# Predicting Sepsis Onset in ICU Patients Using FCNN and Random Forest

Maziyar Mirzaei
UT-Austin 2024

GitHub Link

# Project Overview

**OBJECTIVE**: PREDICT THE RISK OF SEPSIS ONSET WITHIN THE ICU STAY, FOCUSING ON THE FIRST 24 HOURS.

**DATASET**: MIMIC-III CLINICAL DATABASE, CONTAINING DETAILED ICU PATIENT INFORMATION.

**JUSTIFICATION**: EARLY DETECTION AND TREATMENT CAN SAVE LIVES AND REDUCE COMPLICATIONS ASSOCIATED WITH SEPSIS.

# Load the Data

We load only the first 500,000 rows of chartevents and labevents as they large tables. But we can adjust the number of rows based on our computer memeory.

Print the shape of each DataFrame to understand the size of each table.

```python
# Load data tables from the MIMIC-III dataset
patients = pd.read_csv('PATIENTS.csv')
admissions = pd.read_csv('ADMISSIONS.csv')
icustays = pd.read_csv('ICUSTAYS.csv')
chartevents = pd.read_csv('CHARTEVENTS.csv', nrows=500000)  # Limit rows for faster loading
labevents = pd.read_csv('LABEVENTS.csv', nrows=500000)  # Limit rows for faster loading

# Display basic information about each DataFrame
print("Patients table shape:", patients.shape)
print("Admissions table shape:", admissions.shape)
print("ICU Stays table shape:", icustays.shape)
print("Chart Events table shape:", chartevents.shape)
print("Lab Events table shape:", labevents.shape)
```

```
Patients table shape: (46520, 8)
Admissions table shape: (58976, 19)
ICU Stays table shape: (61532, 12)
Chart Events table shape: (500000, 15)
Lab Events table shape: (500000, 9)
```

# Merge the Data

WE'LL NEED TO MERGE THE DATA TO GET COMPREHENSIVE INFORMATION FOR EACH PATIENT IN THE ICU.

WE JOIN ICUSTAYS WITH ADMISSIONS AND PATIENTS TO GET EACH ICU STAY'S ASSOCIATED ADMISSION AND PATIENT INFO.

THIS MERGED DATAFRAME (MERGED_DF) IS THE BASIS FOR LINKING ICU STAY DATA WITH PATIENT DEMOGRAPHICS AND OUTCOMES.

```python
# Merge ICU stay data with admissions and patient data for a complete dataset
merged_df = icustays.merge(admissions, on=['SUBJECT_ID', 'HADM_ID'], how='inner') \
                    .merge(patients, on='SUBJECT_ID', how='inner')
print("Merged data shape:", merged_df.shape)
```

```
Merged data shape: (61532, 36)
```

# Select Indicators of Sepsis

- Identify specific item IDs in `chartevents` and `labevents` that serve as sepsis indicators. Here, we'll use heart rate, blood pressure, temperature, and others. Adjust as per your study requirements
- Each `ITEMID` represents a specific measurement in MIMIC-III (e.g., heart rate, temperature).
- We'll use these values to extract time-series data from the `chartevents` and `labevents` tables.

```python
# Merge ICU stay data with admissions and patient data for a complete dataset
merged_df = icustays.merge(admissions, on=['SUBJECT_ID', 'HADM_ID'], how='inner') \
                    .merge(patients, on='SUBJECT_ID', how='inner')
print("Merged data shape:", merged_df.shape)
```

```
Merged data shape: (61532, 36)
```

```python
# Create a binary sepsis label based on the diagnosis description
merged_df['SEPSIS_LABEL'] = merged_df['DIAGNOSIS'].str.contains('sepsis', case=False, na=False).astype(int)
```

```python
merged_df.head()
```

| | ROW_ID_x | SUBJECT_ID | HADM_ID | ICUSTAY_ID | DBSOURCE | FIRST_CAREUNIT | LAST_CAREUNIT | FIRST_WARDID | LAST_WARDID | INTIME | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 365 | 268 | 110404 | 280836 | carevue | MICU | MICU | 52 | 52 | 2198-02-14 23:27:38 | ... |
| **1** | 366 | 269 | 106296 | 206613 | carevue | MICU | MICU | 52 | 52 | 2170-11-05 11:05:29 | ... |
| **2** | 367 | 270 | 188028 | 220345 | carevue | CCU | CCU | 57 | 57 | 2128-06-24 15:05:20 | ... |
| **3** | 368 | 271 | 173727 | 249196 | carevue | MICU | SICU | 52 | 23 | 2120-08-07 23:12:42 | ... |
| **4** | 369 | 272 | 164716 | 210407 | carevue | CCU | CCU | 57 | 57 | 2186-12-25 21:08:04 | ... |

5 rows × 37 columns

# Filter Data for the First 24 Hours of ICU Stay

To predict sepsis early, we'll focus on data from the first 24 hours of each ICU stay.

