Capstone Project 2: Early Prediction of Sepsis from Clinical Data Final Report

Introduction

Sepsis is a highly dangerous condition where the body's response to infection causes further damage, from organ system damage, to septic shock, to death. According to the CDC, approximately 1.7 million people develop sepsis, while 270,000 die from it in the USA, each year. Internationally, the WHO estimates that 30 million people suffer from sepsis, with a death rate of 6 million each year. In addition, the costs of diagnosing and treating sepsis in US hospitals alone costs up to $24 billion dollars annually. There is a very strong need for new methods to detect sepsis early and accurately. This is a significant need for hospitals and patients alike who are greatly affected by this disease. Creating a precise and accurate model that can predict early onset of sepsis would save lives, and hospitals from expending valuable resources. Predicting it too early consumes limited hospital resources, while predicting it too late makes improving sepsis outcomes incredibly challenging.

To create such a model, I used data made available through Physionet.org. Physionet is a platform that provides publically available datasets related to clinical and medical data. These datasets are MESSY and have not been preprocessed for typical statistical analysis and machine learning applications.

The data consisted of 5000 PSV files, each containing a patient's clinical data starting at the patient's admission to the ICU. The clinical data was collected hourly, represented row-wise, with features including heart rate, temperature, white blood cell count, and many more.

I looked to examine, based on the provided data, whether a patient would develop sepsis. My approach would thus be to wrangle the data into a single dataframe preprocessed appropriately for Supervised Learning. Because this was a Time-Series problem, Time-Series Analysis and resources were implemented through adding features that appropriately sought significant trends within features.

My deliverables are the code/algorithms to access my models, as well as a paper and slides summarizing my thought process throughout this project.

Exploratory Data Analysis/Data Cleansing

The initial steps I took for exploring the data were as follows:

- I checked to see how many of the 5000 patients ended up testing positive for Sepsis
- I created a visual representation of the hour at which these patients were diagnosed with Sepsis, with t = 0 being when the patient was admitted to the ICU
- I examined the dataset to look for missing values

Each patient file included 40 features that could be put into three major categories. Vital Signs, Laboratory Values, and Demographics. Below, I show the categories and their corresponding features:

- **Vital signs**
  - HR: Heart Rate (beats per minute)
  - O2Sat: Pulse oximetry, (%)
  - Temp: Temperature (Deg C)
  - SBP: Systolic BP (mm Hg)
  - MAP: Mean arterial pressure (mm Hg)
  - DBP: Diastolic BP (mm Hg)
  - Resp: Respiration rate (breaths per minute)
  - EtCO2: End tidal carbon dioxide (mm Hg)
- **Laboratory Values**
  - BaseExcess: Measure of excess bicarbonate, (mmol/L)
  - HCO3: Bicarbonate, (mmol/L)
  - FiO2: Fraction of inspired oxygen, (%)
  - pH: potential Hydrogen
  - PaCO2: Partial pressure of carbon dioxide from arterial blood, (mm Hg)
  - SaO2: Oxygen saturation from arterial blood, (%)
  - AST: Aspartate transaminase, (IU/L)
  - BUN: Blood urea nitrogen, (mg/dL)
  - Alkalinephos: Alkaline phosphatase, (IU/L)
  - Calcium: (mg/dL)
  - Chloride: (mmol/L)
  - Creatinine: (mg/dL)
  - Bilirubin_direct: (mg/dL)
  - Glucose: Serum glucose, (mg/dL)
  - Lactate: Lactic acid, (mg/dL)
  - Magnesium: (mmol/dL)
  - Phosphate: (mg/dL)
  - Potassium: (mmol/L)
  - Bilirubin_total: Total bilirubin, (mg/dL)
  - TroponinI: Troponin I, (ng/mL)
  - Hct: Hematocrit, (%)
  - Hgb: Hemoglobin, (g/dL)
  - PTT: partial thromboplastin time, (seconds)
  - WBC: Leukocyte count (count*10^3/µL)

    - Fibrinogen: (mg/dL)
    - Platelets: (count*10^3/μL)
- **Demographics**
    - Age: Years (100 for patients 90 or above)
    - Gender: Female (0) or Male (1)
    - Unit1: Administrative identifier for ICU unit (MICU)
    - Unit2: Administrative identifier for ICU unit (SICU)
    - HospAdmTime: Hours between hospital admit and ICU admit
    - ICULOS: ICU length-of-stay (hours since ICU admit)

In addition, for each row for each patient, the binary feature, 'SepsisLabel', indicated whether a patient had been diagnosed with Sepsis, or not. In other words, patients who did not get diagnosed with Sepsis have this column filled only with 0s, while those who did, start out with 0s, which eventually convert to 1s. Knowing this, I created a feature, 'GetsSepsis', indicating whether a patient was eventually diagnosed with Sepsis.

From this I saw that there were only 279 Sepsis-positive patients, making this a highly imbalanced dataset. This meant it would be imperative to monitor not just Accuracy from our models, but also Precision, Recall, and Area Under the ROC curve (AUROC).

The number of rows for each patient file were not uniform. Some contained as few as 20 rows, while others contained as many as 200. This simply showed that some patients had longer stays in the ICU than others.

