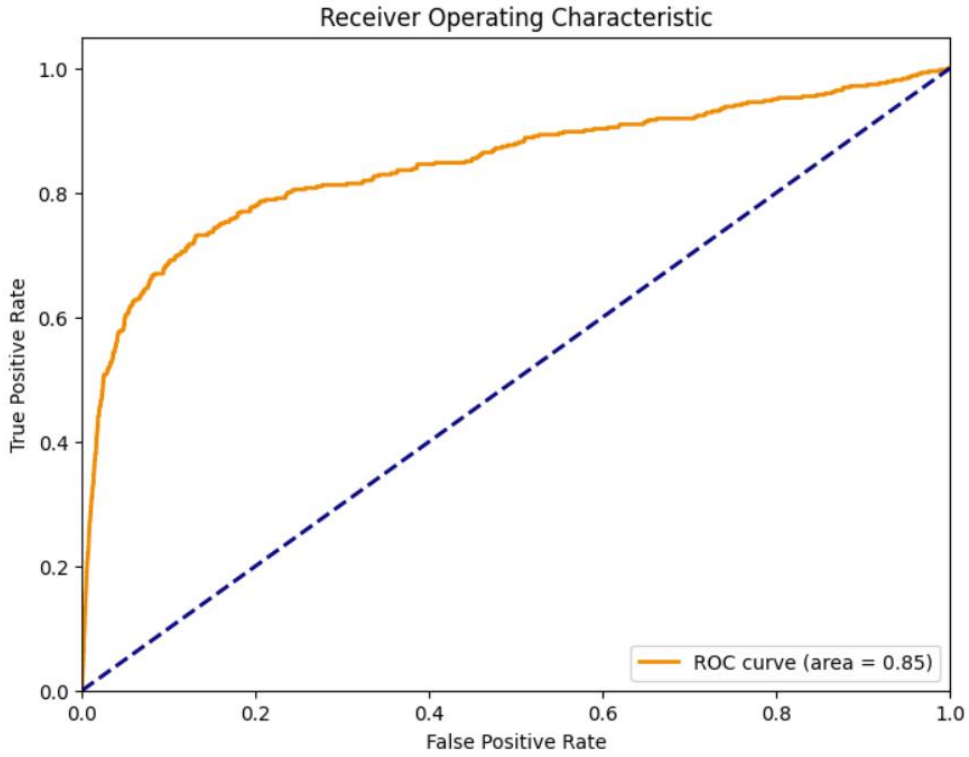
- TCN leverages the strengths of convolutional neural networks for use in sequential prediction
- Compared to RNN-like models, TCN uses causal convolutions and dilated convolutions to effectively capture long-term dependencies in data and flexibly adjust the receptive field.
- As a CNN-based model, it can perform convolutions in parallel across the entire sequence, resulting in faster training speed compared to RNNs and LSTMs.

Result

Precision-Recall curve



ROC curve



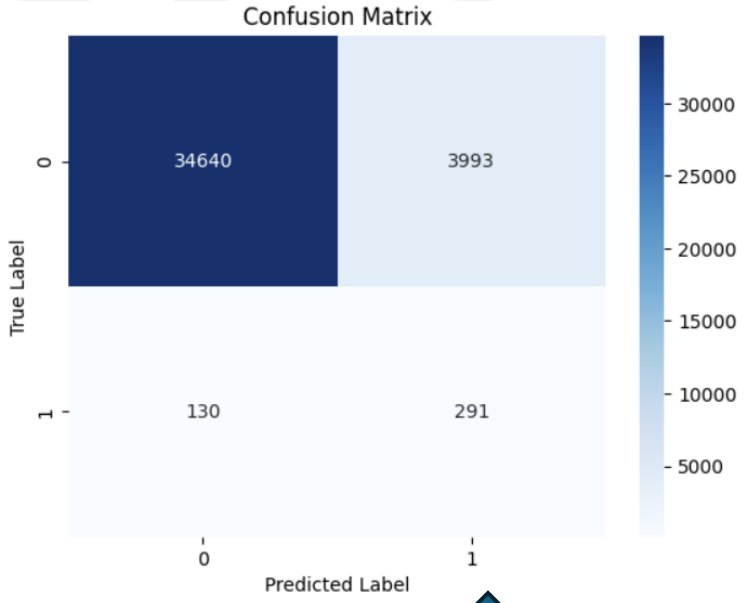
Performance

Table 3: Model performance metrics

Model	Accuracy	Precision	Recall	AUROC	AUPRC
RNN	0.976	0.2	0.578	0.8761	0.23
LSTM	0.9045	0.0721	0.6627	0.8308	0.1533
TCN	0.8944	0.0679	0.6912	0.8453	0.1595

Model Performance

We consider which metric is the most important. Due to such an imbalanced dataset, the value of **true negatives** is the most crucial. It truly matters for the detection of decompensation, even if the reduction of other metrics leads to erroneous judgments or the waste of medical resources.



TCN Confusion matrix (most true negative)

Conclusion

Experience

Used Database tools such as SQL and Python to preprocess the MIMIC-3 dataset

Used XGBoost to observe feature importance

Applied RNN, LSTM, and TCN models to predict the likelihood of decompensation.

Takeaways

Understanding of EMR dataset

Practical machine learning skills

More thoughts on AI in healthcare

Thank you!