Links

- Github: https://github.com/mrmattkennedy/CS598-DLH-Project
- Video: https://youtu.be/gx1JOyjfGmI
- Original study: https://translational-medicine.biomedcentral.com/articles/10.1186/s12967-020-02620-5

Team 65 members:

Matthew Kennedy

Introduction

This study focuses on predicting 30-days mortality for MIMIC-III patients diagnosed with sepsis-3 using the XGBoost algorithm. This is a predictive modeling problem aiming to improve mortality prediction over traditional methods. The importance of solving this problem is aimed at tackling sepsis, which is a major cause of mortality especially in ICU patients. Early and accurate prediction of mortality in these patients is important as it can guide time-critical and appropriate treatment for patients, potentially improving survival outcomes. This study aims to enhance the predictive accuracy using machine learning, which could provide clinicians with a powerful tool for risk assessment and management.

The complexity of sepsis, including its vague syndrome definitions, unknown sources of infection, as well as high mortality rates, makes establishing a reliable and effective prognostic model challenging. Traditional machine learning models, which are based on small sample sizes or simplistic statistical assumptions, are limited in their predictive power. Traditional methods for diagnosing sepsis include the use of serum markers and scoring systems like APHACHE-II and SAPS-II. However, these have limitations in sensitivity, specificity, and the ability to handle complex interactions within data. XGBoost has shown improved predictive performance by efficiently handling missing data and assembling weak prediction models to create a more accurate composite model.

L. et al. [1] proposed a machine learning model using the XGBoost algorithm to predict 30-day mortality among patients with sepsis-3 in the MIMIC-III dataset, comparing its performance against traditional logistic regression and SAPS-II score models. The innovation for this study is found in applying the XGBoost algorithm, known for its efficiency with missing data and capability to enhance predictive accuracy by combining weak models. This study demonstrates the superiority of XGBoost over conventional methods in the context of sepsis mortality prediction.

For post-results analysis and scoring, tthe XGBoost model showed superior performance with higher AUCs compared to traditional logistic regression and SAPS-II models, indicating better predictive accuracy. This study significantly contributes to the sepsis research field by showcasing the application of a machine learning algorithm to accurately predict mortality, potentially aiding clinicians in making informed decisions and tailoring patient management strategies effectively.

Scope of Reproducibility:

Reproducability is possible from a high-level, but there are limitations for a few reasons:

- The dataset utilized in this study is the MIMIC-III dataset, which is too large to reproduce fully exactly as the authors used this data.
- Additionally, access to MIMIC-III is limited, and thus reproducing even certain samples and displaying data is restricted by law. Because of this, to replicate this study, no raw data can be displayed.
- While there are many aspects of the models discussed at length in this study, there are also many details missing. This is discussed in later points as well, but there are quite a few assumptions made about each model. These are pointed out, but the result is likely a model that differs significantly from the models used by L. et al. [1]

For testable hypotheses, I believe the below hypotheses can be tested and observed from L. et al. [1].

1. Hypothesis 1: The XGBoost algorithm can outperform traditional logistic regression and SAPS-II scoring models in predicting 30-day mortality among patients with sepsis-3 based on scoring metrics.

- Corresponding experiment: Create a logistic regression model, a SAPS-II dataset, and an XGBoost model, and use the features specified by L. et al. [1] for these models. Then, analyze the results of both models in predicting 30-day mortality due to sepsis-3 and compare.

1. Hypothesis 2: The set of features identified by the XGBoost model as significant predictors of 30-day mortality are significant compared to features not used

- Corresponding experiment: Perform several ablations, including dropping several features and adding several others, and compare post-training scores across different models to see which features area most important in each model's decisions, and how effective each model is.

Methodology

Environment

For this notebook, we will be using **Python 3**. As for dependencies, the below packages are required, and will be installed in the code cell below:

- google-cloud-bigquery: In this study replication, we will use Google BigQuery to host the data, as well as create views and preprocess data
- pandas: The pandas library is used to create a tabular object to hold the data for the models, as well as post-training data and results
- db-dtypes: This is a required accessory library to use google-cloud-bigquery
- scikit-learn: The sklearn library has a host of uses, including metrics for evaluating models, different types of models, preprocessing, and much more
- xgboost: The XGBoost library will be used to create an XGBoost model
- opencv-python: CV2 can display images for inline
- matplotlib: The Matplotlib library can create charts and graphs, which will be useful for interpreting results

In [1]:
```
!pip install google-cloud-bigquery
!pip install pandas
!pip install db-dtypes
!pip install scikit-learn
!pip install xgboost
```

```
!pip install opencv-python
!pip install matplotlib
```

```
Requirement already satisfied: google-cloud-bigquery in /usr/local/lib/python3.10/dist-p
ackages (3.21.0)
Requirement already satisfied: google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.10.*,!=2.2.*,!=
2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,!=2.8.*,!=2.9.*,<3.0.0dev,>=1.34.1 in /usr/local/l
ib/python3.10/dist-packages (from google-cloud-bigquery) (2.11.1)
Requirement already satisfied: google-auth<3.0.0dev,>=2.14.1 in /usr/local/lib/python3.1
0/dist-packages (from google-cloud-bigquery) (2.27.0)
Requirement already satisfied: google-cloud-core<3.0.0dev,>=1.6.0 in /usr/local/lib/pyth
on3.10/dist-packages (from google-cloud-bigquery) (2.3.3)
Requirement already satisfied: google-resumable-media<3.0dev,>=0.6.0 in /usr/local/lib/p
ython3.10/dist-packages (from google-cloud-bigquery) (2.7.0)
Requirement already satisfied: packaging>=20.0.0 in /usr/local/lib/python3.10/dist-packa
ges (from google-cloud-bigquery) (24.0)
Requirement already satisfied: python-dateutil<3.0dev,>=2.7.2 in /usr/local/lib/python3.
10/dist-packages (from google-cloud-bigquery) (2.8.2)
Requirement already satisfied: requests<3.0.0dev,>=2.21.0 in /usr/local/lib/python3.10/d
ist-packages (from google-cloud-bigquery) (2.31.0)
Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in /usr/local/
lib/python3.10/dist-packages (from google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.10.*,!=2.2.
*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,!=2.8.*,!=2.9.*,<3.0.0dev,>=1.34.1->google-clo
ud-bigquery) (1.63.0)
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.
21.3,!=4.21.4,!=4.21.5,<5.0.0.dev0,>=3.19.5 in /usr/local/lib/python3.10/dist-packages
(from google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.10.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.
6.*,!=2.7.*,!=2.8.*,!=2.9.*,<3.0.0dev,>=1.34.1->google-cloud-bigquery) (3.20.3)
Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in /usr/local/lib/python3.10/dist-
packages (from google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.10.*,!=2.2.*,!=2.3.*,!=2.4.*,!=
2.5.*,!=2.6.*,!=2.7.*,!=2.8.*,!=2.9.*,<3.0.0dev,>=1.34.1->google-cloud-bigquery) (1.63.
0)
Requirement already satisfied: grpcio-status<2.0.dev0,>=1.33.2 in /usr/local/lib/python
3.10/dist-packages (from google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.10.*,!=2.2.*,!=2.3.*,!
=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,!=2.8.*,!=2.9.*,<3.0.0dev,>=1.34.1->google-cloud-bigquer
y) (1.48.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-
packages (from google-auth<3.0.0dev,>=2.14.1->google-cloud-bigquery) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-p
ackages (from google-auth<3.0.0dev,>=2.14.1->google-cloud-bigquery) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages
(from google-auth<3.0.0dev,>=2.14.1->google-cloud-bigquery) (4.9)
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /usr/local/lib/python3.10/d
ist-packages (from google-resumable-media<3.0dev,>=0.6.0->google-cloud-bigquery) (1.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil<3.0dev,>=2.7.2->google-cloud-bigquery) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
t-packages (from requests<3.0.0dev,>=2.21.0->google-cloud-bigquery) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
(from requests<3.0.0dev,>=2.21.0->google-cloud-bigquery) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
ages (from requests<3.0.0dev,>=2.21.0->google-cloud-bigquery) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
ages (from requests<3.0.0dev,>=2.21.0->google-cloud-bigquery) (2024.2.2)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-pa
ckages (from pyasn1-modules>=0.2.1->google-auth<3.0.0dev,>=2.14.1->google-cloud-bigquer
y) (0.6.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
(from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
(from pandas) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages
(from pandas) (1.25.2)
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: db-dtypes in /usr/local/lib/python3.10/dist-packages (1.
2.0)
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.10/dist-package
s (from db-dtypes) (24.0)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages
(from db-dtypes) (2.0.3)
Requirement already satisfied: pyarrow>=3.0.0 in /usr/local/lib/python3.10/dist-packages
(from db-dtypes) (14.0.2)
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.10/dist-packages
(from db-dtypes) (1.25.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24.2->db-dtypes) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
(from pandas>=0.24.2->db-dtypes) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
(from pandas>=0.24.2->db-dtypes) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil>=2.8.2->pandas>=0.24.2->db-dtypes) (1.16.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages
(1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages
(from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages
(from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages
(from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from scikit-learn) (3.5.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.
3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xg
boost) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xg
boost) (1.11.4)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages
(4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages
(from opencv-python) (1.25.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.
7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages
(from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (f
rom matplotlib) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-package
s (from matplotlib) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil>=2.7->matplotlib) (1.16.0)
```

In [2]:

```
# Need to check if google colab or not to properly show images
import sys
```

```
GOOGLE_COLAB ='google.colab' in sys.modules
print('Using Google Colab: ', GOOGLE_COLAB)

if GOOGLE_COLAB:
  from google.colab.patches import cv2_imshow
```

```
Using Google Colab:  True
```

In [3]:
```
import io
import os
import sys
import cv2
import json
import requests
import zipfile

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.cloud import bigquery

# Set env variables
os.environ["GCLOUD_PROJECT"] = 'dlh-project-418923'
os.environ["BQ_DATASET_BASE"] = 'mimiciii_clinical'
os.environ["BQ_DATASET_DERIVED"] = 'mimiciii_derived'
os.environ["BQ_DATASET_FEATURES"] = 'models_features'
os.environ["LOG_REG_FEATURES_VIEW"] = 'logreg_features'
os.environ["XGBOOST_FEATURES_VIEW"] = 'xgboost_features'
os.environ["SAPSII_FEATURES_VIEW"] = 'sapsii_features'
RANDOM_STATE = 42

# URLs for downloading from drive
gcloud_ids = {'sa-creds': '1qNBEkuIQgj0K-PJ9ETvqF6Cel8wMlVGA',
              'mimic-views': '13uCK5Go9XvANkeujrDaWY2cK3ocWakVM',
              'model-features': '1_z6HLDKnHH8iUddS_wMPg3UhexhczpZq',
              'img-features': '1VTAdcxG0Tz6vLhRWhc2b_kWaHjfeK_am',
              'graph-roc': '1Hc_vLhGYR4yU4LG-o42p3rVdUEDHXAuR',
              'graph-dca': '1q6c4b_IWql7hp829K1dDaoS6GEP6Bhcv',
              'graph-cic': '1x56v9kFDM_dS3JcML3UL0BlOSoJr7XGx',
              'graph-nomogram': '1yyFhCauWCAgqRyJe5E-DZdkdfiJiOiIC',
              'my-roc': '1LHFSwLI1X5fOCVtPClQr9od6c3ZfD4GD',
              'my-dca': '1mVIHMui69o3SRv76cbrrm9CnnbsT7zWn',}
gcloud_url = 'https://drive.google.com/uc?export=download&id={}'
```

In [4]:
```
# Define a function to show images based on URL
def show_image(gcloud_id: str, resize: bool=False) -> None:
  # Download bytes from URL
  f = io.BytesIO()
  # Write to BytesIO and reset stream position
  f.write(requests.get(gcloud_url.format(gcloud_ids[gcloud_id])).content)
  f.seek(0)
  # Write bytes into numpy array, then to a cv2 image
  file_bytes = np.asarray(bytearray(f.read()), dtype=np.uint8)
  img = cv2.imdecode(file_bytes, cv2.IMREAD_COLOR)
  # Resize if specified
  if resize: img=cv2.resize(img, (800,600))
  # Close bytesIO
  f.close()

  # Show image
  if GOOGLE_COLAB: cv2_imshow(img)
  else: cv2.imshow('', img)
```

Data

Source

The data used in this project is the MIMIC III dataset. However, due to the size of the original dataset, a subsample of roughly half of the raw original dataset was loaded into Google Bigquery. The reason BQ was chosen was due to the availability of this GitHub repo from MIT that has easily creatable views and analyses of the MIMIC datasets, and can be processed via BQ.