Among the Positive patients, I wanted to see how quickly they were able to be diagnosed. Therefore I made a bar plot showing the number of patients who were diagnosed within their first day in the ICU, those within the second day, and so on. The results from this can be seen below in **Figure 1**:

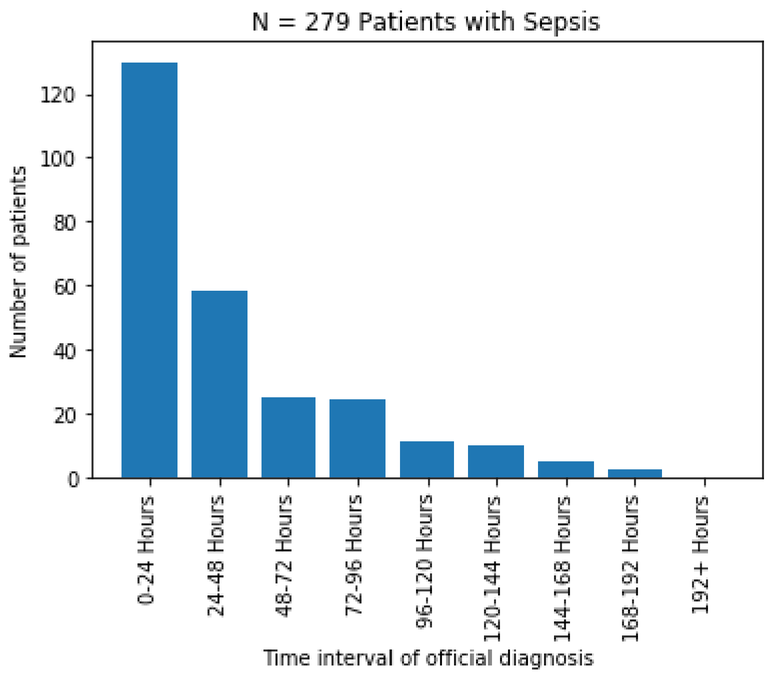


Figure 1: Duration in ICU before official diagnosis across all Sepsis patients

We see that only a little over 120 patients get diagnosed within the first 24 hours, while others didn't receive the diagnosis until days later. This distribution elucidated the fact that of the patients who were eventually diagnosed with Sepsis, approximately half were diagnosed within the first 24 hours. Interestingly, of the remaining Positive patients, half of those patients received their diagnosis in the next 24 hours, and this trend continued onward with the exception that the third and fourth days were consistent with each other, as well as the fifth and sixth days.

While there were no missing values in the 'SepsisLabel' features, they certainly occurred and needed to be addressed in other columns. As mentioned previously, these patient files were messy, due to the inconsistent missing values present among them. I handled the missing values in the Vital Sign columns by filling with the mean, because these features were more commonly collected hourly, meaning less frequent missing values, so such imputation methods like filling with the mean would suffice. The missing values in the Laboratory Value columns tended to be much more abundant. In many cases, only one or two values were real. This was likely due to the Lab Values being collected much less frequently at the hospitals where the data originated. Therefore, the missing values from these columns were handled through forward and backward filling.

To visualize the effects of my imputation techniques, I created histograms with both the original and imputed datasets overlapping each other, for each feature from the vital signs and lab values, the resulting distributions maintain their normal shape and do not cause major skews, which is a good indication that this can be an effective method for filling the missing values. Examples of the generated overlapping distributions can be seen in **Figures 2 and 3**.