We merge chartevents with icustays to get each chart event's timestamp relative to ICU admission.

Filter chartevents to include only data within the first 24 hours of ICU stay.

Also we write tests to confirm the expected results

## Filter Data for the First 24 Hours of ICU Stay

```python
# Convert CHARTTIME and INTIME to datetime format
chartevents['CHARTTIME'] = pd.to_datetime(chartevents['CHARTTIME'])
icustays['INTIME'] = pd.to_datetime(icustays['INTIME'])

# Test: Confirm the column types are datetime
assert pd.api.types.is_datetime64_any_dtype(chartevents['CHARTTIME']), "CHARTTIME column is not datetime"
assert pd.api.types.is_datetime64_any_dtype(icustays['INTIME']), "INTIME column is not datetime"
print("Datetime conversion test passed!")
```

```
Datetime conversion test passed!
```

```python
# Merge chart events with ICU stay start times to calculate elapsed time
merged_chartevents = chartevents.merge(icustays[['ICUSTAY_ID', 'INTIME']], on='ICUSTAY_ID', how='inner')
chartevents_24hr = merged_chartevents[merged_chartevents['CHARTTIME'] <= merged_chartevents['INTIME'] + pd.Timedelta(hours=24)]

# Test: Confirm all times in chartevents_24hr are within 24 hours of INTIME
assert (chartevents_24hr['CHARTTIME'] - chartevents_24hr['INTIME']).dt.total_seconds().max() <= 24 * 3600, "Some records are beyond 24 hou
print("24-hour filter test passed!")
```

```
24-hour filter test passed!
```

# Aggregate Vital Signs

To get a summarized view of the first 24 hours, calculate the mean, maximum, and minimum for each measurement.

We filter chartevents_24hr by sepsis_itemids, focusing only on sepsis-relevant measurements.

Using .pivot(), we create a new DataFrame (sepsis_features) with each ICU stay as a row and features as columns.

## Aggregate Vital Signs

```python
# Filter for relevant sepsis-related ITEMIDs
filtered_sepsis_vitals = chartevents_24hr[chartevents_24hr['ITEMID'].isin(sepsis_itemids)]

# Aggregate statistics for mean, max, and min values
sepsis_agg = (
    filtered_sepsis_vitals.groupby(['ICUSTAY_ID', 'ITEMID'])['VALUENUM']
    .agg(['mean', 'max', 'min'])
    .unstack(fill_value=0)
)

# Flatten the multi-level columns by concatenating statistics and ITEMID
sepsis_agg.columns = [f"{stat}_{itemid}" for stat, itemid in sepsis_agg.columns]

# Diagnostic: Check which ITEMIDs are present and the column names
print("Unique ITEMIDs in filtered data:", filtered_sepsis_vitals['ITEMID'].unique())
print("Expected ITEMIDs:", sepsis_itemids)
print("Columns after aggregation:", sepsis_agg.columns)

# Define all expected columns based on `sepsis_itemids`
expected_columns = [f"{stat}_{itemid}" for itemid in sepsis_itemids for stat in ['mean', 'max', 'min']]

# Reindex `sepsis_agg` to ensure all expected columns are present, filling missing ones with 0
sepsis_features = sepsis_agg.reindex(columns=expected_columns, fill_value=0).reset_index()

# Adjust the assertion to verify that the final columns match the expected set
assert set(sepsis_features.columns) == set(['ICUSTAY_ID'] + expected_columns), \
    "Warning: Some expected ITEMIDs are missing from the final features."

# Display the first few rows of the final features
print("Final Sepsis Features DataFrame with Filled Missing Columns:")
sepsis_features.head()
```

```
Unique ITEMIDs in filtered data: [220045 220179 220180 220210 220277]
Expected ITEMIDs: [220045, 220179, 220180, 220210, 220277, 50983, 50971, 50885]
Columns after aggregation: Index(['mean_220045', 'mean_220179', 'mean_220180', 'mean_220210',
       'mean_220277', 'max_220045', 'max_220179', 'max_220180', 'max_220210',
       'max_220277', 'min_220045', 'min_220179', 'min_220180', 'min_220210',
       'min_220277'],
      dtype='object')
Final Sepsis Features DataFrame with Filled Missing Columns:
```

