

Sepsis Prediction App for ICU Patients

1.0 Introduction

In the field of healthcare, accurate and prompt diagnosis is critical for providing successful patient treatment. Sepsis, a potentially fatal illness caused by the body's response to an infection, necessitates prompt diagnosis to avoid its progression. A FastAPI-based online application has been created to assist medical practitioners in this attempt. Based on input features, this application uses machine learning to predict the occurrence of sepsis in patients. This program provides a significant tool for healthcare providers to anticipate and handle sepsis cases quickly by employing a trained model on pertinent patient data.

2.0 Review

Healthcare practitioners are continuously looking for new methods to enhance patient outcomes, and predictive tools can help tremendously. Traditional diagnostic methods can be time-consuming and prone to human error. Machine Learning entrance into medical practice has paved the way for more efficient and accurate diagnosis. This paper goes into a machine learning-powered application for predicting sepsis in patients. The tool's use of FastAPI for API development, Docker for deployment, and integration with the Hugging Face platform is quite exciting.

3.0 Setup and Installation

It is critical to set up the proper environment before getting into the specifics of the application. The article begins by demonstrating how to install essential libraries such as scikit-learn and imbalanced-learn. These libraries are required for the development and deployment of machine learning models. Furthermore, the article emphasizes the usage of joblib for object storing, which is an important technique for ensuring model consistency and reusability.

4.0 Data Handling and Exploratory Data Analysis (EDA)

The lifeblood of every machine learning venture is data. The article shows the detailed methods used to load and explore the dataset in this part. The article covers an in-depth examination of the dataset using frameworks such as pandas, numpy, seaborn, and matplotlib. This research employs data visualization and statistical insights to comprehend the structure, distribution, and properties of the dataset. Furthermore, the paper emphasizes the importance

of dealing with missing values and duplicate entries in order to assure data quality and trustworthiness.

5.0 Data Loading and Dataset Overview

The article starts by showing how to load the dataset into Google Colab using the drive mount mechanism. It shows the first few rows of the dataset to give you an idea of its structure and content. The dataset includes parameters such as plasma glucose, blood pressure, body mass index, and others, all of which can predict the occurrence of sepsis. Furthermore, the categorical variable "Sepsis" indicates whether or not a patient has developed sepsis. This is shown in the table below.

ID	PRG	PL	PR	SK	TS	M11	BD2	Age	Insurance	Sepssis	
0	ICU200010	6	148	72	35	0	33.6	0.627	50	0	Positive
1	ICU200011	1	85	66	29	0	26.6	0.351	31	0	Negative
2	ICU200012	8	183	64	0	0	23.3	0.672	32	1	Positive
3	ICU200013	1	89	66	23	94	28.1	0.167	21	1	Negative
4	ICU200014	0	137	40	35	168	43.1	2.288	33	1	Positive
...
594	ICU200604	6	123	72	45	230	33.6	0.733	34	0	Negative
595	ICU200605	0	188	82	14	185	32.0	0.682	22	1	Positive
596	ICU200606	0	67	76	0	0	45.3	0.194	46	1	Negative
597	ICU200607	1	89	24	19	25	27.8	0.559	21	0	Negative
598	ICU200608	1	173	74	0	0	36.8	0.088	38	1	Positive

599 rows × 11 columns

6.0 Data Types and Missing Values

A full investigation of the dataset's metadata is provided, outlining the meaning and significance of each column. The article then examines the general information in the dataset, indicating that there are no missing values in any of the columns. This clean dataset is required for training effective machine learning models.

7.0 Exploratory Data Analysis (EDA)

The article delves into Exploratory Data Analysis (EDA), a crucial phase for understanding the dataset's characteristics and relationships. Through histograms, summary statistics, and visualizations, the article provides valuable insights into the numerical variables' distribution and behaviour. Key observations include the right-skewed distribution of Plasma glucose, the relatively normal distribution of Blood Work Result-1, and the highly right-skewed Blood Work Result-3 distribution due to extreme values. This is shown in Figure 1.