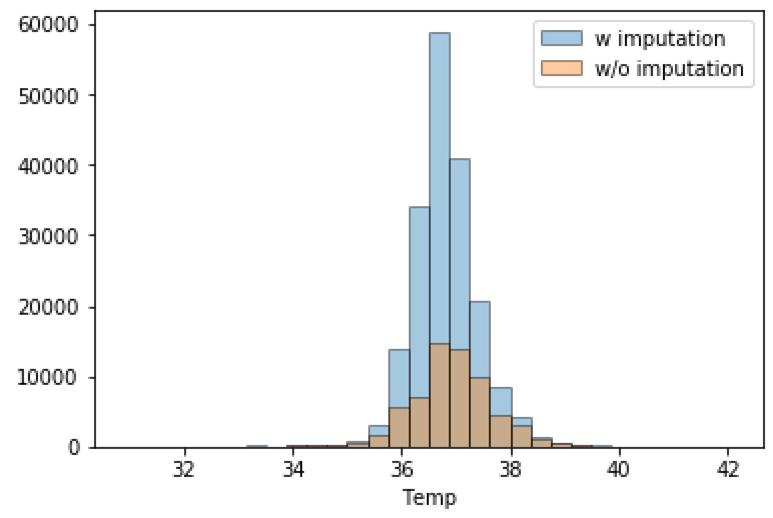


Figure 2: Distribution of all values from the Temperature feature

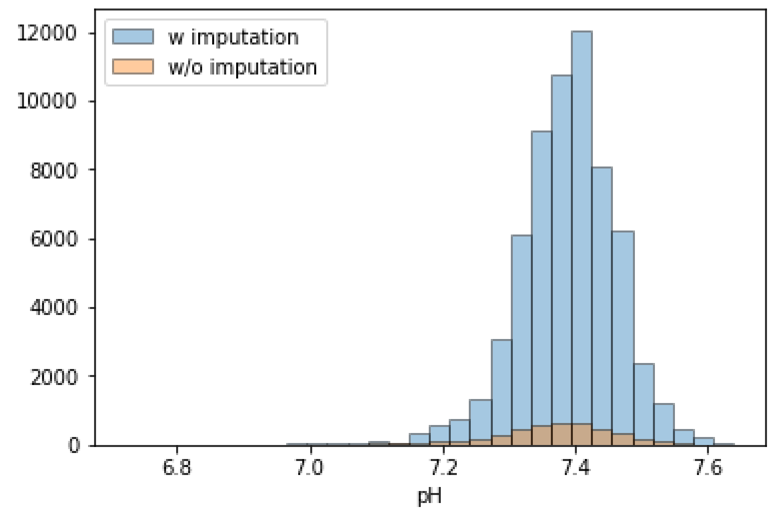


Figure 3: Distribution of all values from the pH feature

Since the PSV files were divided, I wanted to concatenate them all into a single dataframe. This facilitated parsing through the different patients, specifically by adding an additional column (patientID), whose value would be unique to each individual patient. Per individual patient, the only missing values left were those initially missing in columns in their entirety. An example would be a patient whose measure of excess bicarbonate (BaseExcess) was not provided.

I aimed to use a **grouped forward selection** process to observe the effects of features on model performance, and wanted to start with models only taking in the Vital Signs. With that in mind, I wanted to observe the number of real values present in the Vital Signs columns shown in **Figure 4**. In addition, I wanted to show, per feature, the number of patients that have columns entirely missing shown in **Figure 5**.

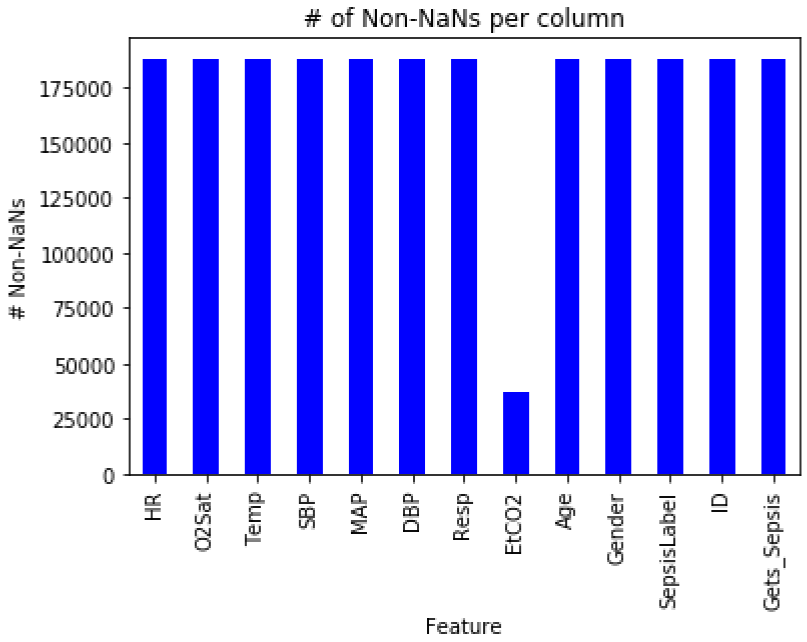
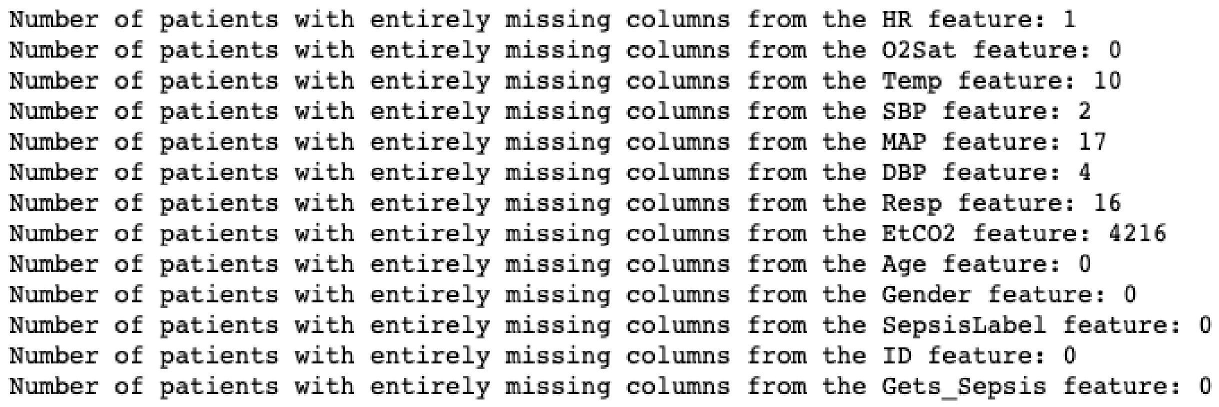


Figure 4: Number of real values, per Vital Sign feature, across all patients



```
Number of patients with entirely missing columns from the HR feature: 1
Number of patients with entirely missing columns from the O2Sat feature: 0
Number of patients with entirely missing columns from the Temp feature: 10
Number of patients with entirely missing columns from the SBP feature: 2
Number of patients with entirely missing columns from the MAP feature: 17
Number of patients with entirely missing columns from the DBP feature: 4
Number of patients with entirely missing columns from the Resp feature: 16
Number of patients with entirely missing columns from the EtCO2 feature: 4216
Number of patients with entirely missing columns from the Age feature: 0
Number of patients with entirely missing columns from the Gender feature: 0
Number of patients with entirely missing columns from the SepsisLabel feature: 0
Number of patients with entirely missing columns from the ID feature: 0
Number of patients with entirely missing columns from the Gets_Sepsis feature: 0
```

Figure 5: Number of patients with entirely empty columns from all Vital Sign features

From these figures, we can see that despite imputation methods, 84% (4216/5000) of patients have fully empty columns from the EtCO2 feature. Moving forward, I removed the EtCO2 column from the dataset, when creating the first round of models. That way, when I used the **.dropna()** command of the dataframe, fewer patients would be removed and valuable data could be preserved. By doing this, I preserved 4956 patients, and ended up, at this point, with a dataframe with no missing values, ready to be used for binary classification models.