| | ICUSTAY_ID | mean_220045 | max_220045 | min_220045 | mean_220179 | max_220179 | min_220179 | mean_220180 | max_220180 | min_220180 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 200563.0 | 96.360000 | 108.0 | 87.0 | 129.631579 | 153.0 | 107.0 | 79.684211 | 101.0 | 62.0 | ... |
| 1 | 200566.0 | 71.000000 | 84.0 | 59.0 | 129.333333 | 168.0 | 88.0 | 50.000000 | 72.0 | 31.0 | ... |
| 2 | 200603.0 | 80.416667 | 82.0 | 80.0 | 88.769231 | 105.0 | 77.0 | 44.692308 | 51.0 | 38.0 | ... |
| 3 | 200746.0 | 96.032258 | 111.0 | 86.0 | 120.400000 | 133.0 | 102.0 | 77.100000 | 91.0 | 60.0 | ... |
| 4 | 200806.0 | 68.911765 | 84.0 | 55.0 | 110.911765 | 164.0 | 71.0 | 53.764706 | 120.0 | 30.0 | ... |

5 rows × 25 columns

# Prepare Data for Training

- Define the target variable and split the data into training and test sets.
- X contains features for each ICU stay; y is the binary target (e.g., sepsis onset).
- `train_test_split` separates data for model training and evaluation.
- `StandardScaler` scales X values for better neural network performance.

**Prepare Data for Training**

```
# Assuming 'SEPSIS_LABEL' is a column in merged_df for labeling cases of sepsis
X = sepsis_features.drop(columns=['ICUSTAY_ID']).values
y = merged_df.set_index('ICUSTAY_ID').loc[sepsis_features['ICUSTAY_ID'], 'SEPSIS_LABEL'].values

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the feature data for neural network training
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

# Train Models

- We train two models and compare them in terms of accuracy :
  - Random Forest and a Fully Connected Neural Network.
- `RandomForestClassifier` is a powerful, interpretable model.
- `fcnn_model` is a neural network with five hidden layers and dropout for regularization.

### Train Random Forest Models

```python
# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

### Train FCNN Model

```python
# Define and train a 5-layer Fully Connected Neural Network (FCNN)
fcnn_model = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),  # Define input layer explicitly
    Dense(128, activation='relu'),           # First hidden layer
    Dropout(0.3),                            # Dropout for regularization
    Dense(64, activation='relu'),            # Second hidden layer
    Dropout(0.3),                            # Dropout for regularization
    Dense(32, activation='relu'),            # Third hidden layer (new layer)
    Dropout(0.3),                            # Dropout for regularization
    Dense(16, activation='relu'),            # Fourth hidden layer (new layer)
    Dropout(0.3),                            # Dropout for regularization
    Dense(1, activation='sigmoid')           # Output layer for binary classification
])

# Compile the model
fcnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = fcnn_model.fit(X_train_scaled, y_train, epochs=20, validation_data=(X_test_scaled, y_test), batch_size=64)
```

# Evaluate Models

- We use accuracy and AUC-ROC for evaluation.
- We calculate accuracy and AUC-ROC to gauge model performance.
- AUC-ROC provides insight into how well the model differentiates between sepsis and non-sepsis cases.

```python
# Evaluate the Random Forest model
y_pred_rf = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_auc = roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1])

# Evaluate the FCNN model
y_pred_fcnn = (fcnn_model.predict(X_test_scaled) > 0.5).astype("int32")
fcnn_accuracy = accuracy_score(y_test, y_pred_fcnn)
fcnn_auc = roc_auc_score(y_test, fcnn_model.predict(X_test_scaled))

# Print evaluation metrics
print("Random Forest - Accuracy:", rf_accuracy, "AUC-ROC:", rf_auc)
print("FCNN - Accuracy:", fcnn_accuracy, "AUC-ROC:", fcnn_auc)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 518us/step
Random Forest - Accuracy: 0.9645390070921985 AUC-ROC: 0.7235294117647059
FCNN - Accuracy: 0.9645390070921985 AUC-ROC: 0.39705882352941174
```