The data is split into 3 datasets in BQ:

1. **mimiciii_clinical**: All of the original, raw data. Data was sourced from Physionet
2. **mimiciii_derived**: Any additional views created from the raw data. SQL for these views was sourced from the mimic-code library mentioned above
3. **mimiciii_features**: Views created that contain the features for each type of model. These features were chosen from L. et al. [1][p.8] based on the author's chose features, which were, "identifed by the results of backward stepwise analysis and strongly associated with mortality in 30 days".

Downloading data

The data already exists in a BigQuery project, as to make creating views easy and working with data much faster. This script will create the necessary views from the raw data, then create feature views based on the derived views.

For data access, downloading, and preprocessing, steps are listed in a cell later describing all the necessary code to be run.

Statistics

Tables and views

Tables contained in 'dlh-project-418923.mimiciii_clinical':

- dlh-project-418923.mimiciii_clinical.admissions, 58976 rows
- dlh-project-418923.mimiciii_clinical.callout, 34499 rows
- dlh-project-418923.mimiciii_clinical.caregivers, 7567 rows
- dlh-project-418923.mimiciii_clinical.chartevents, 330712483 rows
- dlh-project-418923.mimiciii_clinical.cptevents, 573146 rows
- dlh-project-418923.mimiciii_clinical.d_cpt, 134 rows
- dlh-project-418923.mimiciii_clinical.d_icd_diagnoses, 14567 rows
- dlh-project-418923.mimiciii_clinical.d_icd_procedures, 3882 rows
- dlh-project-418923.mimiciii_clinical.d_items, 12487 rows
- dlh-project-418923.mimiciii_clinical.d_labitems, 753 rows
- dlh-project-418923.mimiciii_clinical.datetimeevents, 4285647 rows
- dlh-project-418923.mimiciii_clinical.diagnoses_icd, 651047 rows
- dlh-project-418923.mimiciii_clinical.drgcodes, 115557 rows
- dlh-project-418923.mimiciii_clinical.icustays, 61532 rows
- dlh-project-418923.mimiciii_clinical.inputevents_cv, 14527935 rows
- dlh-project-418923.mimiciii_clinical.inputevents_mv, 3218991 rows
- dlh-project-418923.mimiciii_clinical.labevents, 23851932 rows

- dlh-project-418923.mimiciii_clinical.microbiologyevents, 631726 rows
- dlh-project-418923.mimiciii_clinical.noteevents, 2083180 rows
- dlh-project-418923.mimiciii_clinical.outputevents, 4349218 rows
- dlh-project-418923.mimiciii_clinical.patients, 46520 rows
- dlh-project-418923.mimiciii_clinical.prescriptions, 4156450 rows
- dlh-project-418923.mimiciii_clinical.procedureevents_mv, 258066 rows
- dlh-project-418923.mimiciii_clinical.procedures_icd, 231945 rows
- dlh-project-418923.mimiciii_clinical.services, 73343 rows
- dlh-project-418923.mimiciii_clinical.transfers, 261897 rows

Views contained in 'dlh-project-418923.mimiciii_derived', as well as descriptions:

- dlh-project-418923.mimiciii_derived.blood_gas_first_day -- Highest and lowest blood gas values in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.blood_gas_first_day_arterial -- As above, but arterial blood gases only.
- dlh-project-418923.mimiciii_derived.echo_data -- Text extracted from echocardiography reports using regular expressions.
- dlh-project-418923.mimiciii_derived.elixhauser_ahrq_v37 -- Comorbidities in categories proposed by Elixhauser et al. AHRQ produced the mapping.
- dlh-project-418923.mimiciii_derived.explicit_sepsis -- Explicitly coded sepsis (i.e. a list of patients with ICD-9 codes which refer to sepsis).
- dlh-project-418923.mimiciii_derived.gcs_first_day -- Highest and lowest Glasgow Coma Scale in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.labs_first_day -- Highest and lowest laboratory values in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.sofa -- The Sequential Organ Failure Assessment (SOFA) scale.
- dlh-project-418923.mimiciii_derived.urine_output_first_day -- Total urine output over the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.ventilation_classifications -- Classifies patient settings as implying mechanical ventilation.
- dlh-project-418923.mimiciii_derived.ventilation_durations -- Start and stop times for mechanical ventilation.
- dlh-project-418923.mimiciii_derived.vitals_first_day -- Highest and lowest vital signs in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.sapsii -- Simplified Acute Physiology Score II (SAPS II)

Views contained in 'dlh-project-418923.models_features', which are used for the 2 models in this analysis (logistic regression and XGBoost):

- dlh-project-418923.models_features.logreg_features
- dlh-project-418923.models_features.xgboost_features
- dlh-project-418923.models_features.sapsii_features

Features selected

From L. et al. [1][p.4] regarding feature selection:

"Firstly, the conventional logistic regression model was conducted using these signifcant variables identifed by backward stepwise analysis with Chi-square test. Then we chose an entry probability of p<0.05 by the stepwise selection method. Secondly, in the construction of SAPS II model, we used these time-stamp variables to do prediction based on the methods provided by the original literature of SAPS II. Thirdly, we performed XGBoost model to analysis the contribution (gain) of each variable to 30-days mortality, at the same time, backward stepwise analysis was processed to select the variable with a threshold of p<0.05 according to the Akaike information criterion (AIC)."

Additionally, the below features are used to calculate SAPS-II:

- Age, GCS
- VITALS: Heart rate, systolic blood pressure, temperature
- FLAGS: ventilation/cpap
- IO: urine output
- LABS: PaO2/FiO2 ratio, blood urea nitrogen, WBC, potassium, sodium, HCO3

Below is an image of which features were selected for both models

In [5]:
```
show_image('img-features')
# Figure 1
```

Table 2 Features selected in the conventional logistic regression

	OR_with_CI	p value
(Intercept)	52,913.003 (87.92–33,934,517.782)	< 0.001
Sofa	1.142 (1.106–1.179)	< 0.001
Aniongap_min	1.078 (1.043–1.115)	< 0.001
Creatinine_min	0.676 (0.592–0.767)	< 0.001
Chloride_min	0.98 (0.962–0.999)	0.03393
Hematocrit_min	1.113 (1.053–1.178)	< 0.001
Hemoglobin_min	0.748 (0.623–0.895)	0.00169
Hemoglobin_max	0.926 (0.863–0.992)	0.02993
Lactate_min	1.308 (1.194–1.435)	< 0.001
Potassium_min	1.179 (1.001–1.389)	0.04922
Sodium_max	1.046 (1.019–1.074)	< 0.001
Bun_min	1.033 (1.018–1.048)	< 0.001
Bun_max	0.986 (0.973–0.997)	0.01542
Wbc_min	1.062 (1.036–1.09)	< 0.001
Wbc_max	0.969 (0.952–0.987)	< 0.001
Heartrate_min	0.987 (0.977–0.997)	0.0111
Heartrate_mean	1.022 (1.011–1.033)	< 0.001
Sysbp_min	0.991 (0.984–0.998)	0.00839
Meanbp_min	0.992 (0.985–1)	0.0468
Resprate_mean	1.062 (1.038–1.086)	< 0.001
Tempc_min	0.897 (0.81–0.993)	0.03242
Tempc_max	0.781 (0.698–0.873)	< 0.001
Spo2_mean	0.947 (0.909–0.986)	0.00839
Age	1.029 (1.022–1.035)	< 0.001
Diabetes	0.779 (0.639–0.948)	0.01328
Vent	1.824 (1.48–2.251)	< 0.001

OR odds ratio, *CI* confidence interval, *SOFA* sequential organ failure assessment,

bun blood urea nitrogen, *wbc* white blood cell, *sysbp* systolic blood pressure, *meanbp* mean blood pressure, *resprate* respiratary rate, *Spo2* oxyhemoglobin saturation, *vent* ventilation, *Max* maximum, *Min* minimum

Table 3 Features selected in the XGboost model

	OR_with_CI	*P*
(Intercept)	493.907 (9.063–27,931.087)	0.00247
Urineoutput	1 (1–1)	< 0.001
Lactate_min	1.401 (1.288–1.527)	< 0.001
Bun_mean	1.018 (1.013–1.023)	< 0.001
Sysbp_min	0.979 (0.974–0.984)	< 0.001
Metastatic_cancer	2.997 (2.217–4.038)	< 0.001
Inr_max	1.058 (1.002–1.115)	0.03709
Age	1.019 (1.013–1.025)	< 0.001
Sodium_max	1.016 (1.001–1.031)	0.03835
Aniongap_max	1.048 (1.026–1.069)	< 0.001
Creatinine_min	0.766 (0.686–0.852)	< 0.001
Spo2_mean	0.897 (0.865–0.93)	< 0.001

OR odds ratio, *CI* confidence interval, *bun* blood urea nitrogen, *sysbp* systolic blood pressure, *INR* international normalized ratio, *Spo2* oxyhemoglobin saturation, *Max* maximum, *Min* minimum

Features statistics

The target classification of this analysis is to accurately predict if a patient will die within 30 days of their first visit from sepsis. From the subsample of data present in BQ, there are 877 positive records (patients who died within 30 days), and 1443 negative records (patients who did not die within 30 days). All features are numerical, either discrete or continuous.

For the model hyperparameters, not much is described in the paper such as number of epochs, scoring functions, learning rates, etc, so for this paper we will use standard approaches to each, as the hypothesis tested in the paper are less focused on hyperparameter tuning, and more about model selection strengths.