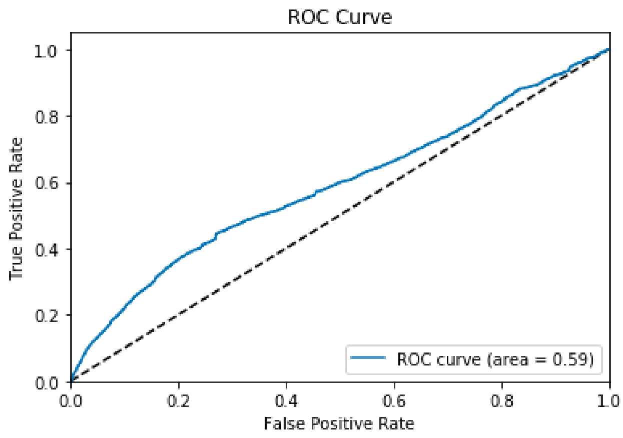
Machine Learning: Starting with the Vital Signs

I then created a model for binary classification, in order to predict, by timestamp, whether a patient WILL or WILL NOT be diagnosed with Sepsis.

To facilitate testing many models, the function '**make_model**' was created. It took in the features, the predictor column, the selected model, and the parameter grid I covered, as this function incorporated **GridSearchCV** to apply cross-validation and hyperparameter tuning. In addition, the parameter, "**scoring = 'roc_auc'**", was specified so that the GridSearchCV object optimized for the model with the best AUROC. As I saw in the initial EDA, only 279 out of 5000 patients were diagnosed with Sepsis, making this an imbalanced dataset. This made the AUROC metric a very valuable metric to optimize as it helped distinguish the binary classes.

When splitting the data into training and testing sets, it needed to be done such that rows from the same patient file did not get split into both sets, much as we would not want the same row to show up in both sets in order to prevent overfitting. Thus, I used **StratifiedKFold** from sklearn to ensure that did not happen. StratifiedKFold split datasets based on their indices, which was essentially what I want to do, but with their IDs instead of indices.

Many types of supervised learning models were tried out, including Logistic Regression, K-Nearest Neighbors, Support Vector Machines, Decision Trees, and Random Forests.



```
Tuned Model Parameter: {'C': 1000000.0, 'penalty': 'l1'}
Tuned AUROC 0.6145563034008895
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     34099
           1       0.20      0.00      0.00      3218

    accuracy                           0.91     37317
   macro avg       0.56      0.50      0.48     37317
weighted avg       0.85      0.91      0.87     37317
```

Figure 6: Model Parameters of Logistic Regression Model with only Vital Sign features

Figure 6 is representative of all the models attempted with this narrow dataset. Most of them output AUROCs near the value of 0.5. The features at this point did not distinguish the two outcomes (Sepsis vs No Sepsis) very well.

Machine Learning: Adding the Laboratory Values

The next step was to include the Laboratory Value columns into the dataset and models. Firstly however, I observed the number of real and missing values I was dealing with across all columns. This analysis was similar to the previous bar graph that revealed the high number of missing values from the EtCO2 column, but more comprehensive. Again, this was after initial imputation techniques.