Based on the output:

- **Random Forest Model**: Achieved an accuracy of 0.965 and an AUC-ROC of 0.724.
- This indicates good accuracy, but the AUC-ROC suggests that the model struggles to separate positive and negative cases as effectively as desired.
- **FCNN Model**: Shows an accuracy of 0.965 (same as Random Forest), but a lower AUC-ROC of 0.397.
- The lower AUC-ROC (below 0.5) suggests that the model's predictions are not reliably distinguishing between the classes, indicating issues with the FCNN's ability to separate positive and negative cases in a meaningful way, despite the high accuracy.

# Visualize Training Performance

- For neural networks, visualize training history to monitor overfitting or underfitting.

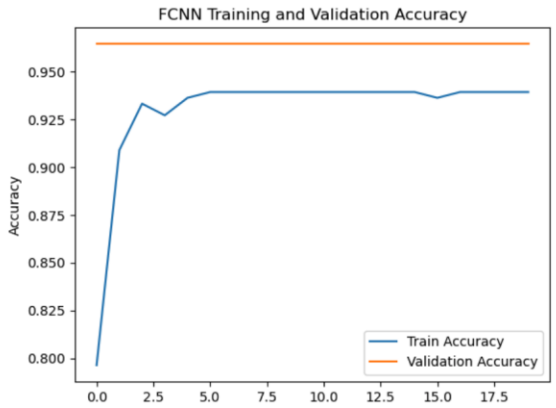
## Visualize Training Performance

```python
# Plot training and validation accuracy for FCNN
# Print available keys in history to confirm accuracy key presence
print("Available keys in history:", history.history.keys())

# Plot training and validation accuracy for FCNN
plt.plot(history.history.get('accuracy', []), label='Train Accuracy')
plt.plot(history.history.get('val_accuracy', []), label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title("FCNN Training and Validation Accuracy")
plt.show()

# Plot training and validation loss for FCNN
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title("FCNN Training and Validation Loss")
plt.show()
```

```
Available keys in history: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

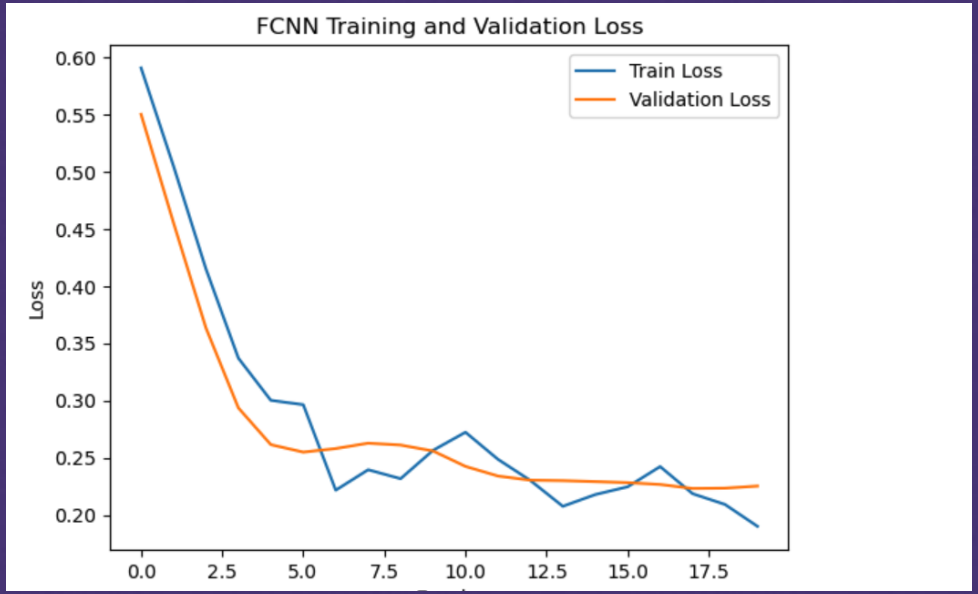
# FCNN Training and Validation Accuracy

- This graph displays the model's accuracy on both training and validation data across each epoch during training.
- **Training Accuracy** (Blue Line): Shows how well the model fits the training data over time.
- **Validation Accuracy** (Orange Line): Indicates the model's performance on unseen data (validation set) as training progresses.
- **Observation**:
  - The training accuracy starts low and gradually improves, stabilizing around 95-97%, which indicates that the model is learning to classify well on the training data.
  - The validation accuracy quickly reaches a high value and plateaus, suggesting that the model generalizes well to validation data.



# FCNN Training and Validation Loss

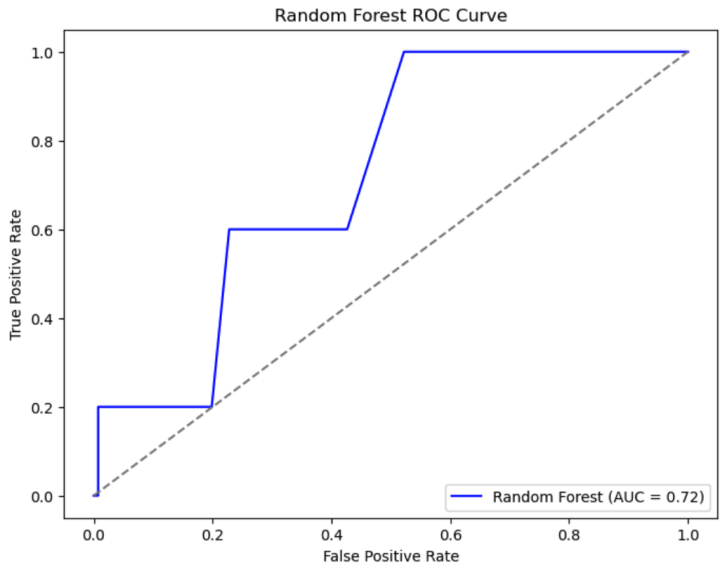
- This graph illustrates the loss on both training and validation data for each epoch.