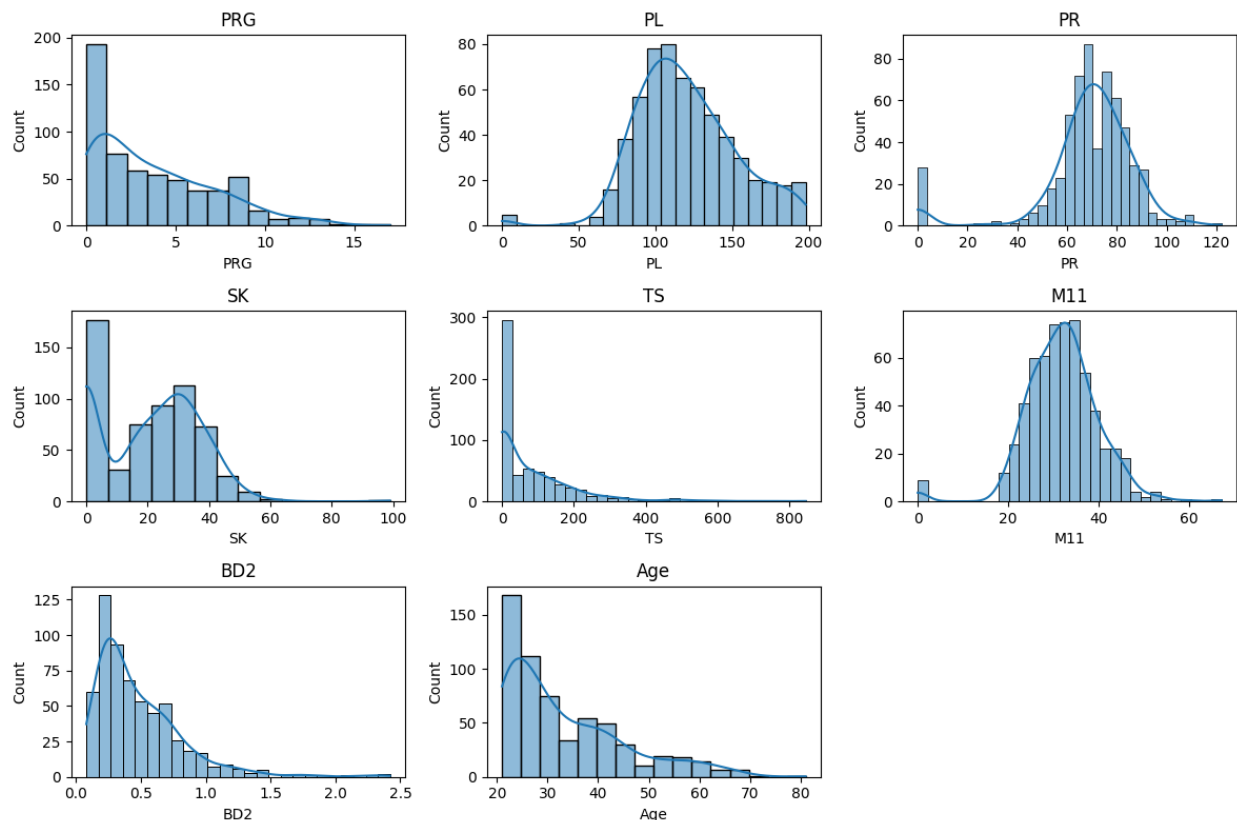


Figure 1

8.0 Univariate and Bivariate Analysis

Uncovering patterns and relationships in a dataset requires the use of univariate and bivariate analytics. The article discusses the principles and then applies them to the dataset to demonstrate their importance. Box plots are used to depict the distribution of numerical variables within the "Sepsis" category. These charts show how different numerical characteristics change between patients with and without sepsis. Furthermore, the report includes a categorical analysis that shows how the presence of sepsis differs with different insurance kinds. These are shown in Figures 2, 3, 4 and 5.