Data preprocessing

Again, not much is described in terms of data preparation and pre-processing - the focus of the paper is on parameter selection and model performance. Because of this, we will be using basic model preprocessing, which will involve the below steps:

1. Download SQL for views, as well as authenticate BigQuery client
2. Drop and reset derived and feature datasets
3. Create views in BQ from raw data tables
4. Create feature-selection views from all derived views and raw data tables
5. Download data from the respective view for all models
6. Drop records that are missing all non-target data
7. Drop non-feature columns
8. Feature normalization using sklearn MinMaxScaler, in order to scale data between 0 and 1 and prevent negative values
9. Create train and test splits for data

To preprocess the data, simply run the below 9 cells for each of the 9 steps specified above.

In [6]:
```
# 1. Download SQL for views, as well as authenticate BigQuery client

# Read SA credentials for BQ
f = io.BytesIO()
f.write(requests.get(gcloud_url.format(gcloud_ids['sa-creds'])).content)
creds = json.loads(f.getvalue().decode())
# Authorize and close
client = bigquery.Client.from_service_account_info(creds)
f.close()

# Read view SQL files
f = io.BytesIO()
f.write(requests.get(gcloud_url.format(gcloud_ids['mimic-views'])).content)
# Parse from memory into dict
view_data = {}
with zipfile.ZipFile(f, 'r') as zipf:
    for file in zipf.namelist():
        # Create file name and data from decoded zipf bytes
        fname = os.path.basename(file)[:-4].lower() #Last 4 chars are .sql
        # Store into StringIO for pandas to read
        data = zipf.read(file).decode()
        view_data[fname] = data
f.close()


# Download model feature SQL files
f = io.BytesIO()
f.write(requests.get(gcloud_url.format(gcloud_ids['model-features'])).content)
# Parse from memory into dict
model_views_data = {}
with zipfile.ZipFile(f, 'r') as zipf:
    for file in zipf.namelist():
        # Create file name and data from decoded zip bytes
        fname = os.path.basename(file)[:-4].lower() #Last 4 chars are .sql
        # Store into StringIO for pandas to read
        data = zipf.read(file).decode()
        model_views_data[fname] = data
f.close()
```

In [7]:
```
# 2. Drop and reset derived and feature datasets

# Create datasets
for dname in [os.environ["BQ_DATASET_DERIVED"], os.environ["BQ_DATASET_FEATURES"]]:
    # Get a list of datasets
    datasets = list(client.list_datasets())  # Make an API request.
    project = client.project
    dataset_id = "{}.{}".format(client.project, dname)

    # Check if the dataset already exists
    dataset_already_exists = dname in [d.dataset_id for d in datasets]

    # Delete if it exists already
    if dataset_already_exists:
        client.delete_dataset(dataset_id, delete_contents=True, not_found_ok=True)  # Ma
        print("Deleted dataset '{}'".format(dataset_id))

    # Create new dataset
    dataset = bigquery.Dataset(dataset_id)
    dataset.location = "US"

    # Send the dataset to the API for creation, with an explicit timeout.
    # Raises google.api_core.exceptions.Conflict if the Dataset already
```

```
    # exists within the project.
    dataset = client.create_dataset(dataset, timeout=30)  # Make an API request.
    print("Created dataset {}.{}".format(client.project, dataset.dataset_id))
```

```
Deleted dataset 'dlh-project-418923.mimiciii_derived'
Created dataset dlh-project-418923.mimiciii_derived
Deleted dataset 'dlh-project-418923.models_features'
Created dataset dlh-project-418923.models_features
```

In [8]:
```
# 3. Create views in BQ from raw data tables

# Create views
for view_name, sql in view_data.items():
    if 'sofa' in view_name: continue #Save sofa for last
    if 'sapsii' in view_name: continue #Save sapsii for last

    # Specify view ID
    view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_DER
    view = bigquery.Table(view_id)
    view.view_query = sql

    # Make an API request to create the view.
    view = client.create_table(view)
    print(f"Created {view.table_type}: {str(view.reference)}")

# Create sofa view
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_DERIVED
view = bigquery.Table(view_id)
view.view_query = view_data['sofa']
# Make an API request to create the view.
view = client.create_table(view)
print(f"Created {view.table_type}: {str(view.reference)}")


# Create sapsii view
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_DERIVED
view = bigquery.Table(view_id)
view.view_query = view_data['sapsii']
# Make an API request to create the view.
view = client.create_table(view)
print(f"Created {view.table_type}: {str(view.reference)}")
```

```
Created VIEW: dlh-project-418923.mimiciii_derived.blood_gas_first_day
Created VIEW: dlh-project-418923.mimiciii_derived.blood_gas_first_day_arterial
Created VIEW: dlh-project-418923.mimiciii_derived.echo_data
Created VIEW: dlh-project-418923.mimiciii_derived.elixhauser_ahrq_v37
Created VIEW: dlh-project-418923.mimiciii_derived.explicit_sepsis
Created VIEW: dlh-project-418923.mimiciii_derived.gcs_first_day
Created VIEW: dlh-project-418923.mimiciii_derived.labs_first_day
Created VIEW: dlh-project-418923.mimiciii_derived.urine_output_first_day
Created VIEW: dlh-project-418923.mimiciii_derived.ventilation_classifications
Created VIEW: dlh-project-418923.mimiciii_derived.ventilation_durations
Created VIEW: dlh-project-418923.mimiciii_derived.vitals_first_day
Created VIEW: dlh-project-418923.mimiciii_derived.sofa
Created VIEW: dlh-project-418923.mimiciii_derived.sapsii
```

In [9]:
```
# 4. Create feature-selection views from all derived views and raw data tables

# Create model feature views
for view_name, sql in model_views_data.items():
    # Specify view ID
    view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEA
    view = bigquery.Table(view_id)
    view.view_query = sql

    # Make an API request to create the view.
```

```
    view = client.create_table(view)
    print(f"Created {view.table_type}: {str(view.reference)}")
```

```
Created VIEW: dlh-project-418923.models_features.logreg_features
Created VIEW: dlh-project-418923.models_features.sapsii_features
Created VIEW: dlh-project-418923.models_features.xgboost_features
```

In [10]:
```
# 5. Download data from the respective view for all models

# Get logistic regression model data
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURE
view = client.get_table(view_id)
query_job = client.query(view.view_query)
df_logreg = query_job.result().to_dataframe()
print(f'Successfully downloaded data for logistic regression - shape {df_logreg.shape}')

# Get XGBoost model data
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURE
view = client.get_table(view_id)
query_job = client.query(view.view_query)
df_xgb = query_job.result().to_dataframe()
print(f'Successfully downloaded data for XGBoost - shape {df_xgb.shape}')

# Get SAPS-II model data
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURE
view = client.get_table(view_id)
query_job = client.query(view.view_query)
df_sapsii = query_job.result().to_dataframe()
print(f'Successfully downloaded data for SAPS-II - shape {df_sapsii.shape}')
```

```
Successfully downloaded data for logistic regression - shape (2320, 30)
Successfully downloaded data for XGBoost - shape (2222, 15)
Successfully downloaded data for SAPS-II - shape (2019, 22)
```

In [11]:
```
# 6. Drop records that are missing all non-target data
# For logistic regression, we cannot have any null values, so any records with null valu

df_logreg.dropna(how='any', inplace=True)
print(f'Successfully purged null values for logistic regression - shape {df_logreg.shape

df_xgb.dropna(thresh=len(df_xgb.columns)-1, inplace=True)
print(f'Successfully purged null values data for XGBoost - shape {df_xgb.shape}')

df_sapsii.dropna(thresh=len(df_sapsii.columns)-1, inplace=True)
print(f'Successfully purged null values data for SAPS-II - shape {df_sapsii.shape}')
```

```
Successfully purged null values for logistic regression - shape (2021, 30)
Successfully purged null values data for XGBoost - shape (2193, 15)
Successfully purged null values data for SAPS-II - shape (1670, 22)
```

In [12]:
```
# 7. Drop non-feature columns
non_feature_cols = ['subject_id', 'hadm_id', 'icustay_id']

df_logreg.drop(non_feature_cols, axis=1, inplace=True)
df_xgb.drop(non_feature_cols, axis=1, inplace=True)
df_sapsii.drop(non_feature_cols, axis=1, inplace=True)
```

In [13]:
```
# 8. Feature normalization using sklearn MinMaxScaler, in order to scale data between 0
# We will not be scaling SAPS-II values since the probability is already calculcated, an

from sklearn.preprocessing import MinMaxScaler

# Scale logreg features
scaler = MinMaxScaler()
df_logreg[df_logreg.columns] = scaler.fit_transform(df_logreg[df_logreg.columns])
```

```
# Scale xgb features
scaler = MinMaxScaler()
df_xgb[df_xgb.columns] = scaler.fit_transform(df_xgb[df_xgb.columns])
```

In [14]:
```
# 9. Create train and test splits for data

from sklearn.model_selection import train_test_split

# Set test size to 15% of the dataset
test_size = 0.1

# Create logistic regression X and y data
X_logreg = df_logreg.drop('target', axis=1)
y_logreg = df_logreg[['target']]
X_train_logreg, X_test_logreg, y_train_logreg, y_test_logreg = train_test_split(X_logreg

# Create xgboost X and y data
X_xgb = df_xgb.drop('target', axis=1)
y_xgb = df_xgb[['target']]
X_train_xgb, X_test_xgb, y_train_xgb, y_test_xgb = train_test_split(X_xgb, y_xgb, test_s

# Create SAPSII X, y_true, and y_pred datasets
X_sapsii = df_sapsii.drop('target', axis=1)
y_true_sapsii = df_sapsii[['target']]
y_pred_sapsii = df_sapsii['sapsii_prob']

# Check sizes
print('Logistic regression data shape\n---------------')
print(f'\tX: {X_logreg.shape}')
print(f'\ty: {y_logreg.shape}')
print()
print('XGBoost data shape\n---------------')
print(f'\tX: {X_xgb.shape}')
print(f'\ty: {y_xgb.shape}')
print('SAPS-II data shape\n---------------')
print(f'\ty: {df_sapsii.shape}')
```

```
Logistic regression data shape
---------------
        X: (2021, 26)
        y: (2021, 1)

XGBoost data shape
---------------
        X: (2193, 11)
        y: (2193, 1)
SAPS-II data shape
---------------
        y: (1670, 19)
```

Sizes can vary slightly between the number of valid records between models because the features selected for each model are different, so there can be more or less records with missing values between the two models.

Model

The model includes the model definitation which usually is a class, model training, and other necessary parts.

- Model architecture: layer number/size/type, activation function, etc
- Training objectives: loss function, optimizer, weight of each loss term, etc

- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.
- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

Model summary

There are 3 models included in testing for this paper. They are not pretrained, and will be simply defined below:

1. Linear regression, which will be made using the sklearn library
2. XGBoost model, based on the XGBoost library. This is a type of ensemble method, similar to random forest, but includes features such as efficient regularization, parallel processing, and sparse data handling
3. SAPS-II, which is a probabilistic model and does not require any additional training, as the disease mortality probability is included in the dataset

Models 1 and 3 (Linear Regression and SAPS-II) are considered "traditional" approaches in the paper for predicting ICU mortality rates within timeframes. Often times, a certain threshold is applied to SAPS-II scores, but this paper does not reference one. The goal of the study is to show that more advanced machine learning approaches, such as XGBoost, can outperform traditional methods.