- **Training Loss** (Blue Line): Represents the model's error on the training data over time.
- **Validation Loss** (Orange Line): Shows how the model performs on unseen data (validation set) in terms of error.
- **Observation**:
  - The training loss decreases over time, showing that the model is minimizing its error on the training data as expected.
  - The validation loss follows a similar trend but plateaus as training progresses, indicating that the model is not significantly overfitting.
  - Both the training and validation loss stabilize at low values, which aligns with the high accuracy observed in the accuracy graph.



# Random Forest ROC Curve for Sepsis Prediction

- **ROC Curve (Receiver Operating Characteristic Curve)**:
  - The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied.
  - In this graph, the **True Positive Rate** (Sensitivity) is plotted against the **False Positive Rate** (1 - Specificity).
  - The **blue line** represents the ROC curve for the Random Forest model.
- **AUC (Area Under the ROC Curve)**:
  - The AUC score, shown as **0.72**, represents the model's ability to distinguish between sepsis and non-sepsis cases.
  - An AUC of 0.72 indicates that the Random Forest model has moderate predictive power, successfully identifying sepsis cases 72% of the time when given randomly chosen positive and negative cases.

```python
from sklearn.metrics import roc_curve, auc

# Get probabilities for positive class (sepsis)
y_proba_rf = rf_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='blue', label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



# Model Performance Summary

- **Random Forest Model**:
  - **Accuracy**: 96.45%
  - **AUC-ROC**: 0.72
- **Fully Connected Neural Network (FCNN)**:
  - **Accuracy**: 96.45%
  - **AUC-ROC**: 0.36
- **Key Takeaways**:
  - Both models achieved similar high accuracy.
  - Random Forest had a higher AUC-ROC, indicating better performance in distinguishing between classes.
  - FCNN's low AUC-ROC suggests that it may not effectively separate positive and negative classes, despite its high accuracy.

```
print("Model Performance Summary")
print("-------------------------")
print("Random Forest - Accuracy:", rf_accuracy, "| AUC-ROC:", rf_auc)
print("FCNN - Accuracy:", fcnn_accuracy, "| AUC-ROC:", fcnn_auc)
```

```
Model Performance Summary
-------------------------
Random Forest - Accuracy: 0.9645390070921985 | AUC-ROC: 0.7235294117647059
FCNN - Accuracy: 0.9645390070921985 | AUC-ROC: 0.3588235294117647
```