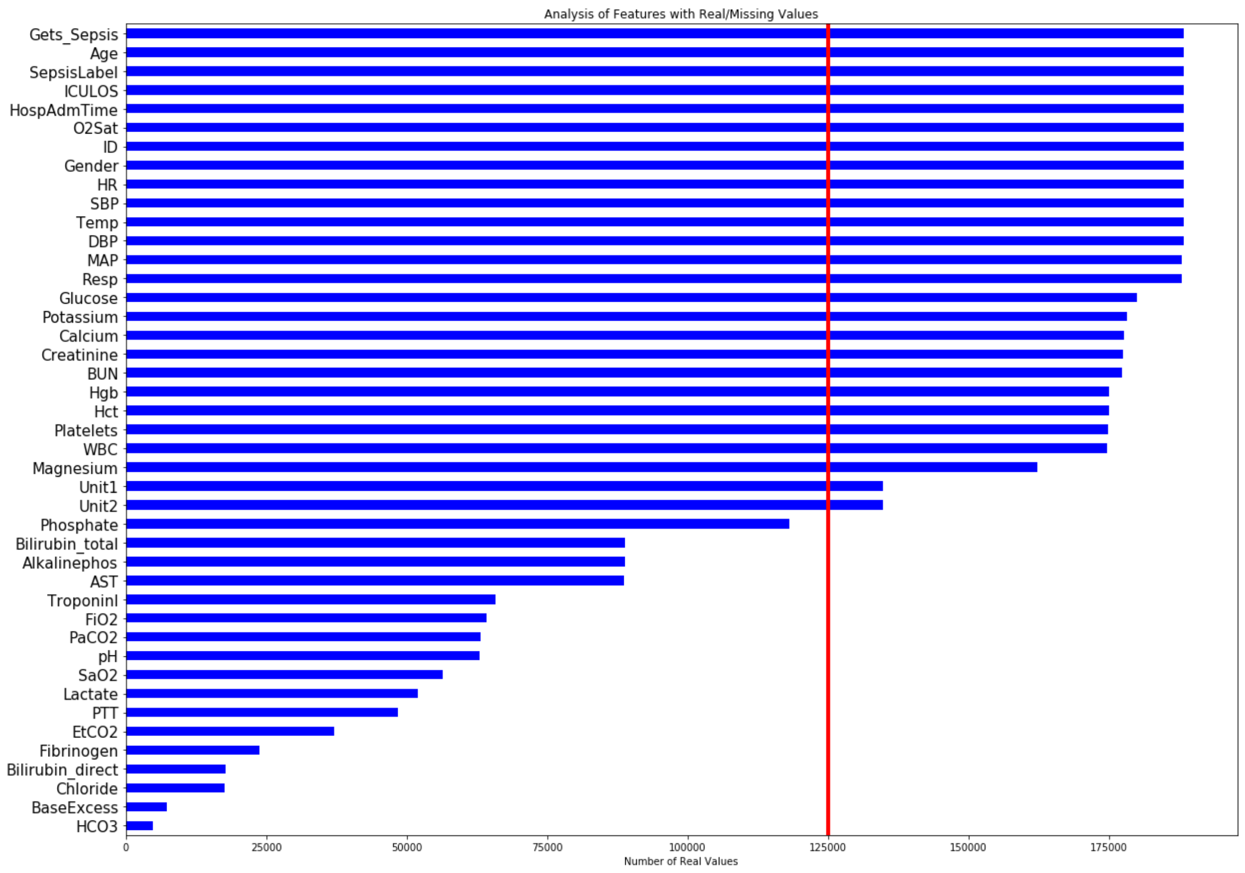
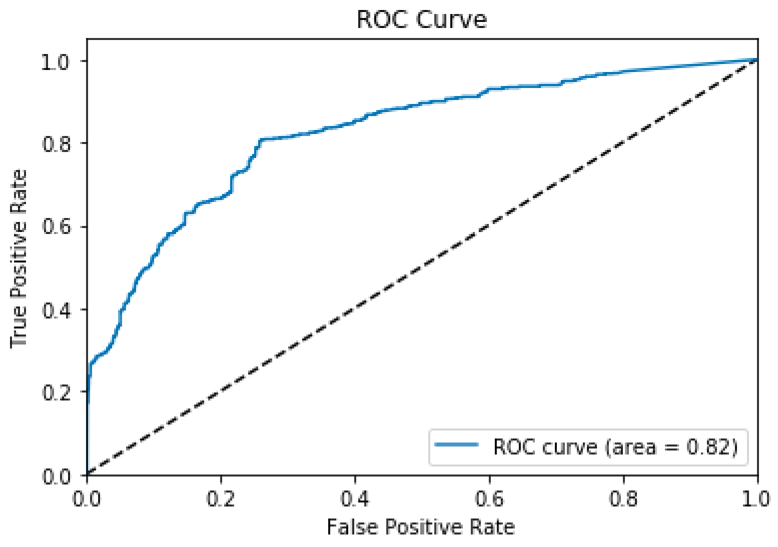
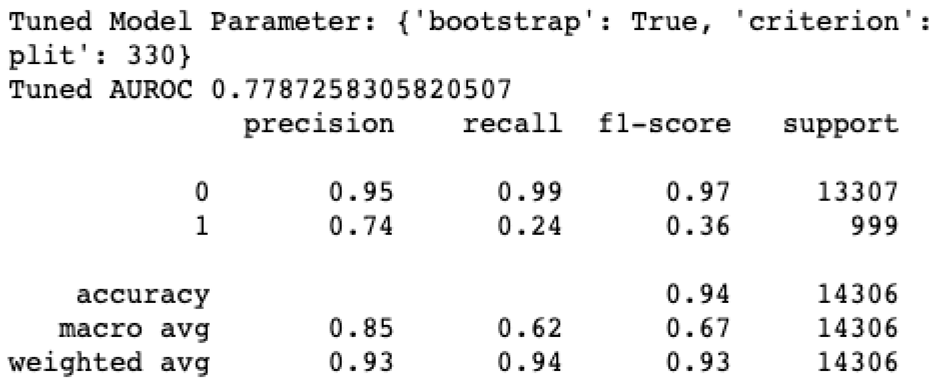


Figure 7: Number of real values, across all features and patients

While it would have been ideal ncorporate all available features, features with too many missing values could inhibit the number of patients that could be used in our models. This is why as part of including the Laboratory Value columns, the next dataset would still not include high-NaN columns. I selected the cut-off point to be at 125,000 rows (out of the 188,453 total). Features that did not have at least 125,000 rows worth of real values were not included. This included all the features listed on the y-axis of **Figure 7**, from ‘Phosphate’ and below.