Model architecture

This study focuses heavily on standard approches and evaluation vs newer, less-tested approaches (XGBoost). However, there is little mentioned about model architecture details.

1. For the linear regression model, the formula is very basic and has little tuning involved, so no details are specified or changed.
2. For the XGBoost model, there are quite a few architecture parameters that can be changed, such as:

    - n_estimators - the number of trees in the model
    - max_depth - maximum depth of trees, which helps control how specialized trees are in the ensemble
    - learning rate - the eta parameter, this controls how much each tree contributes to the overall model

    However, within the paper, there are no architecture parameters specified. Instead of trying to give an advantage to the XGBoost model via hyperparameters that were not specified in this paper, we will assume all default values for the model to compare the base-state of the XGBoost model to linear regression and SAPS-II results.

3. For the SAPS-II model, the probability is calculated based on Simplified Acute Physiology Score II, which is a measure of patient severity of illness. This score ranged from 0 and 163, with a predicted mortality of 0 to 1, with 1 being certain death. We will be using this predicted mortality for our model.

Training objectives

1. Linear regression has a non-customizable loss function based on the linear regression equation and we won't be customizing any loss hyperparameters
2. For XGBoost, we will be using the default squarederror loss, as we do not want to make hyperparameter assumptions the authors didn't inform the readers of. This includes other

hyperparameters including optimizers, learning steps, minimum loss, etc.
3. SAPS-II is a probabilistic model and has no training involved

Computation requirements

There are limited computational requirements for these models - the unchanged regressors have very limited number of weights, as well as hyperparameters. The number of trees, max depth, and other parameters discussed earlier, when left unchanged, have minimal requirements for the model. Any modern computer can host these models. The largest requirements come from being able to hold the data, which is done via bigquery. The dataframes are not holding much.

Training

Hyperparameters

Within the paper, there is not anything listed for any of the hyperparameters. Because of this, I will be using default values for almost all hyperparameters. These are broken out by each model type below:

Logistic Regression

- penalty: This is the penalty term, "l2" is the default value
- n_jobs: How many jobs the scikit-learn backend should use in training. "None" is the default value, and sets the number of jobs to the maximum available
- random_state: The random state of data shuffling. Used for deterministic results

XGBoost

- booster: Specifies the type of model to use. "gbtree" is the default value, and uses a tree-based model
- objective: Sets the objective of the learning task as well as the learning objective
- eval_metric: Evaluation metrics for validation data. The default is mapped to the training objective
- eta: Learning rate - the default value is 0.3
- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be. Default value is 0
- max_depth: Maximum depth of a tree. Default is 6
- min_child_weight: Minimum sum of instance weight (hessian) needed in a child. Default value is 1
- max_delta_step: Maximum delta step we allow each leaf output to be. Default value is 0
- subsample: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Default is 1
- colsample_bytree: Subsampling ratio of columns when constructing tree. Default is 1
- base_score: The initial prediction score of all instances, global bias
- silent: Makes training mode silent
- seed: Random state

SAPS-II

This is a predictability model, and the model scores are already calculated using the formula `1 / (1 + exp(- (-7.7631 + 0.0737*(sapsii) + 0.9971*(ln(sapsii + 1))) ))`, where `sapsii` is the SAPS-II score.

Computational requirements

Requirements for each model are very minimal due to datasets being limited and models being computationally minimal. Average runtime for training the XGBoost and Logistic Regression model take less than 3 seconds. The SAPS-II model probabilistic scores are created during data preprocessing. No GPus are required.

In [15]:
```
from sklearn.linear_model import LogisticRegression
import xgboost as xgb

# Set parameters for logistic regression
params_logreg = {}
params_logreg['penalty'] = 'l2'
params_logreg['n_jobs'] = None
params_logreg['random_state'] = RANDOM_STATE

# Create parameters for XGBoost
params_xgb = {}
params_xgb['booster'] = 'gbtree'
params_xgb['objective'] = 'binary:logistic'
params_xgb["eval_metric"] = "auc"
params_xgb['eta'] = 0.3
params_xgb['gamma'] = 0
params_xgb['max_depth'] = 6
params_xgb['min_child_weight'] = 1
params_xgb['max_delta_step'] = 0
params_xgb['subsample'] = 1
params_xgb['colsample_bytree'] = 1
params_xgb['base_score'] = 0.5
params_xgb['silent'] = 1
params_xgb['seed'] = RANDOM_STATE
```

In [16]:
```
# Create models
model_logreg = LogisticRegression(**params_logreg)
model_xgb = xgb.XGBRegressor(**params_xgb)
```

In [17]:
```
# Train models
model_logreg.fit(X_train_logreg, y_train_logreg)
model_xgb.fit(X_train_xgb, y_train_xgb)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the sh
ape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [19:03:49] WAR
NING: /workspace/src/learner.cc:742:
Parameters: { "silent" } are not used.

  warnings.warn(smsg, UserWarning)
```

Out[17]:

XGBRegressor

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
             device=None, early_stopping_rounds=None, enable_categorical=False,
             eta=0.3, eval_metric='auc', feature_types=None, gamma=0,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=0,
             max_depth=6, max_leaves=None, min_child_weight=1, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, ...)
```

Evaluation

For evaluation in this study, the authors used AUROC and DCA curves for comparisons. Additionally, we are going to use standard metrics, including:

- accuracy
- precision
- recall
- support
- F1
- confusion matrix

The reason for these metrics is they can provide a general comparison and results analysis for our 3 models, and can help us understand the performance of each across certain metrics. Understanding accuracy, precision, sensitivity, recall, etc shows different strengths and weaknesses in each model.

For our SAPS-II predictive model, a threshold must be defined for mortality classifications because this is a probabilistic model. The author's do not provide one, so we will assume that anything greater than or equal to probability 0.5 predicts "1", and anything less than 0.5 predicts "0".

In [18]:
```
from sklearn import metrics

# Get probabilistic predictions from each model and use these
predict_logreg = model_logreg.predict_proba(X_test_logreg)[:, 1]
predict_xgb = model_xgb.predict(X_test_xgb)
predict_sapsii = y_pred_sapsii

# Get classifications for each
predict_logreg_clf = np.where(predict_logreg >= 0.5, 1, 0)
predict_xgb_clf = np.where(predict_xgb >= 0.5, 1, 0)
predict_sapsii_clf = np.where(predict_sapsii >= 0.5, 1, 0)
```

In [19]:
```
# Get precision, recall, and macro fbeta scores
logreg_metrics = list(metrics.precision_recall_fscore_support(y_test_logreg, predict_log
xgb_metrics = list(metrics.precision_recall_fscore_support(y_test_xgb, predict_xgb_clf,
sapsii_metrics = list(metrics.precision_recall_fscore_support(y_true_sapsii, predict_sap

# Get accuracy scores
logreg_metrics.append(metrics.accuracy_score(y_test_logreg, predict_logreg_clf))
xgb_metrics.append(metrics.accuracy_score(y_test_xgb, predict_xgb_clf))
sapsii_metrics.append(metrics.accuracy_score(y_true_sapsii, predict_sapsii_clf))

# Get confusion matrix
cm_logreg = metrics.confusion_matrix(y_test_logreg, predict_logreg_clf)
cm_xgb = metrics.confusion_matrix(y_test_xgb, predict_xgb_clf)
cm_sapsii = metrics.confusion_matrix(y_true_sapsii, predict_sapsii_clf)
```

In [20]:
```
# Get ROC curves
fpr_logreg, tpr_logreg, _ = metrics.roc_curve(y_test_logreg, predict_logreg)
fpr_xgb, tpr_xgb, _ = metrics.roc_curve(y_test_xgb, predict_xgb)
fpr_sapsii, tpr_sapsii, _ = metrics.roc_curve(y_true_sapsii, y_pred_sapsii)

# Get AUC scores
logreg_metrics.append(metrics.roc_auc_score(y_test_logreg, predict_logreg))
xgb_metrics.append(metrics.roc_auc_score(y_test_xgb, predict_xgb))
sapsii_metrics.append(metrics.roc_auc_score(y_true_sapsii, y_pred_sapsii))
```

In [21]:
```
# Get the DCA for the models
```

```
# Calculate the net benefit based on a given threshold
def net_benefit(tp, fp, tn, fn, threshold):
    benefit = tp - (fp * threshold / (1 - threshold))
    total = tp + fn  # Total number of actual positives
    return benefit / total

# DCA
def decision_curve_analysis(y_true, y_prob):
    thresholds = np.linspace(0.01, 0.99, 100)
    net_benefits = []
    for threshold in thresholds:
      y_pred = np.where(y_prob >= threshold, 1, 0)
      tn, fp, fn, tp = metrics.confusion_matrix(y_true, y_pred).ravel()
      nb = net_benefit(tp, fp, tn, fn, threshold)
      net_benefits.append(nb)

    return thresholds, net_benefits
```

Results

Looking at results, the XGBoost model performs slightly better than the logistic regression model. Through research, the deterministic nature of XGBoost can usually lead to repeated results, although results have varied slightly in the past.

For `precision`, `recall`, `f1`, `accuracy`, and `AUC`, XGBoost performs the best, with `recall`, `f1`, and `accuracy` being a sizable difference, and `precision` and `AUC` being similar to logistic regression.

For the SAPS-II model, all metrics were signficicantly outperformed by the Logistic Regression model, and the XGBoost model. The ROC curve for the SAPS-II model also indicated inferior predictive performance compared to the other models.

Hypothesis and Results from original paper

From the original paper, the claim was that the XGBoost model can outperform Logistical Regression, and the standard probabilistic model SAPS-II. Additionally, I posed the 2 below hypotheses:

1. Hypothesis 1: The XGBoost algorithm can outperform traditional logistic regression and SAPS-II scoring models in predicting 30-day mortality among patients with sepsis-3 based on scoring metrics.

- Corresponding experiment: Create a logistic regression model, a SAPS-II dataset, and an XGBoost model, and use the features specified by L. et al. [1] for these models. Then, analyze the results of both models in predicting 30-day mortality due to sepsis-3 and compare.

1. Hypothesis 2: The set of features identified by the XGBoost model as significant predictors of 30-day mortality are significant compared to features not used

- Corresponding experiment: Perform several ablations, including dropping several features and adding several others, and compare post-training scores across different models to see which features area most important in each model's decisions, and how effective each model is.

From the results acquired from my sub-study, we see below across metrics, as well as AUROC and DCA graphs, that hypotheses 1 is not quite true. Logistic Regression and XGBoost perform similarly, while both outperform the SAPS-II model. However, in the original paper, as discussed below, the XGBoost model

does outperform both the Logistic Regression and SAPS-II model on all metrics. So, in this substudy, hypothesis 1 failed, but in the original study, hypothesis 1 held true.

For hypothesis 2, we will discuss this further with an ablation study below.

Experiments beyond the original paper

The original paper includes metrics solely focused on AUROC, DCA, and feature-analysis related metrics. However, there is quite a bit of value in looking at over-arching raw metrics, such as `precision`, `recall`, `f1`, `accuracy`, and `AUC`. Below, these metrics help us paint a story that clearly shows Logistic Regression outperforming both XGBoost and SAPS-II, while SAPS-II performs pretty poorly compared to the other models, and XGBoost is fairly "middle of the road". From the paper, we do not have these metrics available to us, but it would be interesting to see how each model would have performed with these summary statistics against each other. From the original study by L. et al. [1], the XGBoost model outperforms both other models - but would this hold true when we look at these summary statistics?

Table of results

In the table below, we can see the different metrics used compared against all 3 models. Across these metrics, we see the XGBoost and LogReg model performing fairly well, and the SAPS-II model performing poorly compared to the other models.

In [22]:
```
# Display results for metric scores as a dataframe
# Create dict out of these
data = {'metric': ['precision', 'recall', 'f1', 'accuracy', 'AUC'], 'LogReg': logreg_met
df_metrics = pd.DataFrame(data).round(4)
df_metrics.set_index('metric', drop=True, inplace=True)