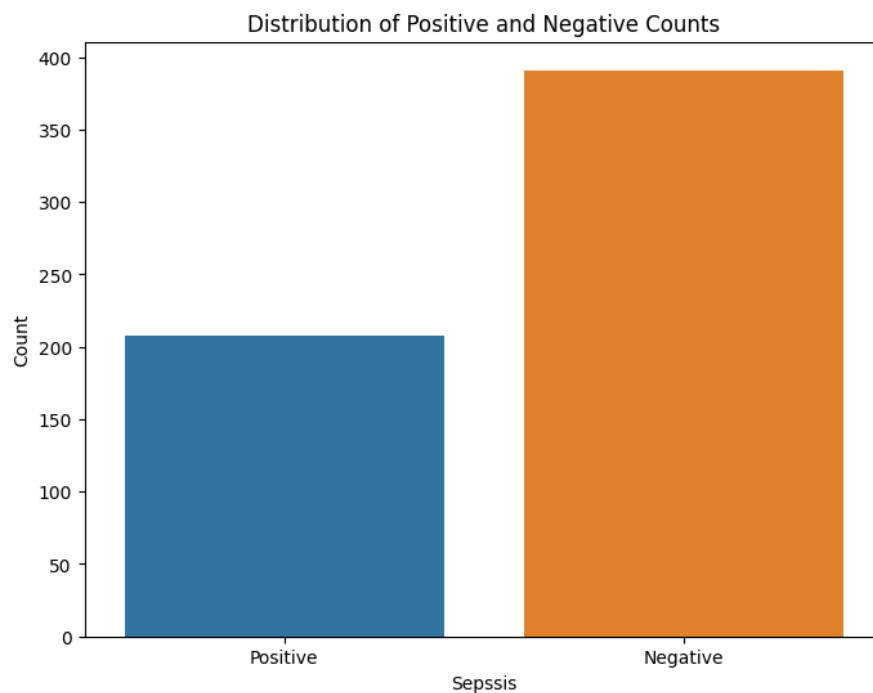


Figure 2

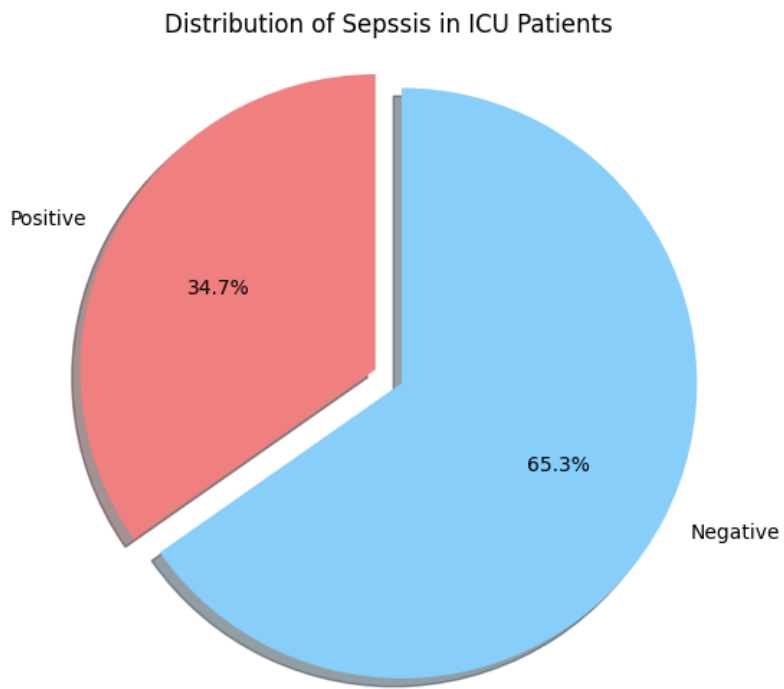


Figure 3

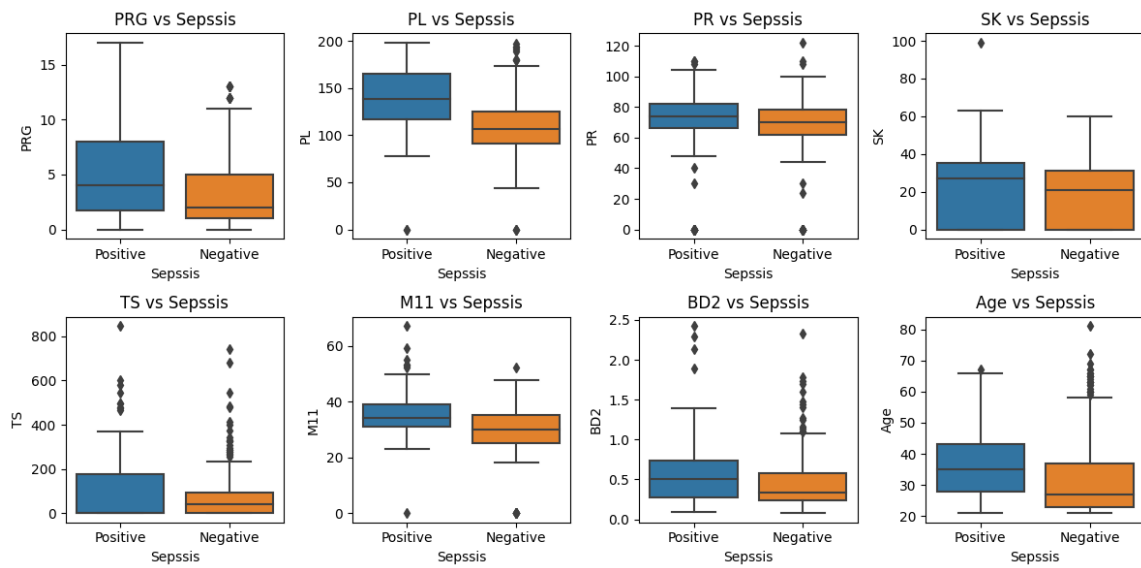


Figure 4

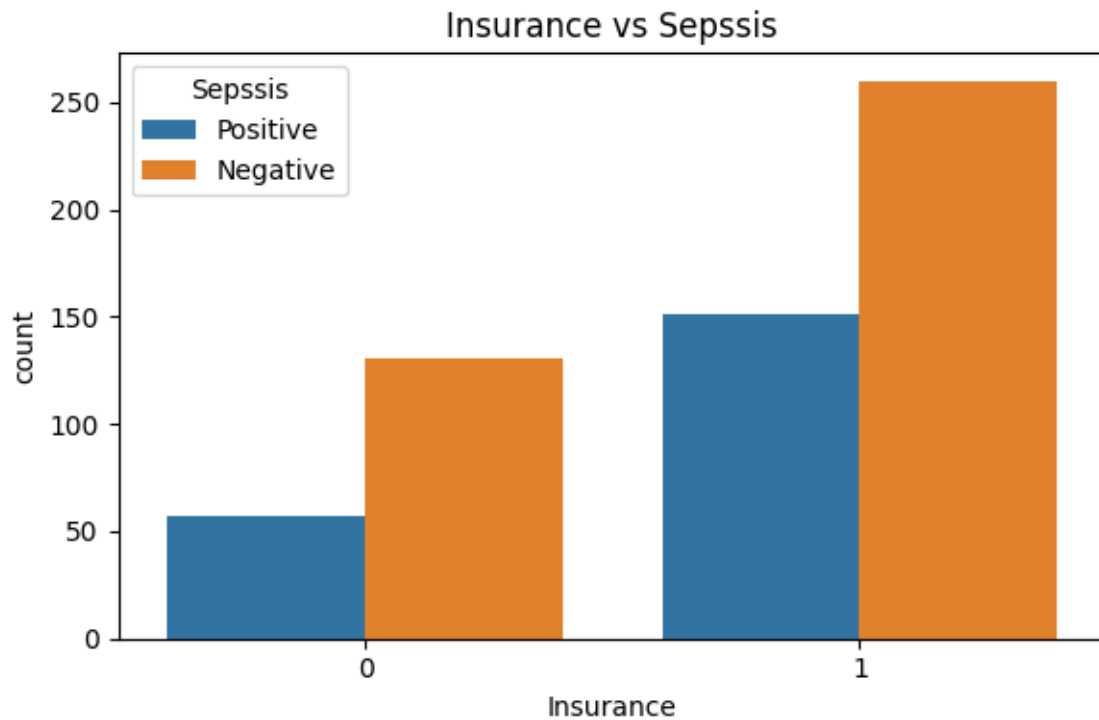


Figure 5

9.0 Correlation Heatmap

Understanding relationships between numerical features is vital for feature selection and model building. The article presents a correlation heatmap, illustrating the strength and direction of correlations between numerical variables. This provides insights into potential multicollinearity and helps in choosing features for machine learning models. The heatmap is shown in Figure 6.

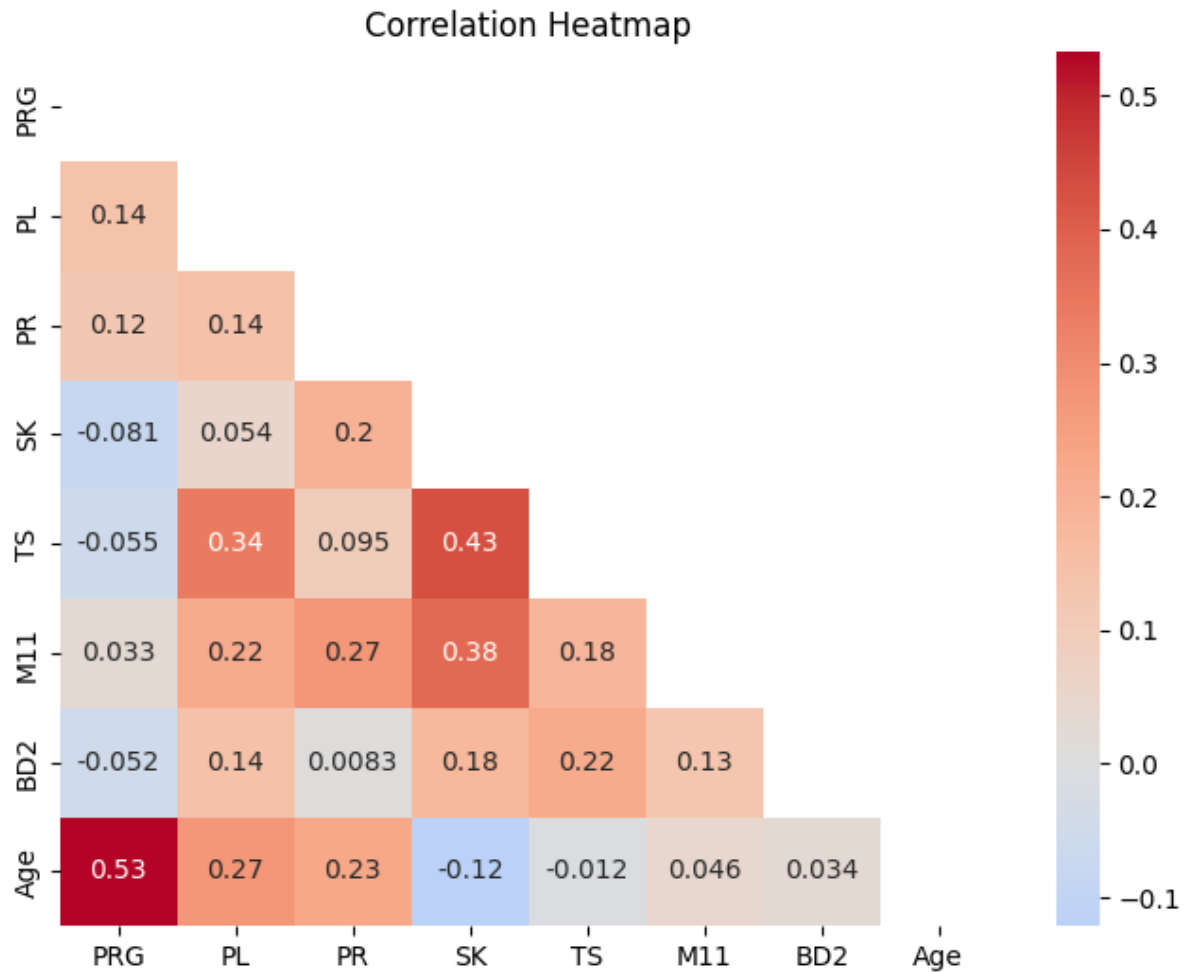


Figure 6

10. Model Selection and Evaluation

After preprocessing the data and engineering relevant features, the next step is to select and evaluate machine learning models. In this section, we will train and evaluate nine different **classification** models to find the best performers for our task. The models we will explore include:

1. **Logistic Regression**
2. **K-Nearest Neighbors**
3. **Decision Tree**
4. **Support Vector Machine (Linear Kernel)**
5. **Support Vector Machine (RBF Kernel)**

6. **Neural Network**
7. **Random Forest**
8. **Gradient Boosting**
9. **XGBoost**

To ensure a fair comparison, each model will be trained using the same pipeline, consisting of a standard scaler for feature scaling and the specific classification algorithm. We will evaluate the models based on precision, recall, F1-score, and accuracy to understand their performance across different evaluation metrics.

11. Hyperparameter Tuning

After identifying promising models, the next step is to fine-tune their hyperparameters to achieve optimal performance. In this section, we will focus on the top-performing models:

1. **Gradient Boosting**
2. **K-Nearest Neighbors**
3. **Support Vector Machine (RBF Kernel)**
4. **Logistic Regression**

We will use GridSearchCV, a technique that exhaustively searches through a predefined hyperparameter grid to find the best combination of hyperparameters. By tuning the hyperparameters, we aim to further improve the models' performance on the validation data. The performance of the four models based on F1-score is shown in the horizontal bar graphs in Figure 7.

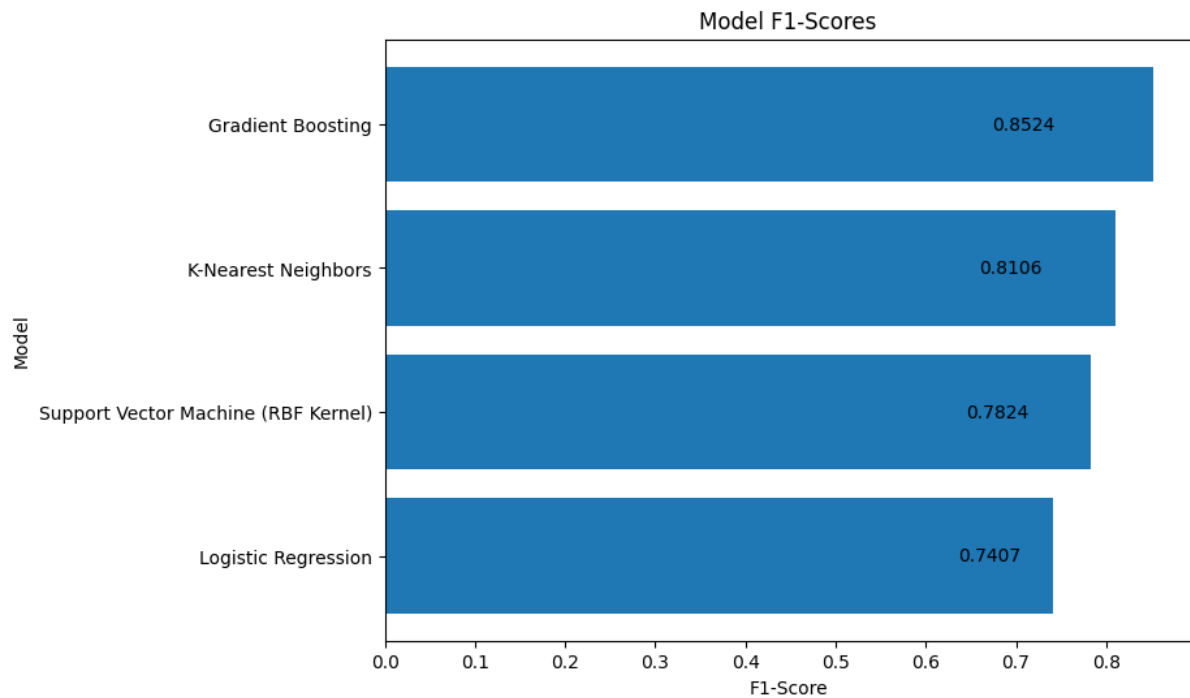


Figure 7

12. Model Selection and Performance Comparison

In this section, we will compare the performance of the hyperparameter-tuned models and select the final model for deployment. We will consider metrics such as the F1-score, accuracy, and other relevant evaluation metrics to make an informed decision. The chosen model will be the one that demonstrates the highest overall performance on the validation data.

13. Feature Importance Analysis

Understanding the importance of different features in a predictive model is crucial for gaining insights into the factors driving predictions. In this section, we will analyze the feature importances of the final selected model (Gradient Boosting). By visualizing and interpreting feature importances, we can identify the most influential features that contribute to the model's predictions as shown in Figure 8.