As was done previously, the **make_model** function was used with the new dataset, applying the same range of different supervised learning methods.





```
Tuned Model Parameter: {'bootstrap': True, 'criterion':
plit': 330}
Tuned AUROC 0.7787258305820507
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     13307
           1       0.74      0.24      0.36       999

    accuracy                           0.94     14306
   macro avg       0.85      0.62      0.67     14306
weighted avg       0.93      0.94      0.93     14306
```

Figure 8: Model Parameters of Random Forest Model with the Vital Sign, Lab Value, and Demographic features

Figure 8 shown above was also representative of all the models attempted with this specific dataset. It showed that with a Random Forest Model implemented with certain specified parameters ({**bootstrap**: True, **criterion**: 'gini', **max_depth**: None, **max_features**: 10, **min_samples_split**: 330}), we can achieve an accuracy of 0.94 and an optimized AUROC of 0.82, which is a massive improvement from the previous dataset. This implied that the Lab Value features were very important predictors for sepsis outcomes.

Machine Learning: Adding the Clinical Latent Variables

The next step I implemented was the addition of Clinical Latent Variables. From the features we had, we could include clinically relevant features that could help discriminate between the two classes. Here, we included **Shock Index**, which is the Heart Rate divided by the Systolic Blood Pressure, and the **Oxygen Delivery Index**, which is the multiplication of the Heart Rate, Pulse Pressure (SBP-DBP), Oxygen Saturation, and Hemoglobin levels [2]. The inspiration and motivation behind this feature engineering comes from research and papers that discuss clinical data modeling to predict the outcome of cardiac arrest.

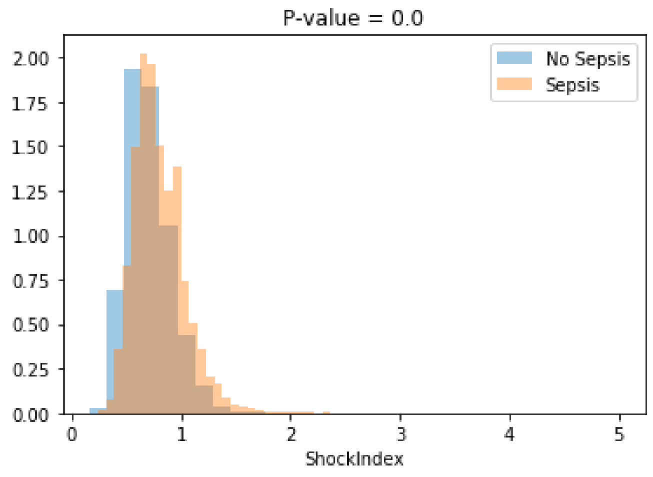


Figure 9: Distribution plot of Shock Index from Sepsis patients and Non-Sepsis patients

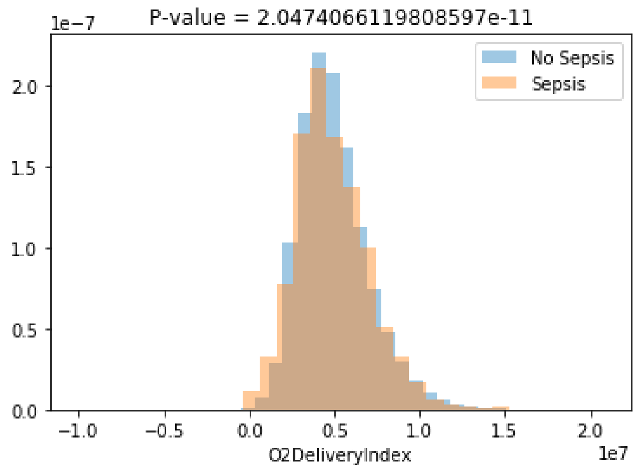


Figure 10: Distribution plot of Shock Index from Sepsis patients and Non-Sepsis patients

Based on these distribution plots and resulting p-values, we gathered that the mean Shock Index and O2 Delivery Index were in fact statistically different between the population diagnosed with Sepsis and the population without. That being said, it is worth noting that for the O2 Delivery index, despite having a next-to-zero p-value (2.04 x 10^-11), the two populations overlap to a large visual extent, indicating the difference may not be as convincing as the p-value implies. Regardless, both features were added to the dataset for the next round of model generation. Comments on their effects are further elucidated in the discussion section.

Machine Learning: Adding the Time-Series Trends

With this being a unique time series problem, the next appropriate round of features to include were trend features, in other words, features that could capture how other features changed over time and captured information in previous rows. Here I implemented two techniques: firstly, for the vital sign columns, I added an additional **lag column**, which shifts the original column down one row. Secondly, for each of the same columns mentioned previously, I added an additional **rolling mean column**, which was the mean of the three previous values of a feature. Comments on their effects are also further discussed in the discussion section.

Discussion

The models generated through adding the clinical latent variables, and those with the time-series trends, outputted metrics with very similar values to models using only the Lab Values, Vital Signs, and Demographics. In other words, unfortunately, the clinical latent variables and time series trends did not help the model discriminate between those who will get Sepsis and those who will not.

So far, we proceeded with a grouped pseudo forward feature selection process, using multiple supervised learning methods that used each addition of features to see how well (or unwell) the model performed. With every model generated, the precision, recall, and ROC curve were output as ways to evaluate model performance.

When choosing the best model for predicting sepsis based on timestamps, the metrics that needed to be monitored were multifaceted. Since this was an unbalanced dataset, accuracy was not the only metric to consider. Recall was important because it was imperative for the model to be able to reliably predict all positive cases. At the same time, precision must also had to be examined because hospital resources for treating sepsis are in need of being more appropriately allocated. Follow-up costs for sepsis are expensive and having a lenient threshold would prove to be extremely inefficient.