# Display data
display(df_metrics)
```

metric	LogReg	XGBoost	SAPS-II
precision	0.7018	0.6610	0.4937
recall	0.6548	0.6540	0.4900
f1	0.6590	0.6569	0.4623
accuracy	0.7094	0.7000	0.5473
AUC	0.7281	0.7145	0.4885

ROC Curve

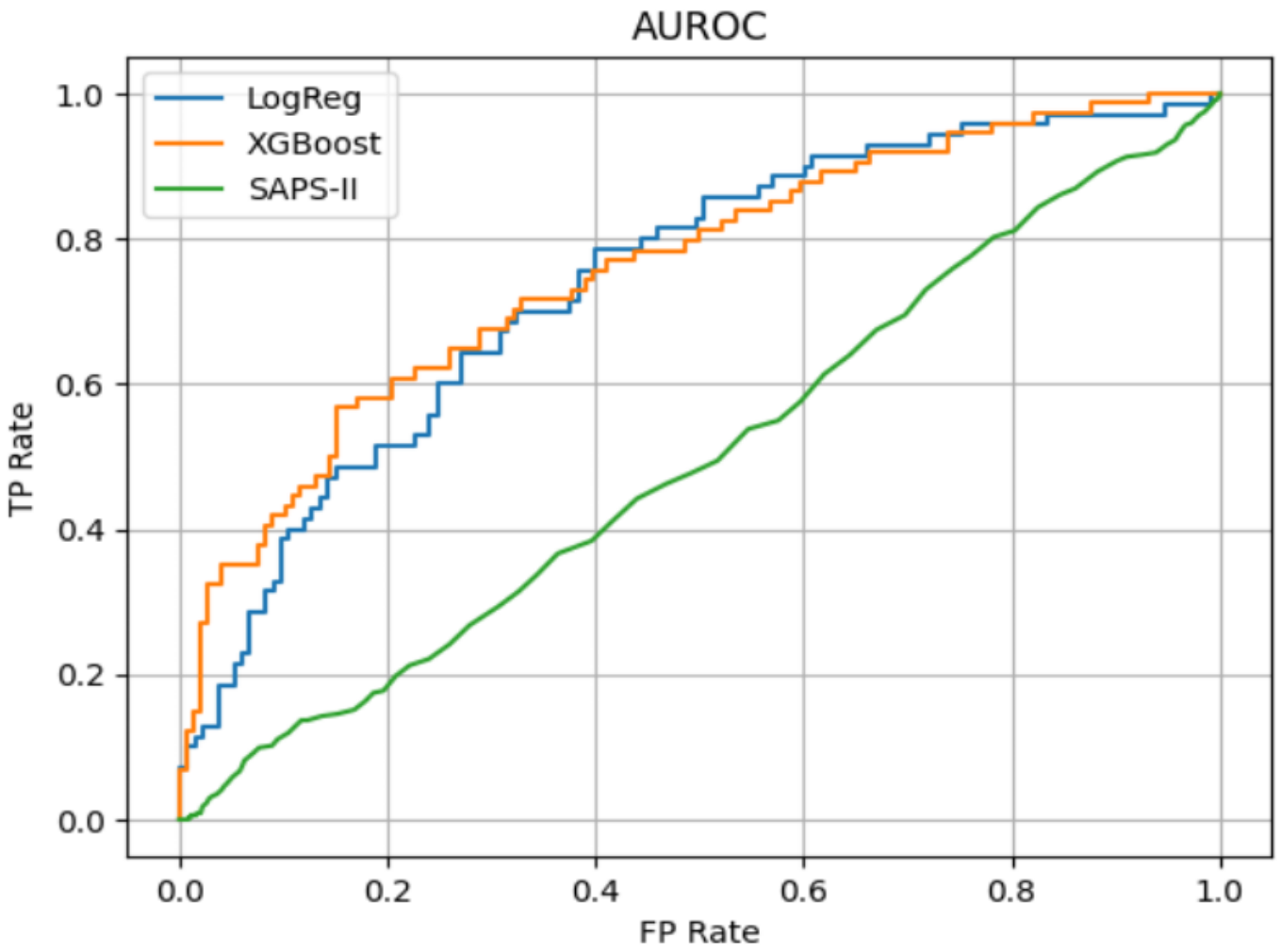
Looking at the ROC curves below for each model type, we can see the difference at each threshold between models. The XGBoost model indicates better performance at lower thresholds, but is evenly matched, and sometimes even outperformed, by the Logistic Regression model, at mid-to-higher thresholds. The SAPS-II model indicated average-to-poor predictive performance, providing a minimal ROC curve more closely matching a linear increase.

In [23]:
```
# Plot AUROC
# plt.figure()
```

```
# plt.plot(fpr_logreg, tpr_logreg, label='LogReg')
# plt.plot(fpr_xgb, tpr_xgb, label='XGBoost')
# plt.plot(fpr_sapsii, tpr_sapsii, label='SAPS-II')

# # Create plot
# plt.title('AUROC')
# plt.xlabel('FP Rate')
# plt.ylabel('TP Rate')
# plt.legend()
# plt.grid()
# plt.show()

# Show my image for faster processing time
show_image('my-roc', resize=True)
```



DCA Scores

DCA, or decision-curve-analysis, is used to compare clinical usefulness and net benefits between each model. The analysis helps in understanding tradeoffs between the benefit of true positive predictions, and the harm of false positive predictions within a clinical context.

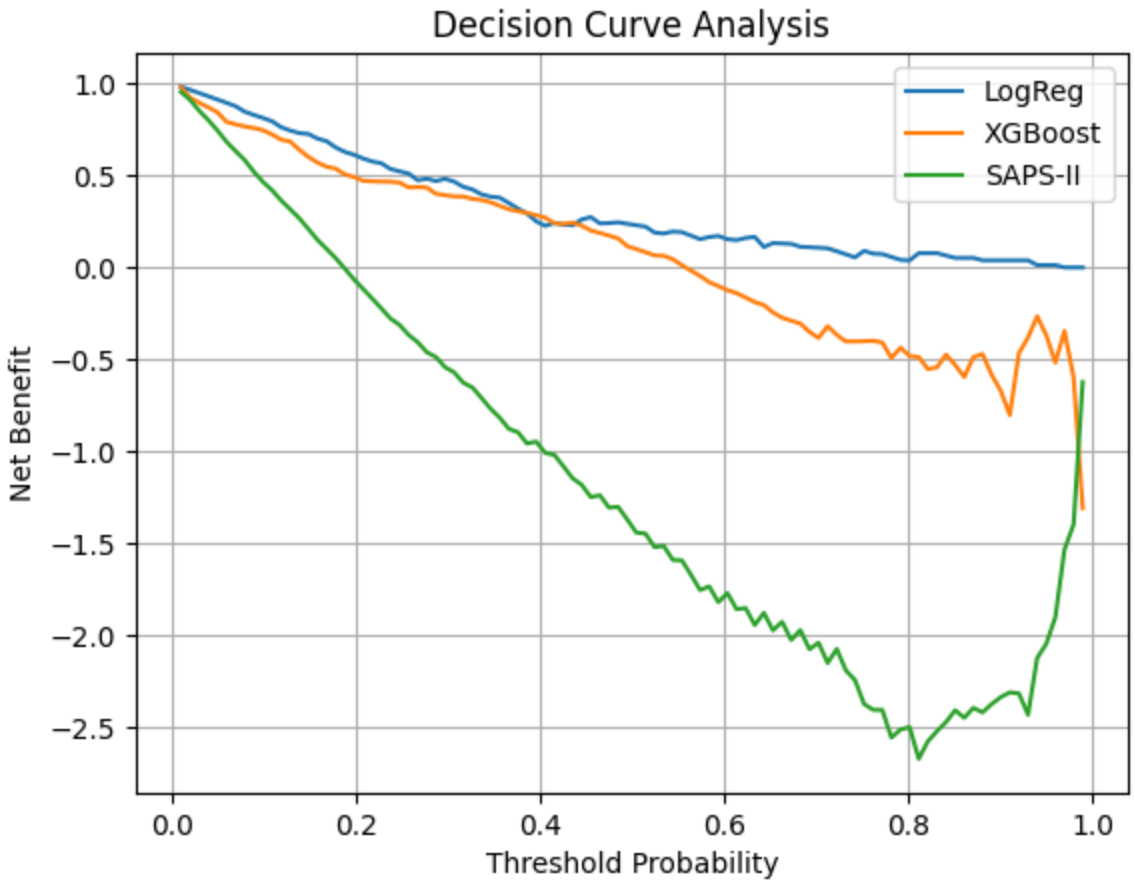
From the DCA curve below, we can see that the Logistic Regression model and XGBoost model perform similarly at each threshold, with some thresholds showing the XGBoost model outperforming. The SAPS-II graph has poor results here, showing low benefit at each threshold.

In [24]:
```
# Actually calculate the DCA scores
thresholds_logreg, net_benefits_logreg = decision_curve_analysis(y_test_logreg, predict_
```

```
thresholds_xgb, net_benefits_xgb = decision_curve_analysis(y_test_xgb, predict_xgb)
thresholds_sapsii, net_benefits_sapsii = decision_curve_analysis(y_true_sapsii, y_pred_s

plt.figure()
plt.plot(thresholds_logreg, net_benefits_logreg, label='LogReg')
plt.plot(thresholds_xgb, net_benefits_xgb, label='XGBoost')
plt.plot(thresholds_sapsii, net_benefits_sapsii, label='SAPS-II')

plt.xlabel('Threshold Probability')
plt.ylabel('Net Benefit')
plt.title('Decision Curve Analysis')
plt.legend()
plt.grid()
plt.show()
```



Model comparison

Comparing the models created in my subsample study to the actual full study, results performed similarly for the Logistic Regression and XGBoost models. For the SAPS-II model, results were not as good, but this could be due to multiple factors that will be brought up in the "Discussion" section.

The actual values for AUC, the ROC curve, and the DCA curves were not as impressive as the actual study - from the graphs below, I will compare the models I created, vs the models from the study. While trends were similar, results were simply better from the study. This could be due to multiple factors, such as the sample selection for my case study failed to capture certain features, or parameter selection differed due to unwritten differences the authors did not provide.

The authors do not provide other summary metrics such as accuracy, precision, recall, or F1-scores, but we can see similar trends matching in this study vs from L. et al. [1].

Comparing ROC curves

We can see similar differences in the Logistics Regression model vs the XGBoost model. The SAPS-II model provided outperforms from L. et al. [1] vs in this sub-study.

In [25]:
```
print('From L. et al. [1]')
show_image('graph-roc', resize=True)
print('From this study')
show_image('my-roc', resize=True)
```