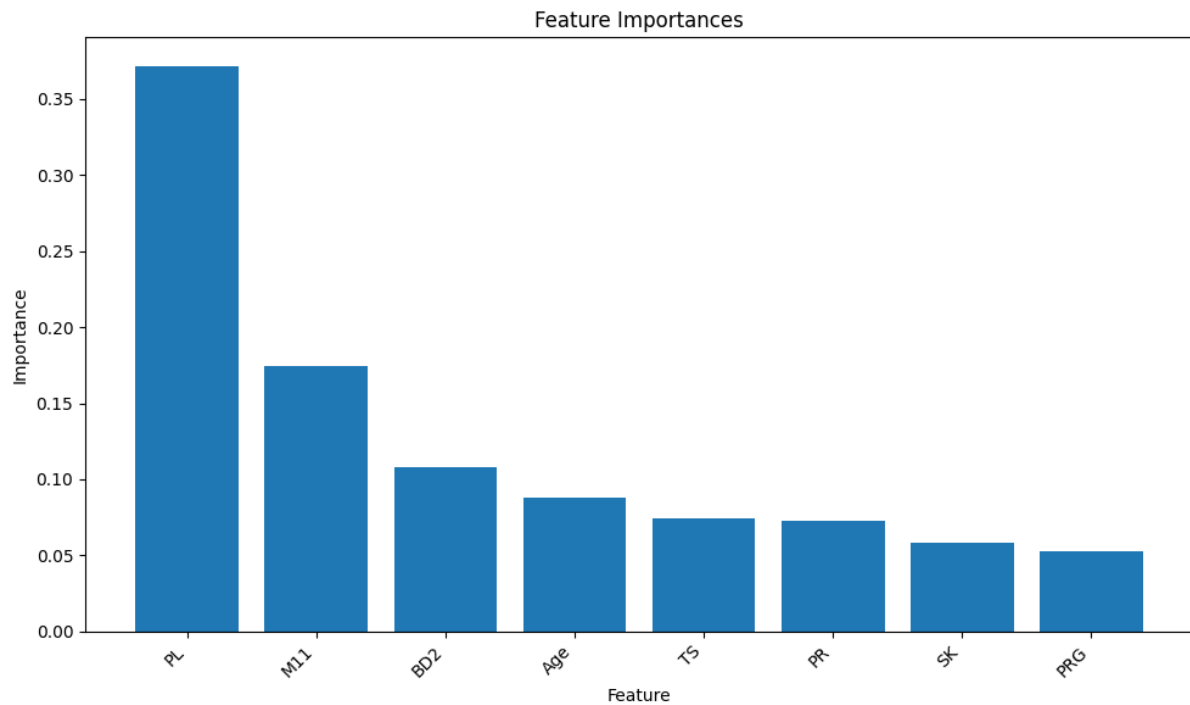


Figure 8

14. ROC Curve and AUC Analysis

The Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (AUC) score are essential tools for evaluating the performance of binary classification models. In this section, we will visualize the ROC curve of the final selected model and calculate the AUC score. The ROC curve helps us understand the trade-off between true positive rate and false positive rate at various classification thresholds, while the AUC score quantifies the model's ability to discriminate between the positive and negative classes.

- The value of 0.785 for the AUC (Area Under the Curve) score indicates the performance of the Gradient Boosting model in distinguishing between the positive and negative classes. The AUC score ranges from 0 to 1, where a score of 0.5 represents a random classifier, and a score of 1.0 represents a perfect classifier. See Figure 9.

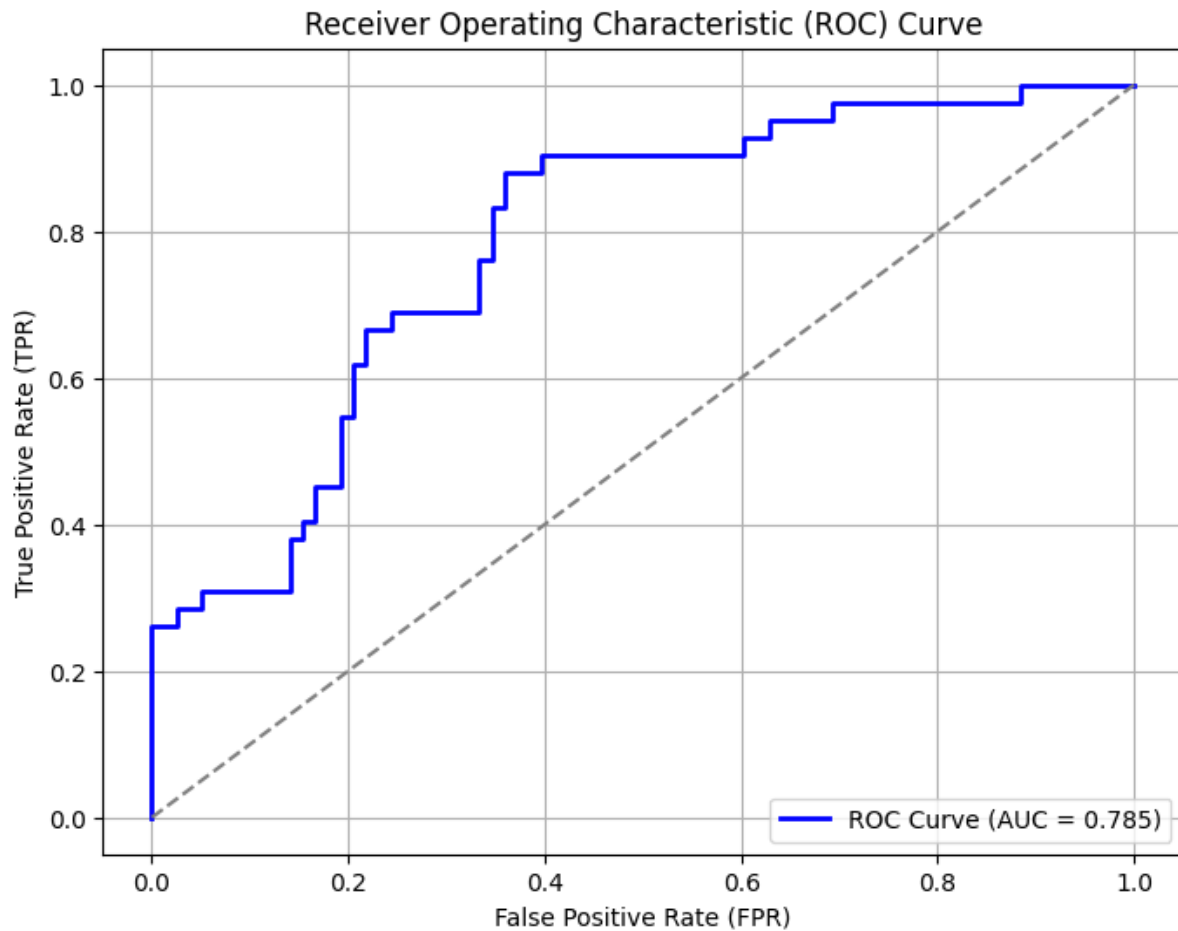


Figure 9

15. Confusion Matrix Analysis

The confusion matrix is an effective visual tool for evaluating a model's performance, particularly in terms of classification errors. In this section, we will examine the final selected model's confusion matrix to obtain insight into its true positive, true negative, false positive, and false negative predictions. We can understand the types of errors made by the model and make informed decisions regarding its performance by studying these numbers. The confusion matrix shown in Figure 10 can be summarised as follows:

The confusion matrix for the Gradient Boosting model indicates the following:

True Positive (TP): There are 26 instances that are correctly predicted as positive (actual positive and predicted positive).

True Negative (TN): There are 62 instances that are correctly predicted as negative (actual negative and predicted negative).

False Positive (FP): There are 16 instances that are incorrectly predicted as positive (actual negative but predicted positive).

False Negative (FN): There are 16 instances that are incorrectly predicted as negative (actual positive but predicted negative).

These results show that the Neural Network model correctly predicted 26 positive cases and 62 negative cases. However, it misclassified 16 instances as false positives and 16 instances as false negatives. The model's performance is not perfect, but it is capturing both positive and negative cases to some extent.