Moving forward with this project, I selected the Random Forest Model, only using the dataset that included the vital signs, demographics, and laboratory values, because general model performance made no significant improvement when adding in the clinical latent features, and time-series trends. Before moving on to transitioning from timestamp predictions to patient predictions, I wanted to further understand this selected model and its most discriminative features, in hopes of being able to understand what might have been the most important predictors for sepsis patients.

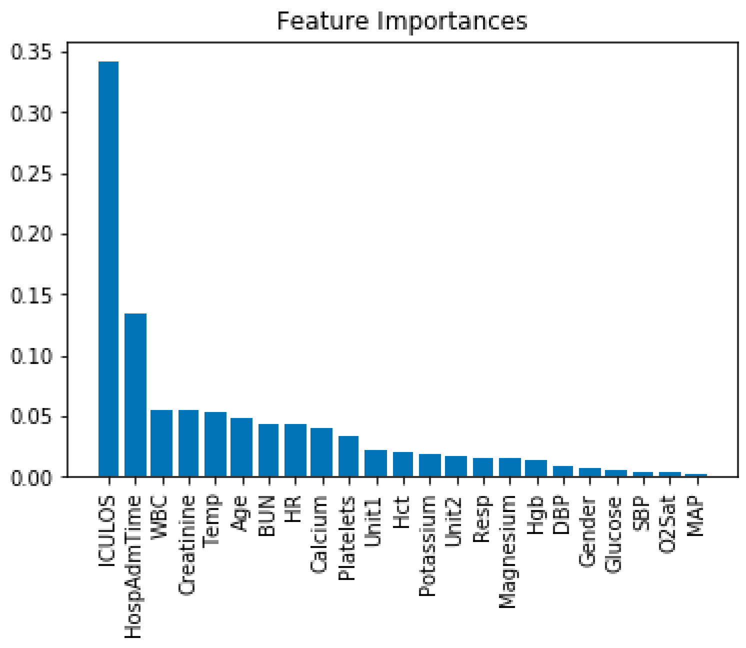


Figure 11: Feature Importances from the Random Forest model

From these results, I derived that the **ICU length-of-stay** (ICULOS) is the top discriminative feature, with **Hospital Admission Time before entering the ICU** (HospAdmTime) coming in second. This could indicate that a patient's length of stay at a hospital was strongly associated with outcome of their diagnosis. However, because the model devoted much importance to the ICULOS feature, three times as much importance as the next feature and at least six times as much as the rest of the features, this revealed the real possibility that the dataset, as it stood, did not have much predictive value. This is because despite the strong feature importance, the concept contradicted with the first barplot, (the duration in ICU before official diagnosis across all Sepsis patients), from the exploratory data analysis, which showed that over half of the majority of patients DID in fact get official diagnosis of Sepsis rather early on (within the first 24 hours). Nonetheless, I proceeded with this model due to its high performance and metrics, as I transitioned from timestamp predictions to patient predictions.

Transitioning from timestamp predictions to patient predictions

Now that we had a model that predict sepsis by timestamp, the next step was to use that model to predict sepsis by patient. One approach to doing this was to take in a patient, whose file would contain a number of timestamps (rows), use the model to predict each timestamp, and output an array of outcomes. The question then became: How many timestamps did we need to be positive in order to appropriately conclude that the patient will in fact get sepsis? The first thing I considered was that since patient file sizes can vary from 20 to 200 rows, it would be fair to determine our threshold based on the percentage of positive outcomes, instead of a direct number. Just as in model tuning and evaluation, where a data scientist uses functions like GridSearchCV to implement k-fold cross-validation and hyperparameter tuning to optimize models, a more manual approach can be done in this context. Here, the 'hyperparameter' we are looking to optimize is the percentage threshold!

The function, **predict_patient**, took in all patients, applied the model to predict sepsis on every row, and inferred whether the patient would get sepsis, based on multiple thresholds of positive timestamps. For each iteration of percentage thresholds, the accuracy, precision, recall, and f1-score were calculated, and could be visualized through the grouped bar plot that summarizes the metrics for each threshold, as seen in **Figure 12**.

The function **make_roc_curve** was also made, which takes in all patient files, predicts risk of Sepsis based on timestamp per patient, and generates a sepsis prediction score (sum of positive timestamps divided by the total number of timestamps), to create an ROC curve and obtain the AUROC as seen in **Figure 13**.