```
From L. et al. [1]
```

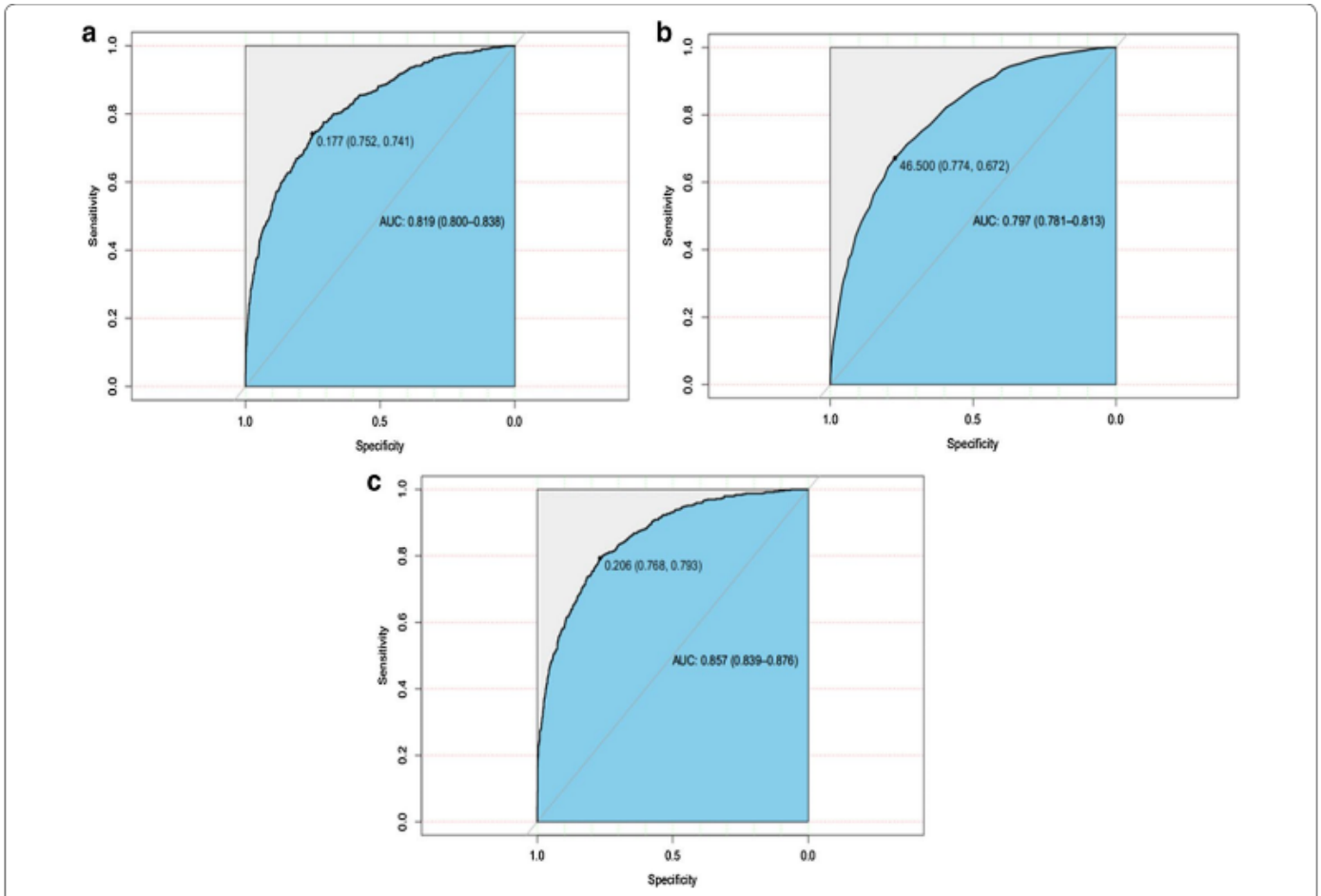
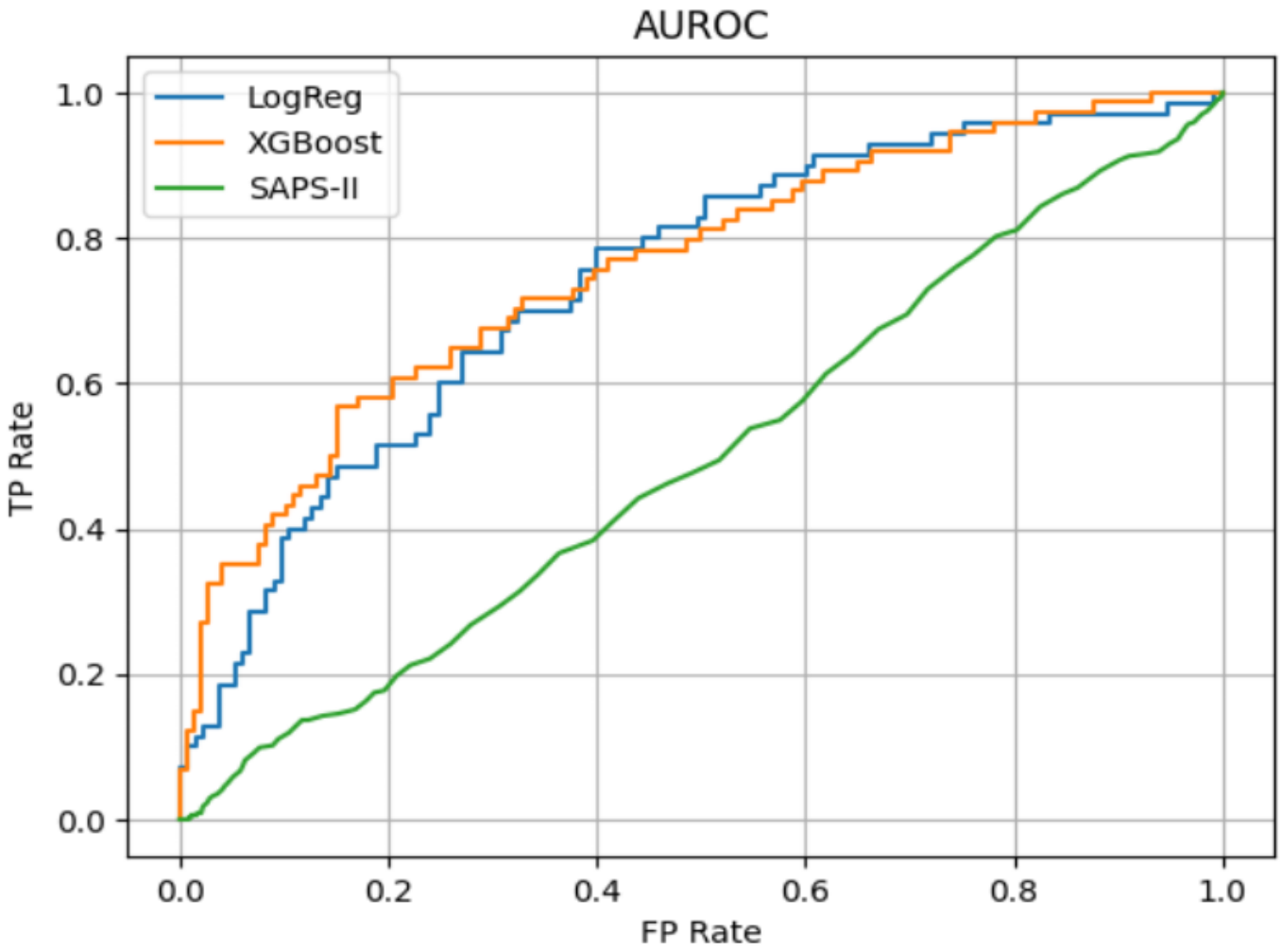


Fig. 4 The receiver operating characteristic (ROC) curves. **a** traditional logistic regression model, area under curves (AUC) is 0.819 [95% confidence interval (CI); 0.800–0.838]; **b** SAPS-II score model, AUC is 0.797 [0.781–0.813]; **c** XGboost model, AUC is 0.857 [0.839–0.876], the best performance of the models was the XGboost model

```
From this study
```



Comparing DCA curves

In [26]:
```
print('From L. et al. [1]')
show_image('graph-dca', resize=True)
print('From this study')
show_image('my-dca', resize=True)
```