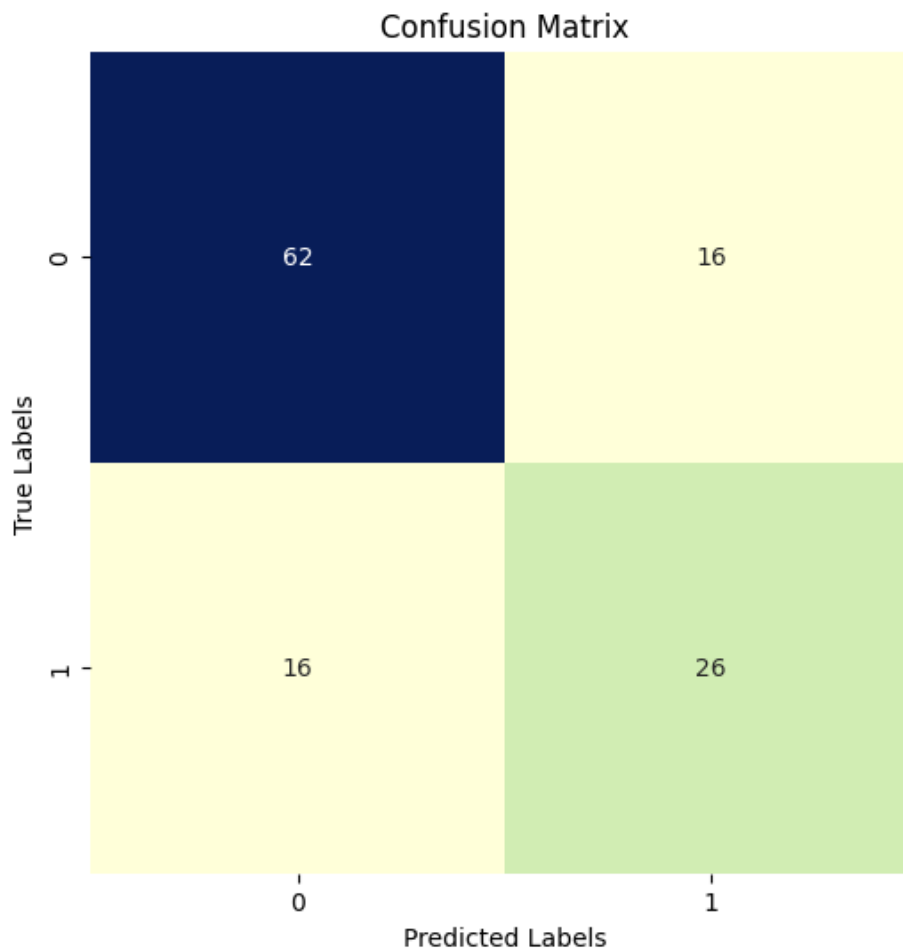


Figure 10

16. Model Predictions on New Data

Finally, when we've constructed and fine-tuned our model, we'll apply it to new, previously unseen data to generate predictions. We'll load the test dataset, preprocess it similarly to the training and validation data, and then apply the chosen model to estimate the likelihood of sepsis for each patient. We will describe how the model performs on this hitherto unseen data and the potential consequences for clinical decision-making.

By addressing these essential aspects, we will be able to write a thorough article that covers the whole process of developing and testing a machine learning model for sepsis prediction. This will be useful for both data scientists and healthcare professionals interested in using machine learning to medical diagnosis.

17. Building and Deploying the FastAPI Application

With the machine learning model trained and evaluated, the next step is to create an API that allows us to make predictions using the model. We'll be using the FastAPI framework, a modern and efficient web framework for building APIs in Python. The FastAPI app will take in patient data as input and provide predictions for sepsis likelihood based on the trained model.

17.1 Setting Up the FastAPI App

In this section, we'll walk through the steps of setting up the FastAPI application. We'll define a Pydantic model to represent the input data for the prediction, load the trained machine learning components, and create API routes to handle prediction requests. The FastAPI code can be found in my repository link at the end of this article..

17.2 Handling Predictions and Providing Results

In the FastAPI app, we've defined an endpoint **/classify** that accepts patient data as input and returns predictions for sepsis likelihood. The input data is processed, and the trained machine learning model is used to make predictions. The app then returns the predictions along with confidence scores for each class. These are illustrated in Figures 11, 12, 13, 14 and 15 below:

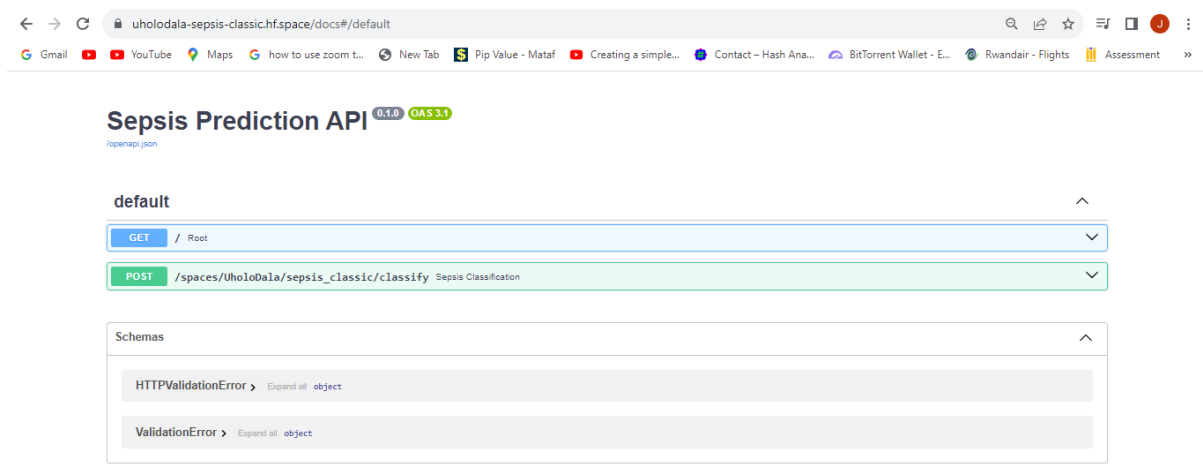


Figure 11

Output of the root endpoint:

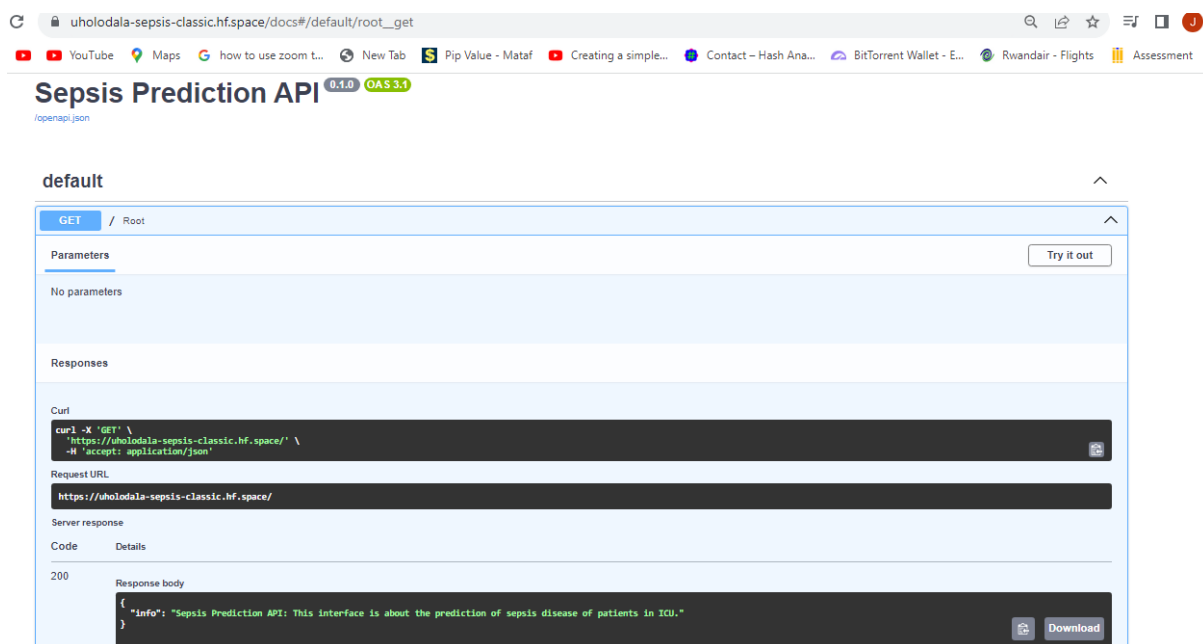


Figure 12

Sepsis_Classification_endpoint_Pre_execution:

POST /spaces/UholoDala/sepsis_classic/classify Sepsis Classification

Parameters

Name	Description
PlasmaGlucose * required integer (query)	7
BloodWorkResult_1 * required integer (query)	150
BloodPressure * required integer (query)	78
BloodWorkResult_2 * required integer (query)	29
BloodWorkResult_3 * required integer (query)	126
BodyMassIndex * required number (query)	35.2
BloodWorkResult_4 * required number (query)	0.692
Age * required integer	54

Cancel

Figure 13

Sepsis_Classification_endpoint_Post_execution:

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'https://uholodala-sepsis-classic.hf.space/spaces/UholoDala/sepsis_classic/classify?PlasmaGlucose=7&BloodWorkResult_1=150&BloodPressure=78&BloodWorkResult_2=29&BloodWorkResult_3=126&BodyMassIndex=35.2&BloodWorkResult_4=0.692&Age=54' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

https://uholodala-sepsis-classic.hf.space/spaces/UholoDala/sepsis_classic/classify?PlasmaGlucose=7&BloodWorkResult_1=150&BloodPressure=78&BloodWorkResult_2=29&BloodWorkResult_3=126&BodyMassIndex=35.2&BloodWorkResult_4=0.692&Age=54


Server response

Code	Details
200	<p>Response body</p> <pre>{ "Execution_msg": "Execution went fine", "execution_code": 1, "prediction": { "PlasmaGlucose": 7, "BloodWorkResult_1(U/ml)": 150, "BloodPressure(mm Hg)": 78, "BloodWorkResult_2(mm)": 29, "BloodWorkResult_3(U/ml)": 126, "BodyMassIndex(kg/m^2)": 35.2, "BloodWorkResult_4(U/ml)": 0.692, "Age (years)": 54, "Predicted label": "Negative", "Confidence_Negative": 98.72859691757037, "Confidence_Positive": 1.2714030824296179 } }</pre> <p>Download</p>

Response headers

Figure 14

Downloaded prediction as displayed in notepad:



The screenshot shows a Notepad window titled "response_1692597408630 - Notepad". The window contains a JSON object representing a prediction response. The JSON structure is as follows:

```
{
  "Execution_msg": "Execution went fine",
  "execution_code": 1,
  "prediction": [
    {
      "PlasmaGlucose": 7,
      "BloodWorkResult_1(U/ml)": 150,
      "BloodPressure(mm Hg)": 78,
      "BloodWorkResult_2(mm)": 29,
      "BloodWorkResult_3(U/ml)": 126,
      "BodyMassIndex(kg/m)^2": 35.2,
      "BloodWorkResult_4(U/ml)": 0.692,
      "Age (years)": 54,
      "Predicted label": "Negative",
      "Confidence_Negative": 98.72859691757037,
      "Confidence_Positive": 1.2714030824296179
    }
  ]
}
```

Figure 15

17.3 Running the FastAPI App

To run the FastAPI app, we'll use the **uvicorn** server. The app will be hosted on a specified port (e.g., 8000) and can be accessed through HTTP requests. The FastAPI app can be run using the following command:

```
uvicorn main:app --host 0.0.0.0 --port 8000 --reload
```