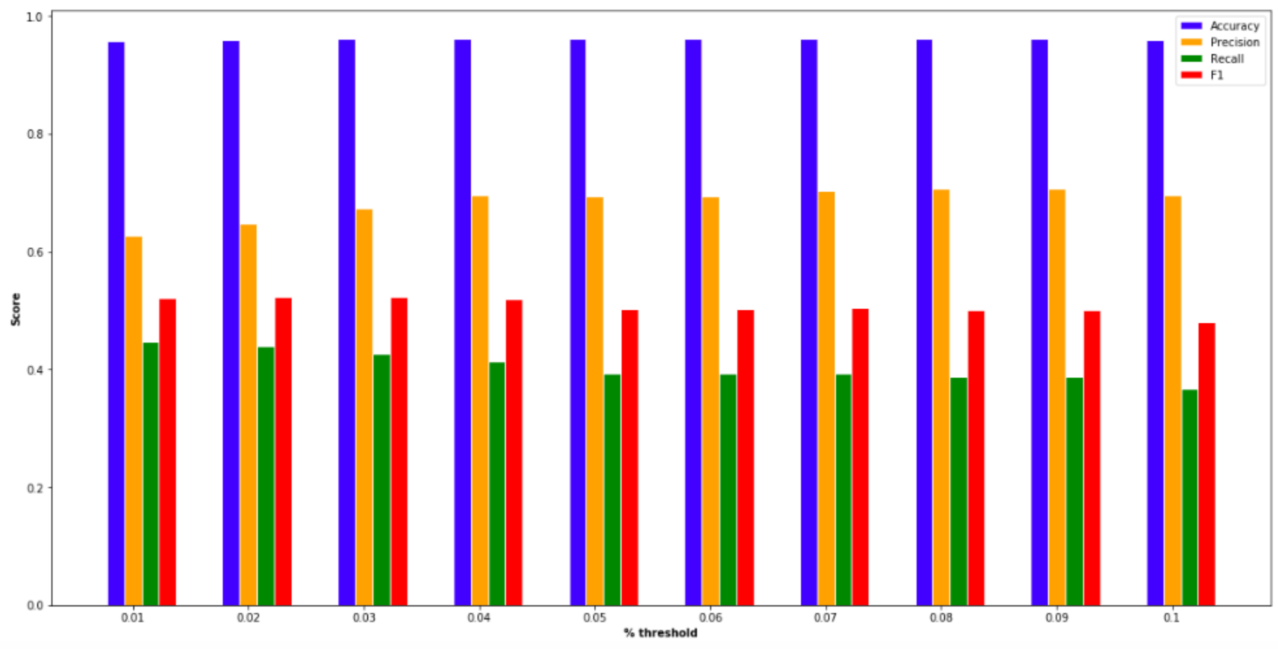


Figure 12: Grouped Bar Plot of metrics based on different positive percentage thresholds

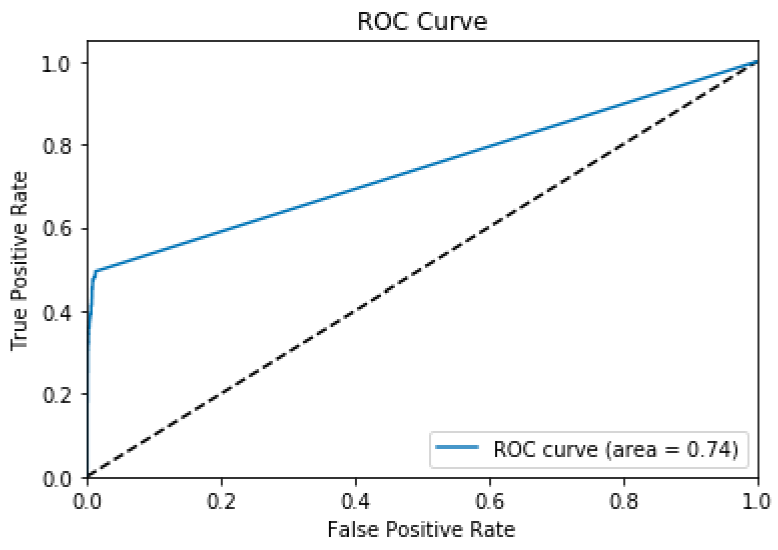


Figure 13: ROC curve generated from the patients and their sepsis scores

Further Discussion and Future Work

An interesting aspect the grouped bar plot shows is the increase in precision and decrease in recall, as the positive percentage threshold increases, which is a commonly well known trade-off in machine learning algorithms. The f1-score however, ever so slightly decreases, with its peak occurring when the positive percentage threshold is set at 0.02 (f1-score = 0.524). This implies the precision and recall are maximally optimized when 2% of timestamps from a patient's file are predicted positive for sepsis when applying our model. Moving forward, the function can be modified with the positive percentage threshold set at 0.02, and new patient files can then be inputted to generate predictions. The AUROC value of 0.74 indicated that the **predict_patient** function that utilized this model did fairly well in distinguishing the whole patients, from those with and without the onset of Sepsis.

Other areas for future work include:

- **Bring in more data:** With more patient files comes more opportunities to see how well the model predicts new cases, and what can potentially be done for further feature optimization.
- **Perform more exploratory data analysis:** I believe it would be worth doing more comparisons between the Sepsis and non-Sepsis population samples, checking aspects like, their mean lengths of stay at the ICU.
- **Predict number of hours before sepsis:** These datasets and models thus far were set up for binary classification, whether a patient will, or will not get sepsis. An important step forward would be to turn this into a regression problem such that our models would

predict the number of hours before a patient develops the condition. Creating such a model will allow hospital resources to be properly allocated at the appropriate times.

- **Attempt more forms of ML, such as Deep Learning:** Despite the clinical latent features not having a tremendous impact on model performance, they did present a very plausible discrimination between patients with and without sepsis, as seen when visualizing their distributions. This line of thought can be taken further when applying more advanced forms of machine learning such as deep learning, which automatically considers various interactions between different features.

[1] https://physionet.org/challenge/2019/

[2] https://tbiomed.biomedcentral.com/articles/10.1186/1742-4682-8-40