```
From L. et al. [1]
```

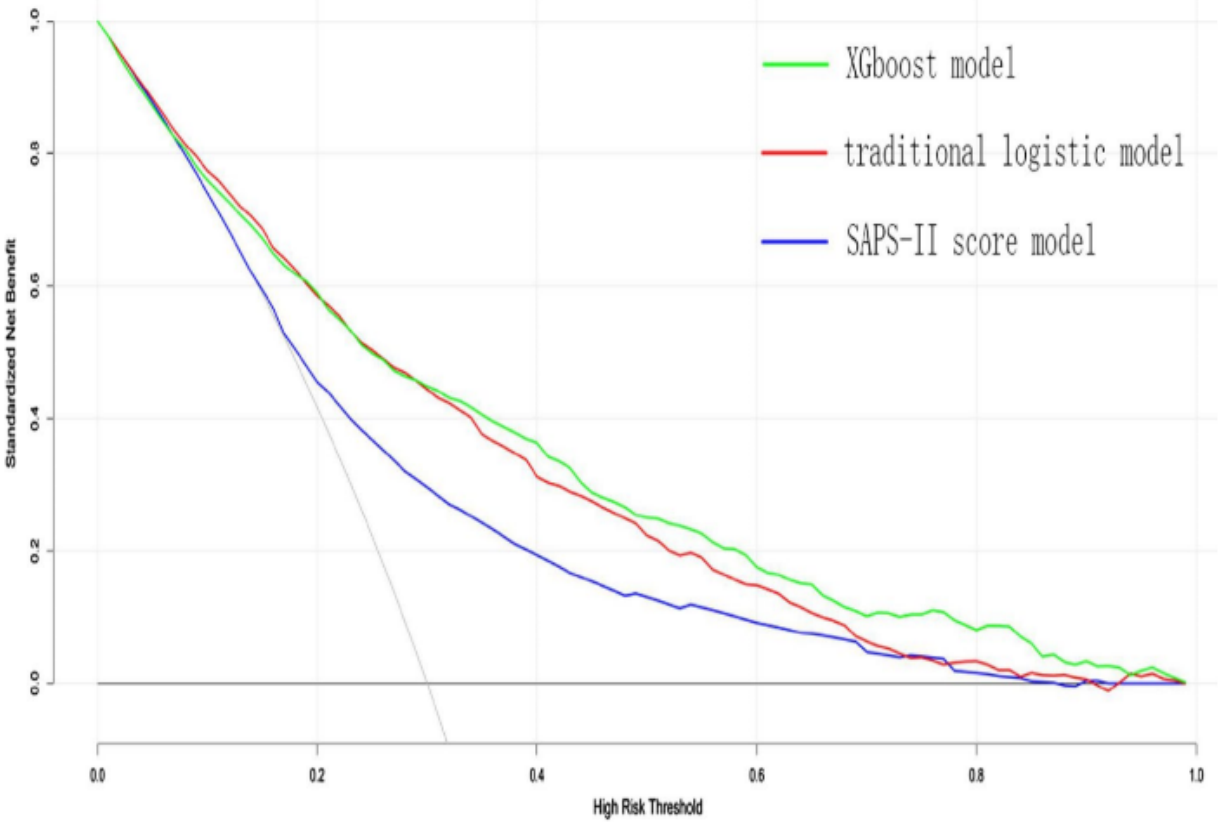
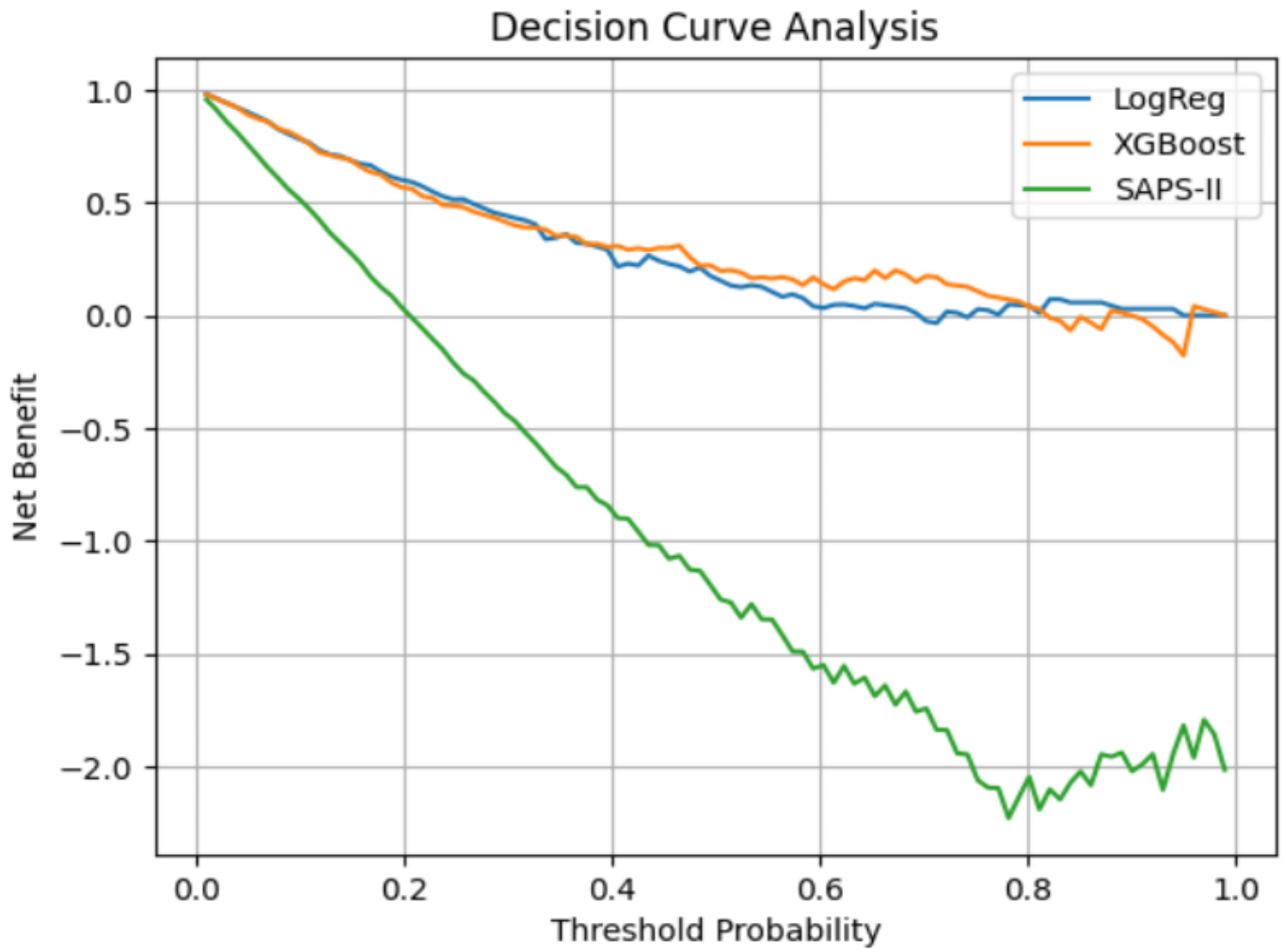


Fig. 5 Decision curve analysis (DCA) of the three prediction models. The net benefit curves for the three prognostic models are shown. *X*-axis indicates the threshold probability for critical care outcome and *Y*-axis indicates the net benefit. Solid green line = XGboost model, solid red line = traditional logistic model, solid blue line = SAPS-II score mode. The preferred model is the XGboost model, the net benefit of which was larger over the range of traditional logistic model and SAPS-II score model

From this study



Nomogram

The authors also provide a nomogram, which helps us see how important each feature is to sepsis mortality rates in this study. In the chart below, we can see how each feature is relevant for scoring in mortality predictions, and assign point values to different features. This is helpful in understanding how each feature plays a part in mortality scoring.

In [27]:
```
print('From L. et al. [1]')
show_image('graph-nomogram', resize=True)
```

```
From L. et al. [1]
```

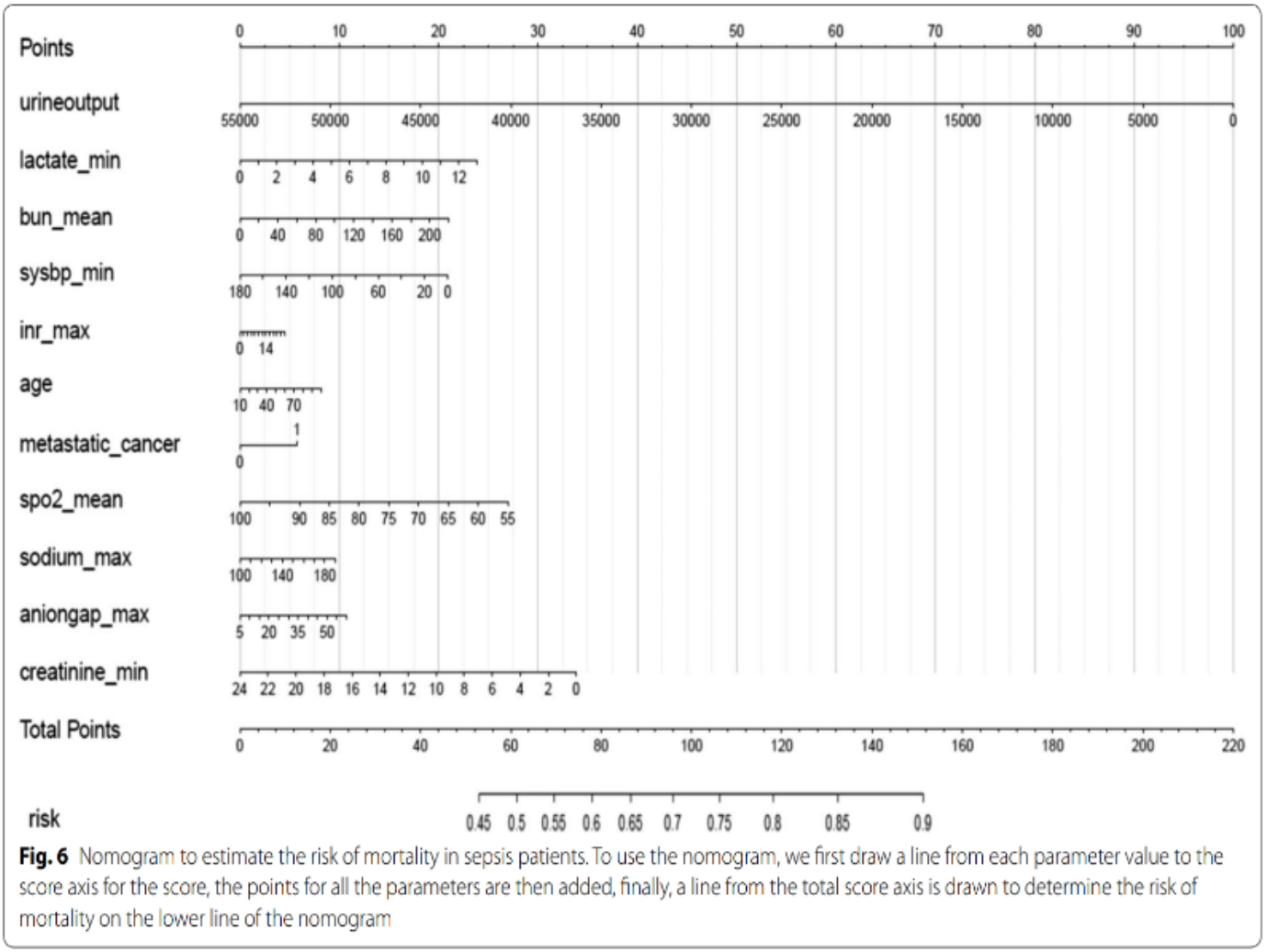


Fig. 6 Nomogram to estimate the risk of mortality in sepsis patients. To use the nomogram, we first draw a line from each parameter value to the score axis for the score, the points for all the parameters are then added, finally, a line from the total score axis is drawn to determine the risk of mortality on the lower line of the nomogram

Ablation Study

For this study by L. et al [1], there is a heavy emphasis placed on feature selection for each model type. In the image directly above this one, we can see the nomogram from L. et al to see how important the range of each feature is in overall predictive power of a model. Additionally, in Figure 1 of this notebook, we can see the features selected as well as related p-scores.

For this ablation study, I am taking a practical approach, and will be testing model results if the `age` and `sodium_max` columns are removed from the Logistical Regression model and XGBoost model - the SAPS-II model is an additive-probabilstic model, and falls apart somewhat if certain features are removed, and wasn't very impressive to begin with in this sub-study. So for this ablation section, we will focus only on the former two models.

Age is a strong predictor for any mortality study, so removing to try and compare results from other indicators could show potentially interesting results. For sodium_max, this is a feature that has a p-value < 0.001 for the Logistic Regression model, but for the XGBoost model, it is far less significant of a feature with a p-value = 0.03835.

Below, we can see directly how important each feature is to the model. For Logistic Regression, we can see that `age` and `sodium_max` are fairly important, and the same is true for the XGBoost model.

In [28]:
```
# Get feature importance for the Logistic Regression model
feature_importance_logreg_old = zip(X_train_logreg.columns, np.abs(model_logreg.coef_[0]
feature_importance_logreg_old = {k: v for k, v in feature_importance_logreg_old}
feature_importance_xgb_old = model_xgb.get_booster().get_score(importance_type='gain')
```

In [29]:
```
# Drop age columns from dataframes
X_train_logreg_abl = X_train_logreg.drop(['age', 'sodium_max'], axis=1)
X_test_logreg_abl = X_test_logreg.drop(['age', 'sodium_max'], axis=1)
X_train_xgb_abl = X_train_xgb.drop(['age', 'sodium_max'], axis=1)
X_test_xgb_abl = X_test_xgb.drop(['age', 'sodium_max'], axis=1)
```

In [30]:
```
# Create models
model_logreg_abl = LogisticRegression(**params_logreg)
model_xgb_abl = xgb.XGBRegressor(**params_xgb)