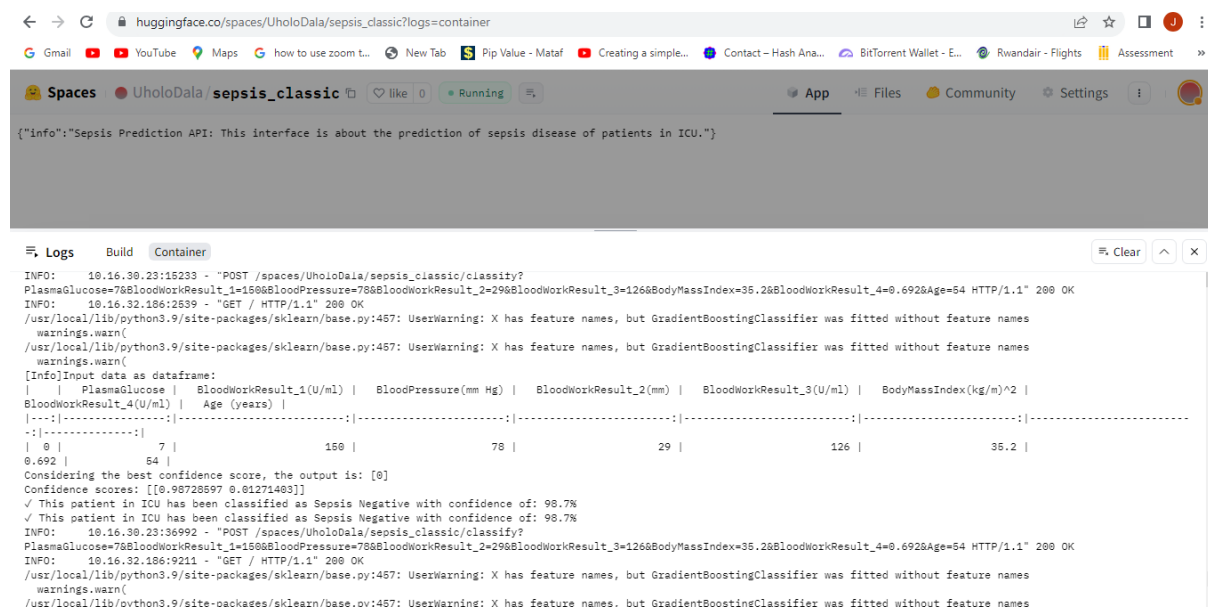
18. Containerization with Docker

Containerization provides a reliable and consistent way to package applications and their dependencies, ensuring they run consistently across different environments. Docker, a popular containerization platform, allows us to create and manage containers for our application. We'll containerize the FastAPI app using Docker to simplify deployment and ensure that the application runs smoothly in various settings.

18.1 Dockerfile

In the **Dockerfile**, we specify the base image, set the working directory, install the necessary dependencies, and copy the application code into the container. The **CMD** instruction defines the command that will be executed when the container starts. This command launches the FastAPI app using **uvicorn**. The container showing the activities of the app as it is executed in huggingface is illustrated in Figure 16.

The Docker Container in action Huggingface:



The screenshot shows a web browser window displaying the HuggingFace Spaces interface for a model named 'sepsis_classic'. The 'Logs' tab is selected, showing the container's output. The logs include an API info message, a POST request with patient data, a GET request, and a detailed prediction output. The prediction output includes a table of input features and a classification result with a confidence score of 98.7%.

```
{
  "info": "Sepsis Prediction API: This interface is about the prediction of sepsis disease of patients in ICU."
}

INFO: 10.16.30.23:15233 - "POST /spaces/UholoDala/sepsis_classic/classify?
PlasmaGlucose=78&BloodWorkResult_1=150&BloodPressure=78&BloodWorkResult_2=29&BloodWorkResult_3=126&BodyMassIndex=35.2&BloodWorkResult_4=0.692&Age=54 HTTP/1.1" 200 OK
INFO: 10.16.32.186:2839 - "GET / HTTP/1.1" 200 OK
/usr/local/lib/python3.9/site-packages/sklearn/base.py:467: UserWarning: X has feature names, but GradientBoostingClassifier was fitted without feature names
warnings.warn(
/usr/local/lib/python3.9/site-packages/sklearn/base.py:467: UserWarning: X has feature names, but GradientBoostingClassifier was fitted without feature names
warnings.warn(
[Info]Input data as dataframe:
|   | PlasmaGlucose | BloodWorkResult_1(U/ml) | BloodPressure(mm Hg) | BloodWorkResult_2(mm) | BloodWorkResult_3(U/ml) | BodyMassIndex(kg/m)^2 |
|---|---|---|---|---|---|---|
| 0 | 78 | 150 | 78 | 29 | 126 | 35.2 |
0.692 | 54 |
Considering the best confidence score, the output is: [0]
Confidence scores: [[0.98728597 0.81271403]]
✓ This patient in ICU has been classified as Sepsis Negative with confidence of: 98.7%
✓ This patient in ICU has been classified as Sepsis Negative with confidence of: 98.7%
INFO: 10.16.30.23:36992 - "POST /spaces/UholoDala/sepsis_classic/classify?
PlasmaGlucose=78&BloodWorkResult_1=150&BloodPressure=78&BloodWorkResult_2=29&BloodWorkResult_3=126&BodyMassIndex=35.2&BloodWorkResult_4=0.692&Age=54 HTTP/1.1" 200 OK
INFO: 10.16.32.186:9211 - "GET / HTTP/1.1" 200 OK
/usr/local/lib/python3.9/site-packages/sklearn/base.py:467: UserWarning: X has feature names, but GradientBoostingClassifier was fitted without feature names
warnings.warn(
/usr/local/lib/python3.9/site-packages/sklearn/base.py:467: UserWarning: X has feature names, but GradientBoostingClassifier was fitted without feature names
```

Figure 16

18.2 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. In the **docker-compose.yml** file, we define the services that compose our application. This includes the **docs** service, which builds and runs the FastAPI app using the **Dockerfile**.

19. Deploying the Application

With the FastAPI app containerized using Docker, deploying the application becomes straightforward. By executing a single command, the entire application stack, including the FastAPI app, the machine learning model, and their dependencies, can be deployed on various platforms such as local servers, cloud platforms, or Kubernetes clusters.

19.1 Running the Docker Container

To run the Docker container and deploy the FastAPI app, use the following command:

The application will be accessible at **http://localhost:8000**, and you can make prediction requests using the **/classify** endpoint.

It is also accessible in huggingface via the link: https://uholodala-sepsis-classic.hf.space/docs#/default/sepsis_classification_spaces_UholoDala_sepsis_classic_classify_post

20. Conclusion

We've gone over the entire process of creating a machine learning model for sepsis prediction and deploying it as a FastAPI application using Docker in this article. We've gone over each stage in depth, from data pre-processing and model selection to building a robust API and containerizing the application. This technique not only allows us to construct predictive models more efficiently, but it also ensures that those models are deployed seamlessly for real-world applications.

As the field of data science and machine learning evolves, using tools like FastAPI and Docker allows us to bridge the gap between model creation and practical deployment, thereby contributing to the growth of healthcare and other disciplines.

21. Future Directions

While the sepsis prediction model has been successfully constructed and deployed, there are various opportunities for additional enhancement and investigation. Enhancing the model's interpretability, incorporating real-time data streaming for dynamic predictions, and linking the program with electronic health record systems for seamless incorporation into clinical workflows are some potential future paths.

We may contribute to ongoing efforts to enhance patient outcomes by continuously refining and expanding on current work using cutting-edge machine learning approaches.

Finally, the route from data pre-processing through model deployment enabled us to bridge the gap between machine learning research and practical implementation, ultimately enabling innovation in healthcare and beyond.

I take this opportunity to thank you for reading this article and don't hesitate to contact me using the email below with your suggestions and input.

Contact:

Email: jaroyajo@gmail.com

GitHub link: https://github.com/Jauloma/Career_Accelerator_P6-ML_API.git