# Train models
model_logreg_abl.fit(X_train_logreg_abl, y_train_logreg)
model_xgb_abl.fit(X_train_xgb_abl, y_train_xgb)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the sh
ape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [19:03:56] WAR
NING: /workspace/src/learner.cc:742:
Parameters: { "silent" } are not used.

  warnings.warn(smsg, UserWarning)
```

Out[30]:

XGBRegressor

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
             device=None, early_stopping_rounds=None, enable_categorical=False,
             eta=0.3, eval_metric='auc', feature_types=None, gamma=0,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=0,
             max_depth=6, max_leaves=None, min_child_weight=1, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, ...)
```

Ablation Results

In [31]:
```
# Get probabilistic predictions from each model and use these
predict_logreg = model_logreg_abl.predict_proba(X_test_logreg_abl)[:, 1]
predict_xgb = model_xgb_abl.predict(X_test_xgb_abl)

# Get classifications for each
predict_logreg_clf = np.where(predict_logreg >= 0.5, 1, 0)
predict_xgb_clf = np.where(predict_xgb >= 0.5, 1, 0)
```

In [32]:
```
# Get precision, recall, and macro fbeta scores
logreg_abl_metrics = list(metrics.precision_recall_fscore_support(y_test_logreg, predict
xgb_abl_metrics = list(metrics.precision_recall_fscore_support(y_test_xgb, predict_xgb_c

# Get accuracy scores
logreg_abl_metrics.append(metrics.accuracy_score(y_test_logreg, predict_logreg_clf))
xgb_abl_metrics.append(metrics.accuracy_score(y_test_xgb, predict_xgb_clf))

# Get confusion matrix
cm_abl_logreg = metrics.confusion_matrix(y_test_logreg, predict_logreg_clf)
cm_abl_xgb = metrics.confusion_matrix(y_test_xgb, predict_xgb_clf)

# Get ROC curves
```

```
fpr_abl_logreg, tpr_abl_logreg, _ = metrics.roc_curve(y_test_logreg, predict_logreg)
fpr_abl_xgb, tpr_abl_xgb, _ = metrics.roc_curve(y_test_xgb, predict_xgb)

# Get AUC scores
logreg_abl_metrics.append(metrics.roc_auc_score(y_test_logreg, predict_logreg))
xgb_abl_metrics.append(metrics.roc_auc_score(y_test_xgb, predict_xgb))
```

In [33]:
```
# Display results for metric scores as a dataframe
# Create dict out of these
data = {'metric': ['precision', 'recall', 'f1', 'accuracy', 'AUC'],
        'LogReg': logreg_metrics, 'LogReg_Ablation': logreg_abl_metrics,
        'XGBoost': xgb_metrics, 'XGBoost_Ablation': xgb_abl_metrics,
        'SAPS-II': sapsii_metrics}
df_metrics = pd.DataFrame(data).round(4)
df_metrics.set_index('metric', drop=True, inplace=True)

# Display data
display(df_metrics)
```

metric	LogReg	LogReg_Ablation	XGBoost	XGBoost_Ablation	SAPS-II
precision	0.7018	0.6905	0.6610	0.6484	0.4937
recall	0.6548	0.6418	0.6540	0.6372	0.4900
f1	0.6590	0.6443	0.6569	0.6410	0.4623
accuracy	0.7094	0.6995	0.7000	0.6909	0.5473
AUC	0.7281	0.7323	0.7145	0.7201	0.4885

From the above table, we can see actual *improvements* from the ablations! This could be a result of several different possibilities - one could be that the features removed were actually causing more noise and making pattern detection harder for the models. Another possibility is that reducing the number of dimensions helps the models focus more clearly on other feature splits.

Regardless, we can see significant improvement in the XGBoost model across the board - every single metric improved for this mdoel. For the linear regression model, results were fairly similar to the model without any features missing, showing non-significant results.

Below, we can see the new feature importance after features have been removed. From the "Diff" column, we can see for Logistic Regression that `bun_min`, `bun_max`, `heartrate_min`, `heartrate_mean` `sysbp_min`, `tempc_min`, and `spo2_mean` all had large changes in feature importance from the ablation study.

For the XGBoost model, we can see `urineoutput` and `metastic_cancer` had larger impacts on feature selection after performing the ablation.

In [34]:
```
# Get feature importance for the Logistic Regression model
feature_importance_logreg = zip(X_train_logreg_abl.columns, np.abs(model_logreg_abl.coef
feature_importance_logreg = {k: v for k, v in feature_importance_logreg}
feature_importance_xgb = model_xgb_abl.get_booster().get_score(importance_type='gain')
```

In [35]:
```
# Compare each key and see the difference for LogReg
max_length = max([len(str(k)) for k in feature_importance_logreg.keys()] + [len(str(k))
print('Logistic Regression')
print("Key\t\t\tNormal\tAbl.\tDiff")
print('------------')
for k, v in feature_importance_logreg.items():
```

```
  old_val = feature_importance_logreg_old[k]
  print("{}\t{:.4f}\t{:.4f}\t{:.4f}".format(k.ljust(max_length), v, old_val, v-old_val))


# Compare each key and see the difference for XGBoost
print('\n\nXGBoost')
print("Key\t\t\tNormal\tAbl.\tDiff")
print('------------')
for k, v in feature_importance_xgb.items():
  old_val = feature_importance_xgb_old[k]
  print("{}\t{:.4f}\t{:.4f}\t{:.4f}".format(k.ljust(max_length), v, old_val, v-old_val))
```

```
Logistic Regression
Key                     Normal  Abl.    Diff
------------
sofa                    2.1553  2.1704  -0.0151
aniongap_min            1.4000  1.4062  -0.0063
aniongap_max            0.2809  0.3206  -0.0397
creatinine_min          1.2918  1.2603  0.0315
chloride_min            0.2303  0.3602  -0.1299
hematocrit_min          0.9021  0.8711  0.0310
hemoglobin_min          0.7408  0.7124  0.0283
hemoglobin_max          0.6173  0.6341  -0.0168
lactate_min             2.7440  2.7394  0.0046
potassium_min           1.6426  1.6877  -0.0451
bun_min                 1.6323  1.6241  0.0082
bun_max                 0.0107  0.0252  -0.0145
wbc_min                 0.6759  0.6638  0.0121
wbc_max                 0.3559  0.3638  -0.0079
heartrate_min           0.8797  0.8817  -0.0020
heartrate_mean          1.1568  1.1667  -0.0099
sysbp_min               1.4146  1.3701  0.0445
meanbp_min              0.7200  0.7051  0.0149
resprate_mean           0.8502  0.8374  0.0128
tempc_min               0.6645  0.6804  -0.0159
tempc_max               2.3776  2.3767  0.0010
spo2_mean               1.7387  1.7465  -0.0078
diabetes                0.4086  0.4257  -0.0171
mechvent                0.7415  0.7388  0.0028


XGBoost
Key                     Normal  Abl.    Diff
------------
urineoutput             2.1599  2.3094  -0.1495
lactate_min             1.6612  1.7759  -0.1147
bun_mean                1.4006  1.4673  -0.0667
metastatic_cancer       1.1136  0.8119  0.3017
inr_max                 1.2870  1.4110  -0.1240
aniongap_max            1.4510  1.4747  -0.0236
creatinine_min          1.2593  1.2683  -0.0090
spo2_mean               1.4553  1.5129  -0.0576
```

Discussion

After reviewing the hypothesis in the paper, we can draw a few tentative conclusions:

1. The XGBoost model has the potential to outperform standard models in predicting 30-day mortality for sepsis patients.
2. Certain features have significant important in creating these mortality predictions, while others are less important or unnecessary altogether.

Reproducability

The study by L. et al. [1] was reproducable to an extent - if a decision was made to invest more into creating an accurate data representation, I believe the study would be fully reproducable, consdering the study by L. et al. [1] uses the MIMIC-III dataset. However, due to the size of the dataset, and the hardware constraints coupled with timing constraints, a sample was chosen for this study that provided somewhat different results, although similar trends were seen.

The SAPS-II results were fairly different from what was seen by L. et al. [1]. This could be due to multiple factors - a lot of assumptions had to be made regarding all models because there were no discussions on model creation, parameter selection, architectures chosen, probabilistic functions, preprocessing steps, etc. Because of this, the models seen in this notebook could be significantly different from the ones used by L. et al. [1]. That being said, with feature selection done already by the authors, we can at least get somewhat comparable results to what the authors had.

What was easy

Model creation was actually fairly easy from this study - most of this is due to the fact that the models were not specified in any way, other than name. Because of this, minimal assumptions were made regarding model architecture, and training requirements were easy. Additionally, due to taking a sample of MIMIC-III data as opposed to the entire dataset, training times and requirements were minimal.

Feature selection was straightforward as well - the authors described their methods for selecting features, and creating datasets based on this left minimal assumptions for recreating this study.

What was not easy

Trying to understand what types of model choices the authors made was difficult - while actually creating and training the models was not difficult, trying to create an accurate replica was. Additionally, preprocessing raw MIMIC-III data was a difficult task, as features were called out by L. et al. [1], but having minimally interacted with MIMIC-III data, trying to scrape together the correct features and understand these features took most of the time involved in this study.

Additionally, creating an accurate sample of the data, as well as hosting it in an accessable format, took planning and trial-and-error as well. Some databases and flavors of SQL do not support necessary date functions, or don't have portability between a simple notebook that can be ran anywhere.

Suggestions for the author

The study was an excellent read on understanding mortality in ICU settings, traditional approaches on predicting 30-day mortality, and potential improvements to these predictions. What I would suggest is perhaps talking more on technical choices after feature selection - why use MIMIC-III, for example? Why the XGBoost model, instead of a random forest or some other ensemble method? With XGBoost, what hyperparameters were selected, such as learning rate, epochs, batch size, tree depth, child nodes, etc. XGBoost has a lot of customizability, which leads to some models training on the same dataset with drastically different results. Understanding the author's choices here would be helpful to any reader.

Additionally, more baseline evaluation metrics would be helpful - we can see how each feature is relevant repeatedly, and see the ROC graphs and AUC scores, and the DCA analysis and nomogram and CDC graph. However, how do these models perform against one another within different metrics? Trying to

understand F1-scores from my sub-study vs the actual study is somewhat fruitless, as there is no F1-score mentioned.

Lastly, what thresholds were used for probabilistic predictions? For SAPS-II, for example, mortality probabilities are given, but the threshold decided for binary classification is entirely up to the author. One can assume that 0.5 is standard and was used in this case, but an explicit callout for this would change results drasitically.

References

1. Hou, N., Li, M., He, L. et al. Predicting 30-days mortality for MIMIC-III patients with sepsis-3: a machine learning approach using XGboost. J Transl Med 18, 462 (2020). https://doi.org/10.1186/s12967-020-02620-5